AN4989
Application note

STM32 microcontroller debug toolbox

Introduction

STM32 end-users are sometimes confronted with non- or partially-functional systems during product development. The best approach to use for the debug process is not always obvious, particularly for inexperienced users.

To address the above concerns, this application note provides a toolbox describing the most common debug techniques and their application to popular recommended IDEs for STM32 32-bit Arm® Cortex® MCUs. It contains detailed information for getting started as well as hints and tips to make the best use of STM32 Software Development Tools in STM32 ecosystem.

This application note applies to the microcontrollers listed in *Table 1*.

Table 1. Applicable products

Type	Sub class
Microcontrollers	STM32 High Performance MCUs STM32 Mainstream MCUs STM32 Ultra Low Power MCUs

Contents

1 Foreword 8

1.1 General information 8

1.2 Software versions 8

1.3 Acronyms 8

2 STM32 ecosystem outlines 9

2.1 Hardware development tools 9

2.1.1 Hardware kits 9

2.1.2 ST-LINK probe 16

2.1.3 Alternative debugger probes 19

2.2 Software development tools 20

2.2.1 STM32CubeMX 21

2.2.2 STM32CubeIDE 22

2.2.3 Partner IDEs 24

2.2.4 STM32CubeProgrammer 25

2.2.5 STM32CubeMonitor 27

2.3 Embedded software 28

2.4 Information and sharing 29

2.4.1 Documentation 30

2.4.2 Wiki platform 31

2.4.3 Github 31

2.4.4 ST Community 31

2.4.5 STM32 Education 32

3 Compiling for debug 33

3.1 Optimization 33

3.1.1 IAR™ EWARM 34

3.1.2 Keil® MDK-Arm µVision 35

3.1.3 STM32CubeIDE 36

3.2 Debugging information 36

3.2.1 IAR™ EWARM 37

3.2.2 Keil®-MDK-Arm µVision 38

3.2.3 STM32CubeIDE 39

4 Connecting to the board 40

4.1 SWD/JTAG pinout 40

4.2 Reset and connection mode 42

4.2.1 Presentation 42

4.2.2 IAR™ EWARM 43

4.2.3 Keil® MDK-Arm µVISION 44

4.2.4 STM32CubeIDE 48

4.2.5 STM32CubeProgrammer 49

4.3 Low-power case 50

5 Breaking and stepping into code 51

5.1 Debug support for timers, RTC, watchdog, BxCAN and I^2C 51

5.2 Debug performance 51

5.2.1 IAR™ EWARM 52

5.2.2 Keil® MDK-Arm µVISION 53

5.2.3 STM32CubeIDE 54

5.3 Secure platform limitation 55

5.3.1 RDP 55

5.3.2 PCROP 56

6 Exception handling 57

6.1 Default weak Handlers 57

6.2 Custom Handlers 58

6.3 Trapping div/0 exception 60

6.3.1 Cortex®-M0/M0+ case 60

6.3.2 Cortex®-M3/4/7 case 61

7 Printf debugging 68

7.1 STM32 Virtual COM port driver 68

7.2 Printf via UART 69

7.3 Printf via SWO/SWV 71

7.4 Semihosting 79

7.4.1 IAR™ EWARM 80

7.4.2 Keil® MDK-Arm µVISION 80

7.4.3 STM32CubeIDE 81

8 Debug through hardware exploration 87
8.1 Easy pinout probing with STMicroelectronics hardware kits 87
8.2 Microcontroller clock output (MCO) 87
8.2.1 Configuration with STM32CubeMX 87
8.2.2 HAL_RCC_MCOConfig 89
8.2.3 STM32 Series differences 90

9 Dual-Core microcontroller debugging 92

10 From debug to release 93

11 Troubleshooting 94

Appendix A Managing DBGMCU registers 95
A.1 By software 95
A.2 By debugger 96

Appendix B Use Nucleo “cuttable” ST-LINK as stand-alone VCP 106

Appendix C Managing various targets on the same PC 109

Appendix D Cortex®-M debug capabilities reminder 116
D.1 Application notes index 116

Revision history 117

List of tables

Table 1. Applicable products . . . 1
Table 2. ST-LINK software pack . . . 19
Table 3. STMicroelectronics documentation guide . . . 30
Table 4. STM32 Series RDP protection extension . . . 56
Table 5. STM32 USART vs. PC terminal WordLength example . . . 71
Table 6. Troubleshooting . . . 94
Table 7. STM32 Series vs. debug capabilties . . . 116
Table 8. STM32 Series vs. debug capabilities . . . 116
Table 9. Document revision history . . . 117

List of figures

Figure 1. STM32 ecosystem overview . . . 9
Figure 2. Development tools overview . . . 10
Figure 3. Nucleo-144, Nucleo-64 and Nucleo-32 boards . . . 10
Figure 4. STM32 Nucleo-144 structure . . . 11
Figure 5. Discovery board example . . . 12
Figure 6. EVAL board example . . . 13
Figure 7. 7X-NUCLEO-LPM01A . . . 14
Figure 8. ST-LINK, ST-LINK/V2, and ST-LINK/V2-ISOL stand-alone probes . . . 16
Figure 9. STLINK-V3SET . . . 16
Figure 10. On-board ST-LINK-V3 on Nucleo . . . 17
Figure 11. STM32 software development . . . 20
Figure 12. STM32CubeMX Configure and code generation . . . 21
Figure 13. STM32CubeIDE . . . 22
Figure 14. STM32Cube programmer . . . 26
Figure 15. STM32Cube monitor . . . 28
Figure 16. STM32CubeProjectList screenshot . . . 29
Figure 17. Get connected to STM32 world . . . 29
Figure 18. IAR™ EWARM Optimization option . . . 34
Figure 19. Keil® µVision Code Optimization option . . . 35
Figure 20. STM32CubeIDE optimization level setting . . . 36
Figure 21. IAR™ EWARM Generate debug Information option . . . 37
Figure 22. Keil® Debug Information option . . . 38
Figure 23. STM32CubeIDE debug information option . . . 39
Figure 24. SWD pins PA13 and PA14 in Reset state under STM32CubeMX . . . 40
Figure 25. SWD pins PA13 and PA14 in Reserved but inactive state under STM32CubeMX . . . 41
Figure 26. SWD pins PA13 and PA14 in Active State under STM32CubeMX . . . 41
Figure 27. Reset Mode in IAR8.10: screenshot . . . 43
Figure 28. Connect and Reset option Keil® . . . 44
Figure 29. Keil® hotplug step1 . . . 45
Figure 30. Keil® hotplug step2 . . . 46
Figure 31. Keil® hotplug step3 . . . 47
Figure 32. Select Generator Options Reset Mode . . . 48
Figure 33. STM32CubeProgrammer Reset mode . . . 49
Figure 34. STM32CubeProgrammer Connection mode . . . 49
Figure 35. IAR™ EWARM ST-LINK SWD Speed setting . . . 52
Figure 36. Keil® SWD Speed Setting . . . 53
Figure 37. Access to Generator Options in STM32CubeIDE V2.0.0 . . . 54
Figure 38. Asking for Handler code generation . . . 58
Figure 39. Keil® Access to Show Caller Code in Contextual menu . . . 60
Figure 40. Cortex®-M3 SCB_CCR Description . . . 61
Figure 41. Cortex-M3 SCB_CFSR Description . . . 61
Figure 42. IAR™ EWARM exception handling . . . 62
Figure 43. Keil® System Control and Configure . . . 63
Figure 44. Keil® Fault Reports . . . 64
Figure 45. STM32CubeIDE SCB register access . . . 65
Figure 46. Fault Analyzer in STM32CubeIDE . . . 66
Figure 47. Virtual COM port on Windows® PC . . . 68

Figure 48. USART Pinout configuration with STM32CubeMX . . . 69
Figure 49. USART2 setting with STM32CubeMX . . . 70
Figure 50. SWO Pin configuration with STM32CubeMX . . . 72
Figure 51. Semihosting/SWO configuration with IAR™ EWARM . . . 73
Figure 52. IAR™ EWARM SWO Clock setting . . . 74
Figure 53. SWO configuration with Keil® . . . 75
Figure 54. Access to SWV in Keil® . . . 75
Figure 55. Enable SWD in STM32CubeIDE . . . 77
Figure 56. Enable SWV ITM Data Console in STM32CubeIDE . . . 78
Figure 57. Enable ITM stimulus Port 0 in STM32CubeIDE . . . 79
Figure 58. Start Trace button in STM32CubeIDE . . . 79
Figure 59. Semihosting configuration in IAR™ EWARM . . . 80
Figure 60. Properties for semihosting in STM32CubeIDE- Source Location . . . 81
Figure 61. Properties for semihosting in STM32CubeIDE- Librairies . . . 82
Figure 62. Properties for semihosting in STM32CubeIDE . . . 82
Figure 63. Semihosting in STM32CubeIDE – Debug configuration . . . 84
Figure 64. Semihosting in STM32CubeIDE – Startup . . . 85
Figure 65. Semihosting in STM32CubeIDE – Run . . . 86
Figure 66. MCO pin selection in STM32CubeMX . . . 87
Figure 67. MCO alternate pin highlight exemple with L073 . . . 88
Figure 68. MCO Multiplexer in STM32CubeMX Clock Configuration Pane . . . 89
Figure 69. STM32F4/F7 dual MCO capabilities . . . 91
Figure 70. DBMCU Register LL Library Functions . . . 95
Figure 71. DBGMCU_CR HAL Library Functions . . . 96
Figure 72. Access to DBGMCU register with IAR™ EWARM . . . 97
Figure 73. EWARM C-SPY® Macro script setting . . . 98
Figure 74. Accessing DBGMCU register in Keil® MDK-Arm µVision (1/2) . . . 99
Figure 75. Accessing DBGMCU register in Keil® MDK-Arm µVision (2/2)) . . . 100
Figure 76. Keil® Initialization script setting . . . 101
Figure 77. Access to Generator Options in STM32CubeIDE V2.0.0 . . . 102
Figure 78. Generator Options debug MCU in STM32CubeIDE . . . 103
Figure 79. Access to DBGMCU settings with STM32CubeIDE V1.3.0 . . . 104
Figure 80. Runtime R/W access to DBGMCU register with SSTM32CubeIDE . . . 105
Figure 81. ST-LINK cuttable part of Nucleo . . . 106
Figure 82. Using ST-LINK stand-alone part of Nucleo-L476RG as VCP . . . 107
Figure 83. Virtual COM port on PC side . . . 108
Figure 84. STM32CubeProgrammer target selection pick list . . . 109
Figure 85. Getting target ST-LINK S/N from the console . . . 110
Figure 86. IAR™ EWARM Debug Probe Selection pop-up window . . . 110
Figure 87. IAR™ EWARM Debug Probe Selection with nickname . . . 111
Figure 88. Probe selection prompt setting on IAR™ EWARM . . . 111
Figure 89. Keil® ST-LINK selection . . . 112
Figure 90. Error message for multiple ST-LINK detected in STM32CubeIDE . . . 113
Figure 91. Forcing specific ST-LINK S/N with STM32CubeIDE with OpenOCD option . . . 114
Figure 92. Forcing specific ST-LINK S/N with STM32CubeIDE with ST-LINK GDB server . . . 115

1 Foreword

1.1 General information

This document applies to STM32 32-bit Arm® Cortex® MCUs.

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

1.2 Software versions

The various examples in this application note are illustrated on basis of the following versions of the tools:

- IAR™ EWARM: V8.32.3
- Keil® MDK-Arm µVision: V5.26
- STM32CubeIDE: V1.3.0
- STM32CubeProg: V2.2.1

1.3 Acronyms

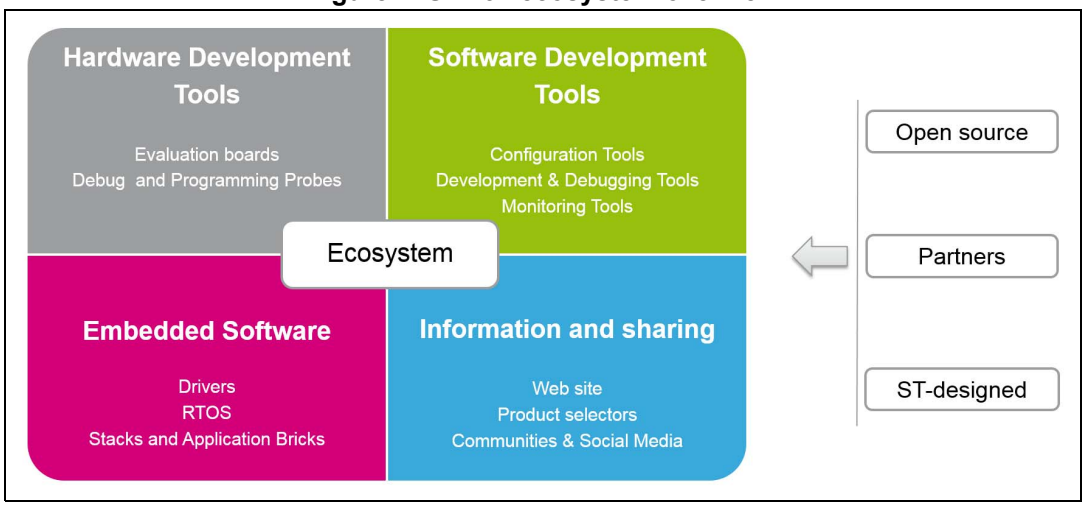
- AN: Application note
- CMSIS: Cortex microcontroller software interface standard
- HAL: Hardware abstraction layer (software library)
- IDE: Integrated development environment
- JTAG: Joint Test Action Group
- MCO: Microcontroller clock output
- MCU: Microcontroller unit
- NVIC: Nested vector interrupt controller
- PM: Programming manual
- RM: Reference manual
- SB: Solder bridge
- SWD: Serial wire debug
- SWO: Single wire output
- SWV: Single wire viewer
- VCP: Virtual COM port

2 STM32 ecosystem outlines

STMicroelectronics and its partners are providing a full hardware and software ecosystem to support rapid evaluation, prototyping, and productizing of complete systems using STM32 microcontrollers.

As presented in *Figure 1*, the ecosystem is composed of all the collaterals required to develop a project with STM32.

Figure 1. STM32 ecosystem overview



This chapter provides a global overview of the main elements composing the ecosystem, outlining debug features and useful pointers, in order to guide the user among available resources.

2.1 Hardware development tools

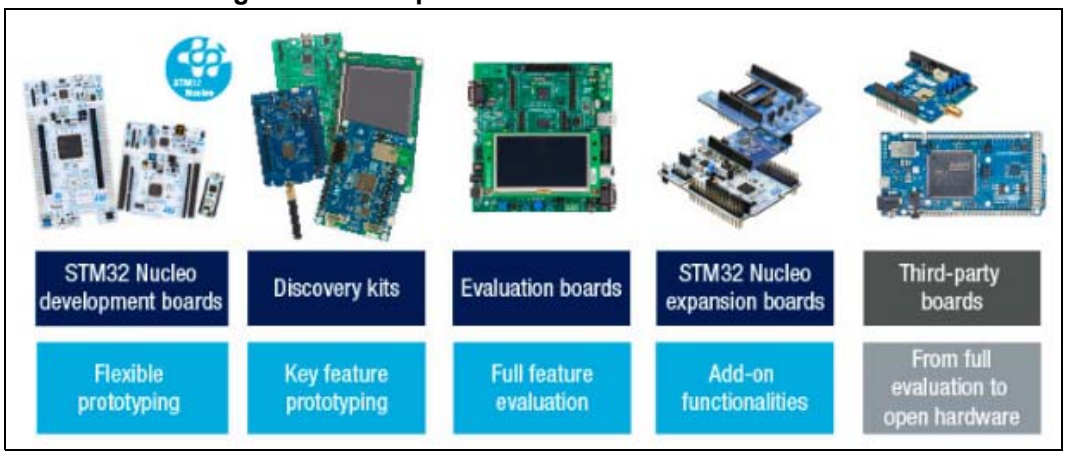
This section introduces the range of available development tools from hardware kits to ST-LINK probes and alternative debugger interfaces.

2.1.1 Hardware kits

This section lists the hardware kits provided by STMicroelectronics for STM32-based development:

- Nucleo boards
- Discovery kits
- Evaluation boards (EVAL)
- STM32 Nucleo expansion
- Third-party boards

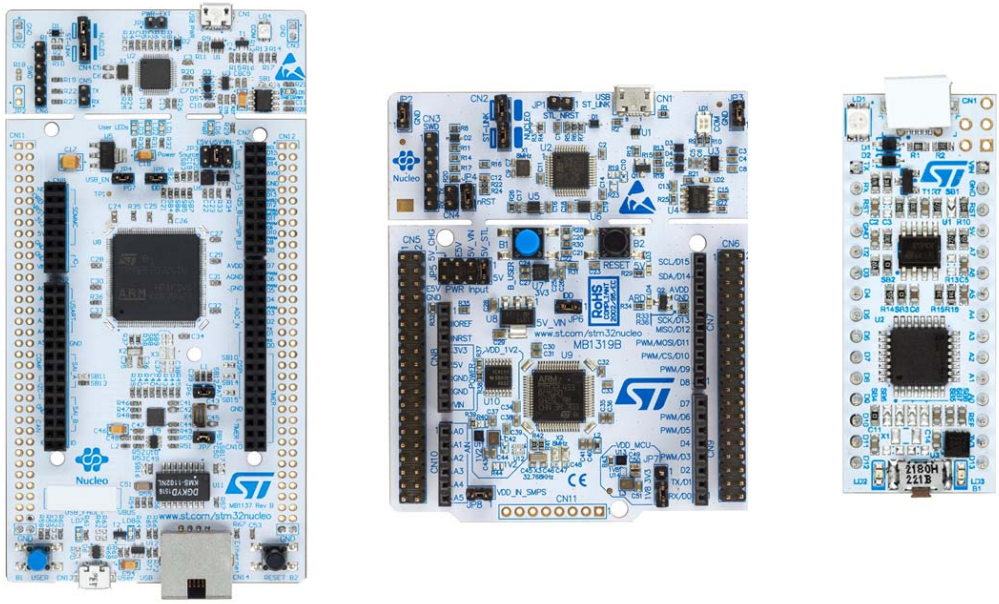
Figure 2. Development tools overview



STM32 Nucleo

STM32 Nucleo boards are affordable solutions for user willing to try out new ideas and to quickly create prototypes based on STM32 MCU.

Figure 3. Nucleo-144, Nucleo-64 and Nucleo-32 boards



STM32 Nucleo boards feature the same connectors. They can easily be extended with a large number of specialized application hardware add-ons.

Note: Nucleo-144 boards include ST Zio connector, which is an extension of ARDUINO® Uno rev3, and ST morpho connector.

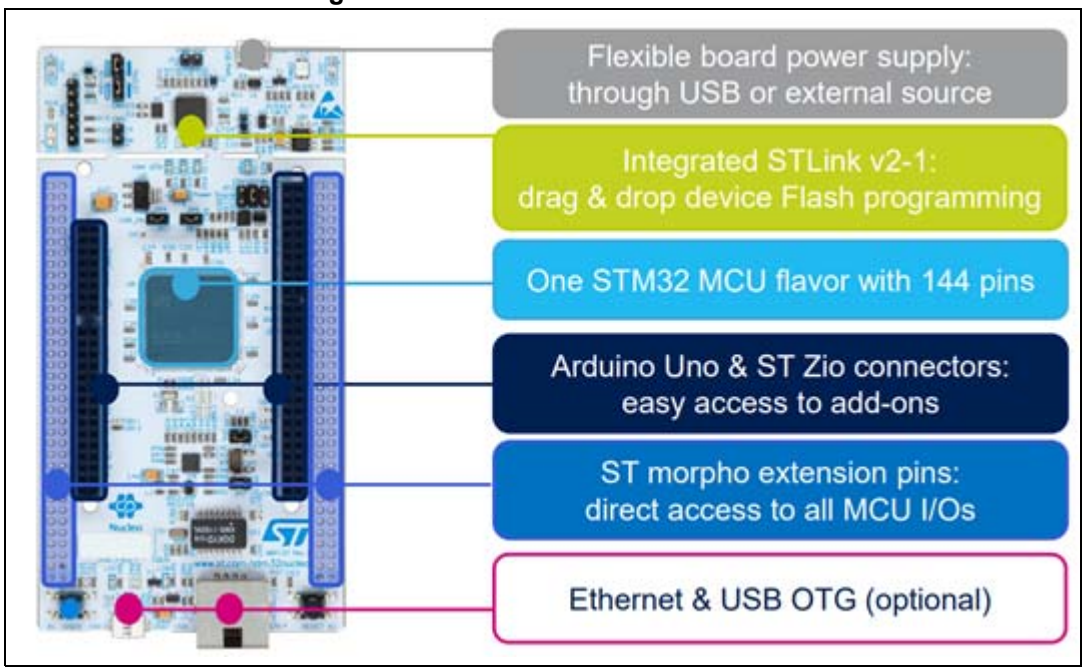
Nucleo-68 board and Nucleo-64 boards include ARDUINO® Uno rev3 and ST morpho connectors.

Nucleo-32 boards include ARDUINO® Nano connectors.

All STM32 Nucleo boards integrate an ST-LINK debugger/programmer, so there is no need for a separate probe.

The figure below shows an example of STM32 Nucleo structure

Figure 4. STM32 Nucleo-144 structure

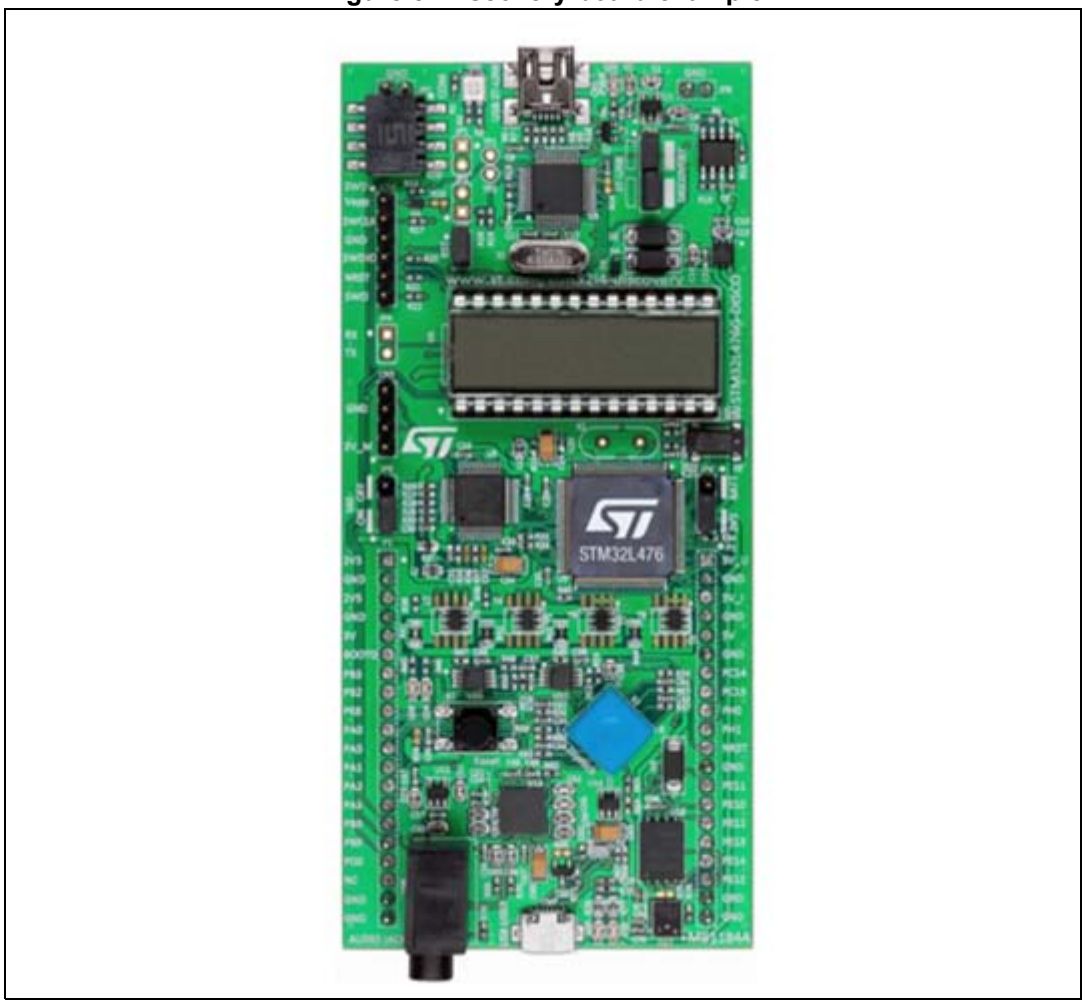


A complete description of the embedded ST-LINK features is provided in *Section 2.1.2: ST-LINK probe on page 16*. Additional information and access to Nucleo boards complete documentation sets are available at *www.st.com*.

Discovery kits

STM32 Discovery kits are a cheap and complete solution for the evaluation of the outstanding capabilities of STM32 MCUs. They carry the necessary infrastructure for demonstration of specific device characteristics, the HAL library, and comprehensive software examples allow to fully benefit from the devices features and added values.

Figure 5. Discovery board example



Extension connectors give access to most of the device's I/Os and make the connection of add-on hardware possible.

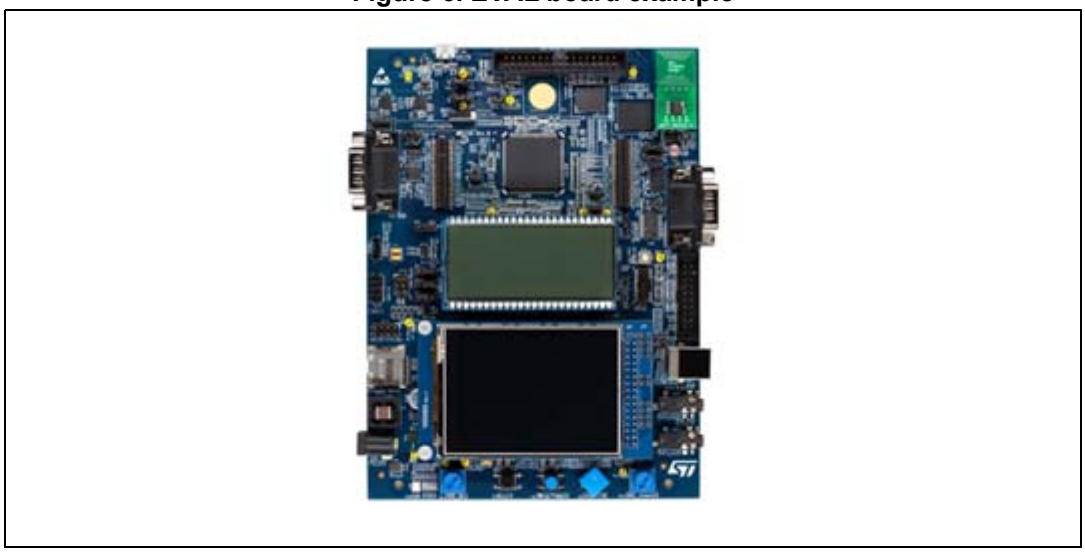
With the integrated debugger/programmer the Discovery kits are ideal for prototyping.

A complete description of the embedded ST-LINK features is provided in *Section 2.1.2: ST-LINK probe on page 16*. Additional information and access to Discovery kits complete documentation sets are available at *www.st.com*.

Evaluation boards

STM32 MCU EVAL boards have been designed as a complete demonstration and development platform for the Arm® Cortex® STM32 MCUs.

Figure 6. EVAL board example



They carry external circuitry, such as transceivers, sensors, memory interfaces, displays and many more. The EVAL boards can be considered as a reference design for application development.

EVAL boards have integrated ST-LINK (USB Type-B connector). For complete description of the embedded ST-LINK features refer to *Section 2.1.2: ST-LINK probe*.

EVAL board has direct access to JTAG/Traces signal through dedicated Arm® JTAG 20-pin connector allowing advanced debug (ETM). For usage of ETM traces refer to *Section 2.1.3: Alternative debugger probes on page 19*.

The usage of a stand-alone probe may require some jumper and solder bridge adaptation from default. Refer to the specific board user manual.

For further information and access to complete documentation visit www.st.com/stm32evaltools.

STM32 nucleo expansion

STM32 Nucleo expansion boards carry all the required components to Evaluate ST devices to be used together with an STM32 MCU.

Build STM32-based applications leveraging functionality and performance of ST's device portfolio.

The expansion boards are equipped with standardized interconnections, such as an ARDUINO Uno R3 connector, or a Morpho connector for a higher level of connectivity

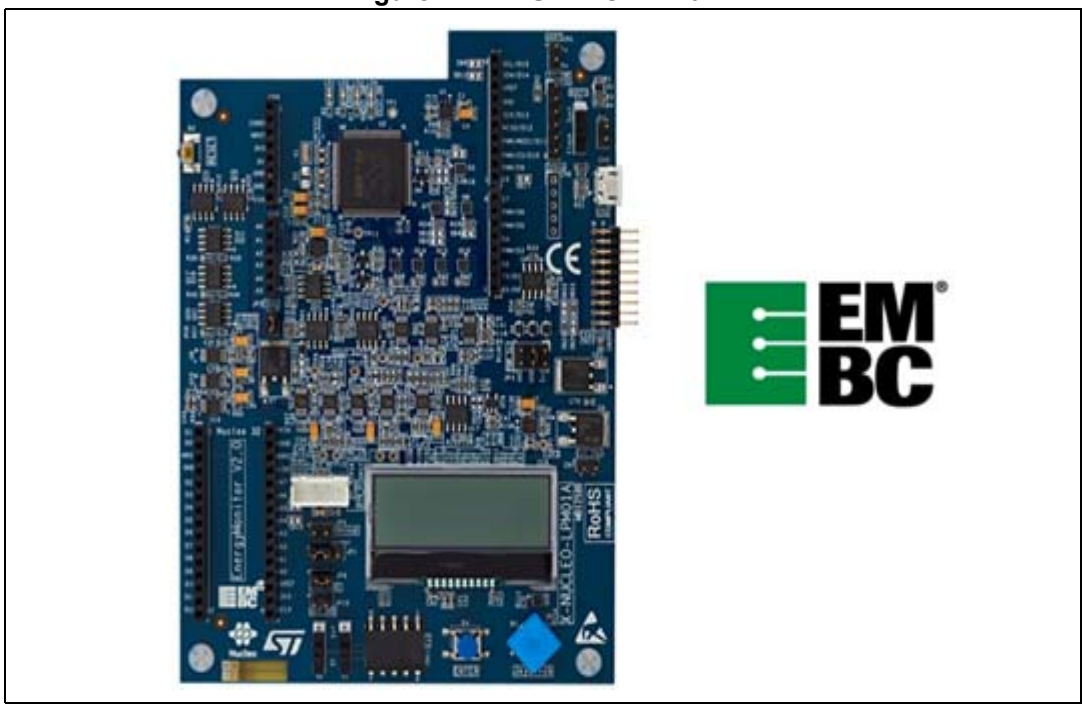
Each expansion board is supported by STM32-based software modules.

The combination of STM32 Nucleo boards and expansion boards is a unified scalable approach with unlimited possibilities for application development, prototyping or product evaluation.

X-NUCLEO-LPM01A

This board is an example of STM32 nucleo expansion.

Figure 7. 7X-NUCLEO-LPM01A



The X-NUCLEO-LPM01A is a 1.8 V to 3.3 V programmable power supply source with advanced power consumption measurement capability.

It performs consumption averaging (static measurement up to 200 mA) as well as real-time analysis (dynamic measurement up to 50 mA with 100 kHz bandwidth).

The X-NUCLEO-LPM01A operates either in standalone mode (using its LCD, joystick and button to display static measurements), or in controlled mode connected to host PC via

USB (using the STM32CubeMonPwr software tool with its comprehensive graphical user interface).

It can be used to supply and measure the consumption of STM32 Nucleo-32, Nucleo-64, Nucleo-68 or Nucleo-144 boards, using ARDUINO connectors.

Alternatively, it supplies and measures the consumption of any target connected by wires via the basic connector.

KEY FEATURES

- STM32L496VGT6 microcontroller featuring Arm® Cortex®-M4 core at 80 MHz / 100 DMIPS and three 12-bit ADC at 5 Msps
- Programmable voltage source from 1.8 V to 3.3 V
- Static current measurement from 1 nA to 200 mA
- Dynamic measurements:
  - 100 kHz bandwidth, 3.2 Msps sampling rate
  - Current from 100 nA to 50 mA
  - Power measurement from 180 nW to 165 mW
  - Energy measurement computation by power measurement time integration
  - Execution of EEMBC ULPMark™ tests
- Mode standalone:
  - Monochrome LCD, 2 lines of 16 characters with backlight
  - 4-direction joystick with selection button
  - Enter and Reset push-buttons
- Mode controlled:
  - Connection to a PC through USB FS Micro-B receptacle
  - Command line (Virtual COM port) or
  - STM32CubeMonitor-Power PC tool.
- Four status LEDs
- Target board connectors:
  - ARDUINO® Uno and Nano connectors
  - Basic connector (white): 4 wires
- Flexible input power-supply options:
  - USB Micro-B (VBUS)
  - External power connector (7 V to 10 V)
  - ARDUINO Uno and Nano connectors (pin 5 V)

2.1.2 ST-LINK probe

The ST-LINK is the JTAG/Serial Wire Debug (SWD) interface used to communicate with any STM32 microcontroller located on an application board.

It is available as:

- Stand-alone in-circuit debugger
- Embedded in all STM32 hardware kits (Nucleo boards, Discovery kits, EVAL boards)

ST-LINK/V2 and ST-LINK-V3 are the main used versions.

Figure 8 shows ST-LINK/V2 and ST-LINK/V2- ISOL stand-alone probes on the right.

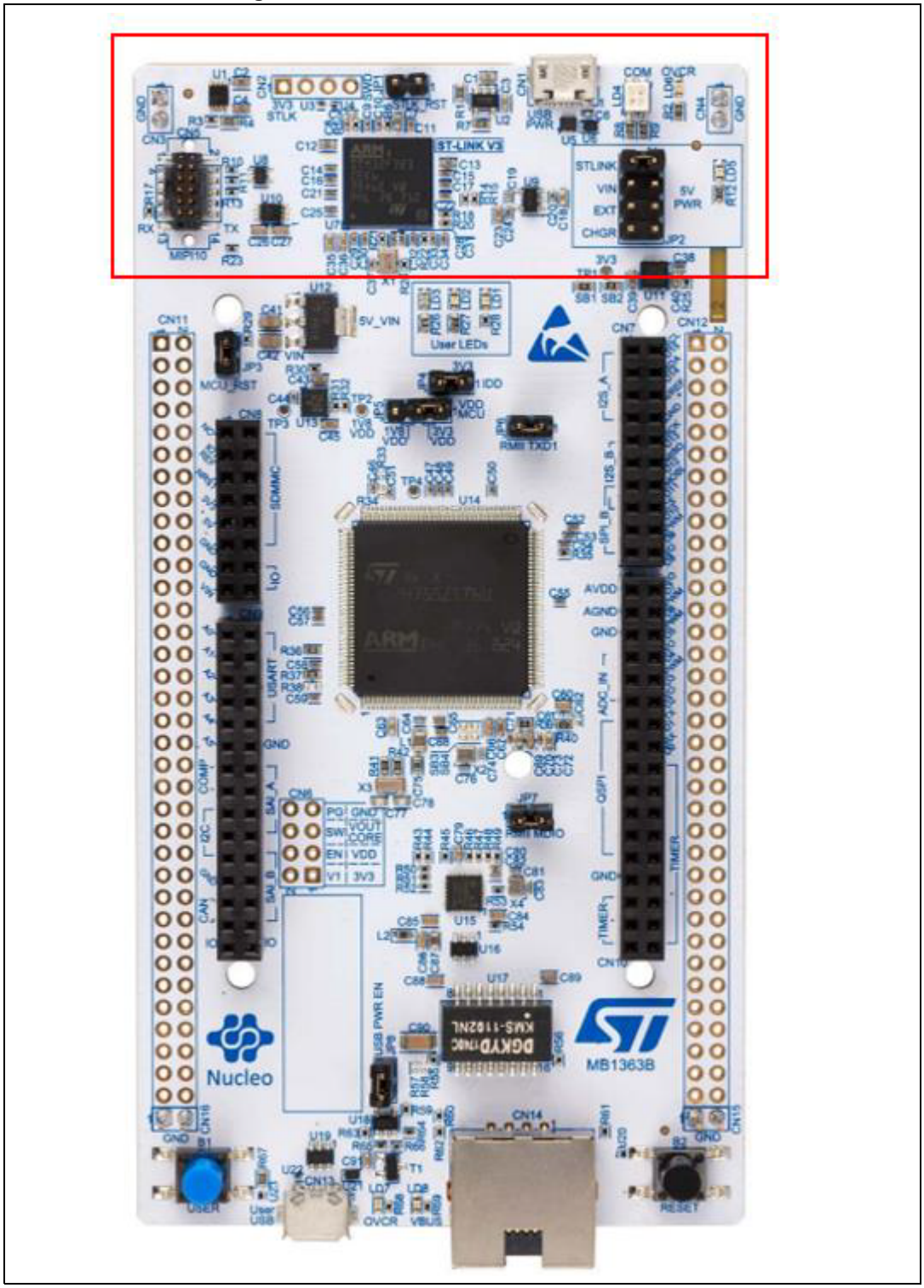
Figure 8. ST-LINK, ST-LINK/V2, and ST-LINK/V2-ISOL stand-alone probes

Figure 9 shows the last STLINK-V3SET version.

Figure 9. STLINK-V3SET

Figure 10 shows an example of an embedded ST-LINK/V2 as part of a Nucleo board.

Figure 10. On-board ST-LINK-V3 on Nucleo



ST-LINK/V2 basic features

- 5 V power supplied by a USB connector
- USB 2.0 full-speed-compatible interface
- USB standard Type-A to Mini- B cable
- JTAG/serial wire debug (SWD) specific features:
  - 1.65 V to 3.6 V application voltage supported on the JTAG/SWD interface and 5 V tolerant inputs
  - JTAG cable for connection to a standard JTAG 20-pin pitch 2.54 mm connector
  - JTAG supported
  - SWD and serial wire viewer (SWV) communication supported
- Device Firmware Upgrade (DFU) feature supported
- Status LED which blinks during communication with the PC
- Operating temperature 0 °C to 50 °C
- 1000 V rms high-isolation voltage (ST-LINK/V2-ISOL only)

Embedded versions usually supports the following additional features:

- Virtual COM port interface on USB. (VCP)
- Mass storage interface on USB

The availability of these additional features depends on software version.

ST-LINK-V3 basic features

- Stand-alone probe with modular extensions
- Self-powered through a USB connector (Micro-B)
- USB 2.0 high-speed compatible interface
- Direct firmware update support (DFU)
- JTAG / serial wire debugging (SWD) specific features:
  - 3 V to 3.6 V application voltage support and 5 V tolerant inputs
  - Flat cables STDC14 to MIPI10 / STDC14 / MIPI20 (connectors with 1.27 mm pitch)
  - JTAG communication support
  - SWD and serial wire viewer (SWV) communication support
- Virtual COM port (VCP) specific features:
  - 3 V to 3.6 V application voltage support on the UART interface and 5 V tolerant inputs
  - VCP frequency up to 15 MHz
  - Available on STDC14 debug connector (not available on MIPI10)
- Multi-path bridge USB to SPI/UART/I2C/CAN/GPIOs specific features:
  - 3 V to 3.6 V application voltage support and 5 V tolerant inputs
  - Signals available on adapter board only (MB1440)
- Drag-and-drop flash programming of binary files
- Two-color LEDs: communication, power

Note: The STLINK-V3SET product does not provide power supply to the target application.

In order to identify the ST-LINK version on a board and the related features associated with it, please refer STMicroelectronics technical note *Overview of the ST-LINK embedded in STM32 MCU Nucleo, Discovery Kits and Eval Boards* (TN1235).

On-board ST-LINK does not support JTAG port.

Note: For Nucleo and Discovery, JTAG port signal can be wired through Morpho / ARDUINO® connectors. On EVAL boards, there is a dedicated 20-pin connector.

The use of ST-LINK requires the software packages listed in *Table 2*.

Table 2. ST-LINK software pack

Part Number	Description
STSW-LINK007	ST-LINK, ST-LINK/V2, ST-LINK/V2-1, STLINK-V3 boards firmware upgrade
STSW-LINK009	ST-LINK, ST-LINK/V2, ST-LINK/V2-1 USB driver signed for Windows® 7, Windows® 8, Windows® 10
STLINK-V3-BRIDGE	Software API compatible with the bridge interface of STLINK-V3

Note: STSW-LINK007 is included in STSW-LINK004.

STSW-LINK009 is included in most IDE installation packages (IAR Systems®, Keil®, STM32CubeIDE) and tools.

Tip: It is recommended to use the latest firmware version of the on-board ST-LINK interface.
Firmware upgrade can be performed thanks to the STM32CubeProgrammer (refer to *Section 2.2.4: STM32CubeProgrammer*) or STM32CubeIDE.

2.1.3 Alternative debugger probes

J-LINK (Segger), I-Jet™ (*IAR Systems*®), and U-LINK (Keil®) are the most common alternatives providing features equivalent to the ones provided by ST-LINK.

For most advanced debugging needs, requiring heavy traffic or ETM port tracing, ST recommends using:

- U-Link Pro in combination with Keil® MDK-Arm µVISION
- I-Jet™ Trace in combination with IAR™ EWARM

For a complete catalog of solutions, refer to *www.st.com*.

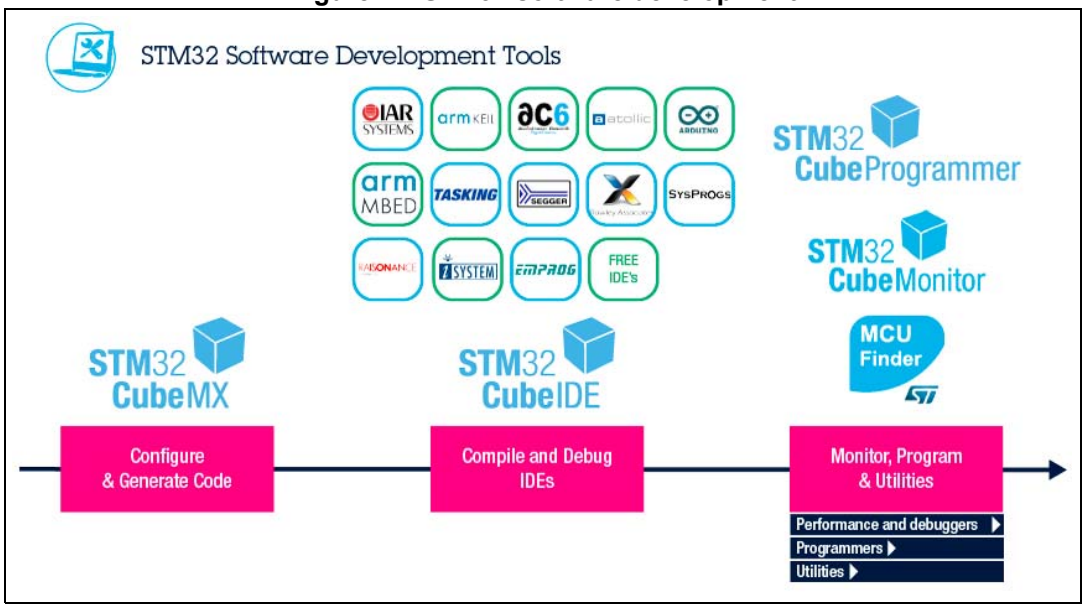
2.2 Software development tools

The STM32 family of 32-bit Arm® Cortex®-M core-based microcontrollers is supported by a complete range of software tools.

It encompasses traditional integrated development environments - IDEs with C/C++ compilers and debuggers from major third parties that are complemented with tools from ST allowing to configure and initialize the MCU or monitor its behavior in run time.

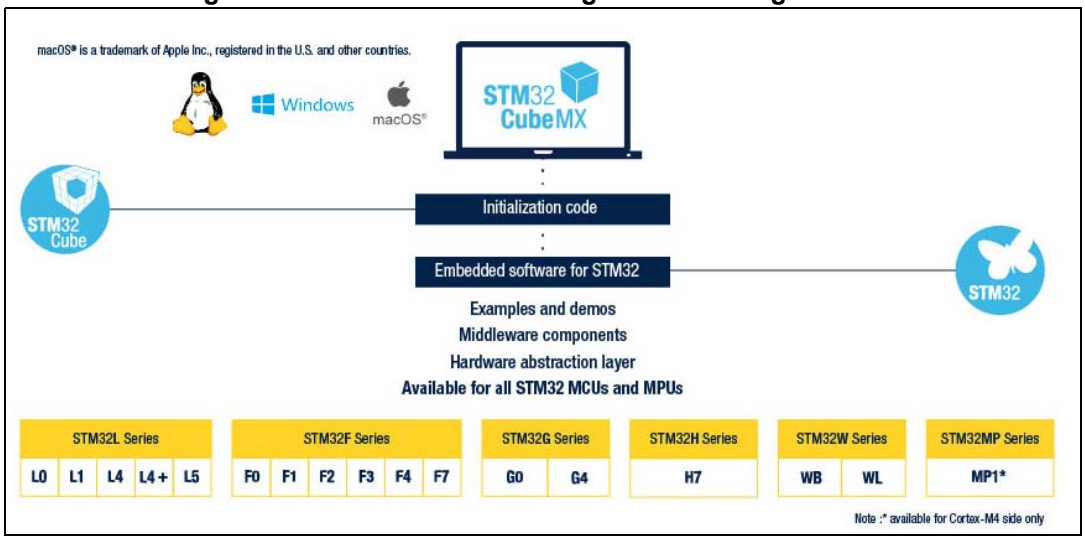
It offers a complete flow, from configuration up to monitoring as illustrated in *Figure 11*.

Figure 11. STM32 software development



2.2.1 STM32CubeMX

Figure 12. STM32CubeMX Configure and code generation



STM32CubeMX is a graphical tool that allows to easily configure STM32 microcontrollers and to generate the corresponding initialization C code through a step-by-step process.

The first step consists in selecting the STM32 microcontroller that matches the required set of peripherals. MCU can be selected as stand-alone for custom PCB (MCU Selector) or pre-integrated into one of STMicroelectronics hardware kit (Board Selector)

In the second step, the user must configure each required embedded software thanks to a pinout-conflict solver, a clock-tree setting helper, a power-consumption calculator, and a utility performing MCU peripheral configuration (GPIO, USART, and others) and middleware stacks (USB, TCP/IP, and others).

Finally, the user launches the generation of the initialization C code based on the selected configuration. This code is ready to be used within several development environments. The user code is kept at the next code generation.

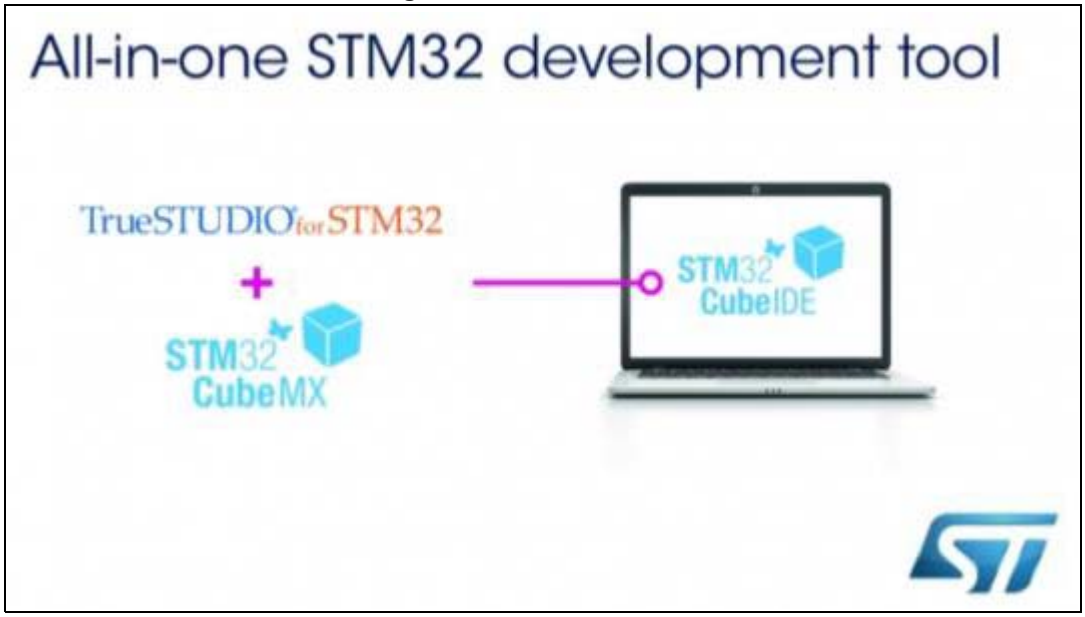
Key features

- Intuitive STM32 microcontroller selection
- Rich graphical user interface configuration:
  - Pinout with automatic conflict resolution
  - Clock tree with dynamic validation of configuration
  - Peripherals and middleware functional modes and initialization with dynamic validation of parameter constraints
  - Power consumption calculation for a user-defined application sequence
- C code project generation covering STM32 microcontroller initialization compliant with *IAR Systems*®, Keil® and GCC compilers.
- Available as a standalone software running on Windows®, Linux®, and macOS® operating systems, or through Eclipse plug-in

2.2.2 STM32CubeIDE

STM32CubeIDE is an all-in-one multi-OS development tool, which is part of the STM32Cube software ecosystem.

Figure 13. STM32CubeIDE



STM32CubeIDE is an advanced C/C++ development platform with peripheral configuration, code generation, code compilation, and debug features for STM32 microcontrollers. It is based on the *Eclipse*®/CDT framework and GCC toolchain for the development, and GDB for the debugging. It allows the integration of the hundreds of existing plugins that complete the features of the *Eclipse*® IDE. STM32CubeIDE integrates all STM32CubeMX functionalities to offer all-in-one tool experience and save installation and development time. After the selection of an empty STM32 MCU or MPU, or preconfigured microcontroller from the selection of a board, the project is created, and initialization code generated. At any time during the development, the user can return to the initialization and configuration of the peripherals or middleware and regenerate the initialization code with no impact on the user code. STM32CubeIDE includes build and stack analyzers that provide the user with useful information about project status and memory requirements. STM32CubeIDE also includes standard and advanced debugging features including views of CPU core registers, memories, and peripheral registers, as well as live variable watch, Serial Wire Viewer interface, or fault analyzer.

Key features

- Integration of STM32CubeMX that provides services for:
  - STM32 microcontroller selection
  - Pinout, clock, peripheral, and middleware configuration
  - Project creation and generation of the initialization code
- Based on *Eclipse*®/CDT, with support of *Eclipse*® add-ons, GNU C/C++ for Arm® toolchain and GDB debugger
- Additional advanced debug features including:
  - CPU core, peripheral register, and memory views
  - Live variable watch view
  - System analysis and real-time tracing (SWV)
  - CPU fault analysis tool
- Support of ST-LINK (STMicroelectronics) and J-Link (SEGGER) debug probes
- Import project from Atollic® TrueSTUDIO® and AC6 System Workbench for STM32 (STM32CubeIDE)
- Multi-OS support: Windows®, Linux®, and macOS®, 64-bit versions only

2.2.3 Partner IDEs

In this application note, all topics are declined for the three main IDEs:

1. IAR™ EWARM
2. Keil® MDK-Arm µVISION

IAR™ EWARM

The IAR Embedded Workbench® for Arm® (IAR™ EWARM) is a software development suite delivered with ready-made device configuration files, flash loaders and 4300 example projects included. IAR Systems® and STMicroelectronics closely cooperate in supporting 32-bit Arm® Cortex®-M based microcontrollers.

Key Features

- Key components:
  - Integrated development environment with project management tools and editor
  - Highly optimizing C and C++ compiler for Arm®
  - Automatic checking of MISRA C rules (MISRA C:2004)
  - Arm® EABI and CMSIS compliance
  - Extensive HW target system support
  - Optional I-jet™ and JTAGjet™-Trace in-circuit debugging probes
  - Power debugging to visualize power consumption in correlation with source code
  - Run-time libraries including source code
  - Relocating Arm® assembler
  - Linker and librarian tools
  - C-SPY® debugger with Arm® simulator, JTAG support and support for RTOS-aware bugging on hardware
  - RTOS plugins available from IAR Systems® and RTOS vendors
  - Over 3100 sample projects for EVAL boards from many different manufacturers
  - User and reference guides in PDF format
  - Context-sensitive on-line help
- Chip-specific support:
  - 4300 example projects included for STMicroelectronics EVAL boards
  - Support for 4 Gbyte applications in Arm® and Thumb® mode
  - Each function can be compiled in Arm® or Thumb® mode
  - VFP Vector Floating Point co-processor code generation
- Intrinsic NEON™ support
- ST-LINK and ST-LINK/V2 support

This product is supplied by a third party not affiliated to ST. For the latest information on the specification, refer to the IAR Systems® web site at http://www.iar.com.

Keil® MDK-Arm µVision

The MDK-Arm-STM32 is a complete software development environment for Cortex®-M microcontroller-based devices. It includes the µVision IDE/Debugger, Arm®C/C++ compiler and essential middleware components. The STM32 peripherals can be configured using STM32CubeMX and the resulting project exported to MDK-Arm.

Free MDK-Arm licenses can be activated for both STM32F0 and STM32L0 Series using the following Product Serial Number (PSN): U1E21-CM9GY-L3G4L.

This product is supplied by a third party not affiliated to ST. For the latest information on the specification refer to the third party's website: http://www2.keil.com/stmicroelectronics-stm32.

Key Features

- Complete support for Cortex®-M devices
- Arm® C/C++ compilation toolchain
- uVision IDE, debugger and simulation environment
- CMSIS Cortex® Microcontroller Software Interface Standard compliant
- ST-LINK support
- Multi-language support: English, Chinese, Japanese, Korean

2.2.4 STM32CubeProgrammer

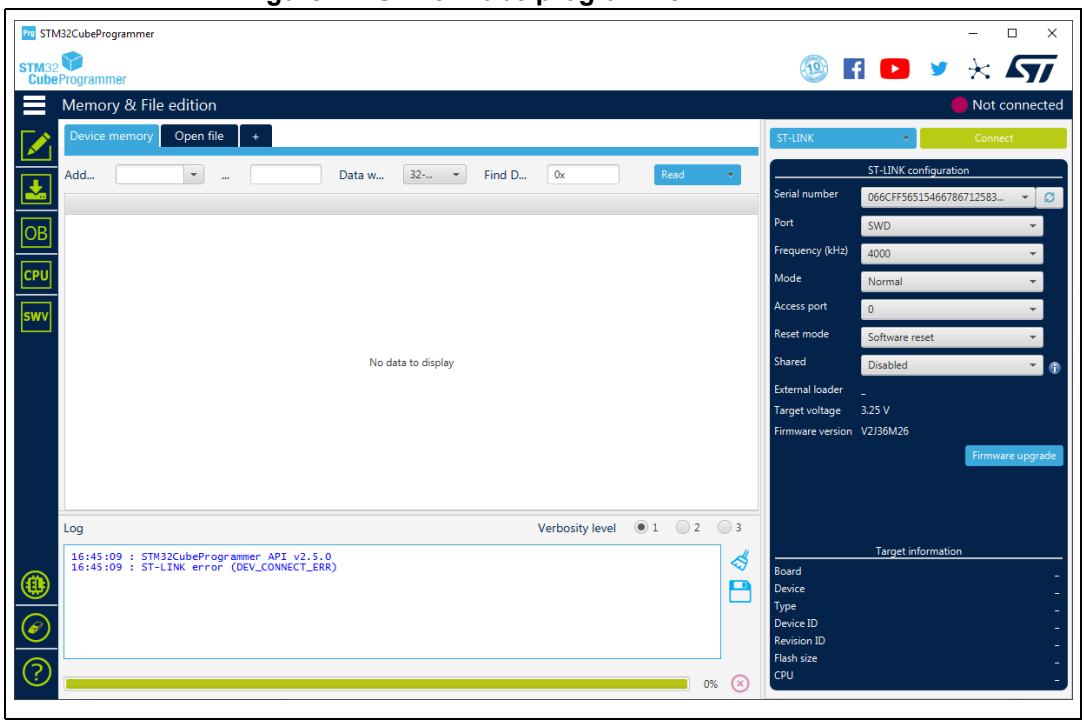
STM32CubeProgrammer (STM32CubeProg) is an all-in-one multi-OS software tool for programming STM32 products.

It provides an easy-to-use and efficient environment for reading, writing and verifying device memory through both the debug interface (JTAG and SWD) and the bootloader interface (UART, USB DFU, I^2C, SPI, and CAN). STM32CubeProgrammer offers a wide range of features to program STM32 internal memories (such as Flash, RAM, and OTP) as well as external memories. STM32CubeProgrammer also allows option programming and upload, programming content verification, and programming automation through scripting. STM32CubeProgrammer is delivered in GUI (graphical user interface) and CLI (command-line interface) versions.

Key Features

- Erases, programs, views and verifies the content of the device Flash memory
- Supports Motorola S19, Intel HEX, ELF, and binary formats
- Supports debug and bootloader interfaces:
  - ST-LINK debug probe (JTAG/SWD)
  - UART, USB DFU, I2C, SPI, and CAN bootloader interfaces
- Programs, erases and verifies external memories, with examples of external Flash loaders to help users to develop loaders for specific external memories
- Automates STM32 programming (erase, verify, programming, configuring option bytes)
- Allows OTP memory programming
- Supports the programming and configuring of option bytes
- Offers a command-line interface for automation through scripting
- ST-LINK firmware update
- Enables secure firmware creation using the STM32 Trusted Package Creator tool
- Supports OTA programming for the STM32WB Series
- Multi-OS support: Windows, Linux, macOS®

Figure 14. STM32Cube programmer



2.2.5 STM32CubeMonitor

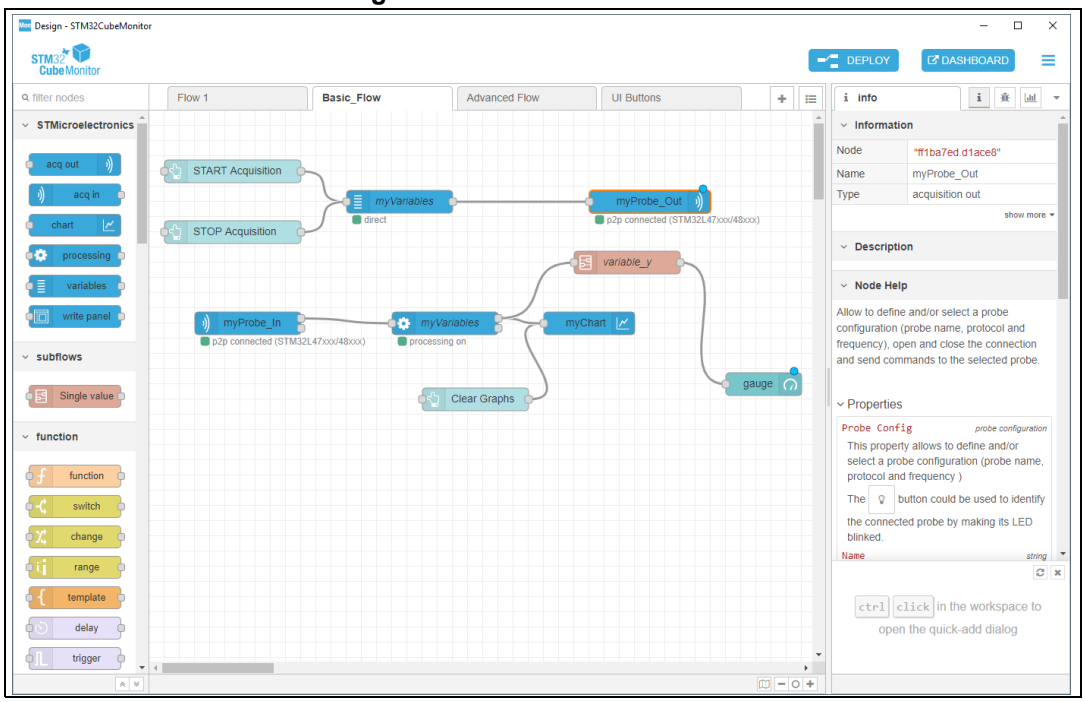
The STM32CubeMonitor family of tools helps to fine-tune and diagnose STM32 applications at run-time by reading and visualizing their variables in real-time. In addition to specialized versions (power, RF, USB-PD), the versatile STM32CubeMonitor provides a flow-based graphical editor to build custom dashboards simply, and quickly add widgets such as gauges, bar graphs and plots. With non-intrusive monitoring, STM32CubeMonitor preserves the real-time behavior of applications, and perfectly complements traditional debugging tools to perform application profiling.

With remote monitoring and native support for multi-format displays, STM32CubeMonitor enables users to monitor applications across a network, test multiple devices simultaneously, and perform visualization on various host devices such as PCs, tablets, or mobile phones. Moreover, with the direct support of the Node-RED® open community, STM32CubeMonitor allows an unlimited choice of extensions to address a wide diversity of application types.

Key Features

- Graphical flow-based editor with no programming needed to build dashboards
- Connects to any STM32 device via ST-LINK (SWD or JTAG protocols)
- Reads and writes variables on-the-fly from and to the RAM in real time while the target application is running
- Parses debugging information from the application executable file
- Direct acquisition mode or snapshot mode
- Trigger to focus on application behaviors of interest
- Enables to log data into a file and replay for exhaustive analysis
- Delivers customized visualization with configurable display windows (such as curves and boxes) and a large choice of widgets (such as gauges, bar graphs and plots)
- Multi-probe support to monitor multiple targets simultaneously
- Remote monitoring with native support of multi-format displays (PCs, tablets, mobile phones)
- Direct support of the Node-RED® open community
- Multi-OS support: Windows®, Linux® Ubuntu® and macOS®

Figure 15. STM32Cube monitor



2.3 Embedded software

The STM32Cube embedded software libraries provides:

- The HAL hardware abstraction layer, enabling portability between different STM32 devices via standardized API calls
- The Low-Layer (LL) APIs, a light-weight, optimized, expert oriented set of APIs designed for both performance and runtime efficiency
- A collection of Middleware components, like RTOS, USB library, file system, TCP/IP stack, Touch sensing library or Graphic Library (depending on the MCU series)
- A complete set of code examples running on STMicroelectronics boards: STM32 Nucleo, Discovery kits and EVAL boards

Tip: There is a fair chance that a Cube Project example matches the project in design. At project start or if an issue is met, it is worth browsing the complete project list package content available in CubeLibraryFolder\Projects\ STM32CubeProjectsList.html (refer to *Figure 16*).

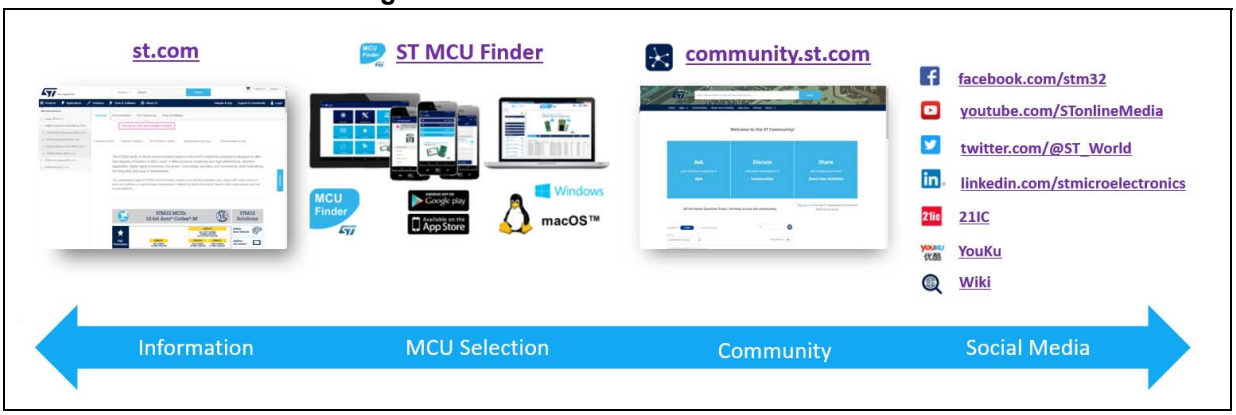
Figure 16. STM32CubeProjectList screenshot

Level	Module Name	Project Name	Description	STM32L4R9I-EVAL	STM32L476G-EVAL
	UART	LPUART_WakeUpFromStop	Configuration of an LPUART to wake up the MCU from Stop mode when a given stimulus is received.	X	-
		UART_HyperTerminal_DMA	UART transmission (transmit/receive) in DMA mode between a board and an HyperTerminal PC application.	-	X
		UART_LowPower_HyperTerminal_DMA	LPUART transmission (transmit/receive) in DMA mode between a board and an HyperTerminal PC application.	X	-
		UART_Printf	Re-routing of the C library printf function to the UART.	X	X
		UART_TwoBoards_ComDMA	UART transmission (transmit/receive) in DMA mode between two boards.	-	-
		UART_TwoBoards_ComIT	UART transmission (transmit/receive) in Interrupt mode between two boards.	-	-
		UART_TwoBoards_ComPolling	UART transmission (transmit/receive) in Polling mode between two boards.	-	-
		UART_WakeUpFromStop	Configuration of an UART to wake up the MCU from Stop 1 mode when a given stimulus is received.	-	-

2.4 Information and sharing

STMicroelectronics offers a very complete and wide range of solution on the web to get connected to STM32 World.

Figure 17. Get connected to STM32 world



2.4.1 Documentation

Several types of documentation are available on *www.st.com*. *Table 3* provides a reminder of the main technical documents with a short description of their contents.

Table 3. STMicroelectronics documentation guide

Acronym	Name	Content
DB	Data Brief	Preliminary Product Specification before complete maturity
DS	Data sheet	Product Specifications, Hardware feature and Electrical Characteristics (Pinout/Alternate function definition table, Memory Map, Electrical Characterization etc.)
RM	Reference manual	How to use the targeted microcontroller series, memory and peripherals.(registers details, default/reset value etc.)
AN	Application note	“How to make” guide helping to achieve a specific application with the targeted MCU.
UM	User manual	“How To Use” guide for a specific software of hardware product (board, software tools etc.)
TN	Technical note	Very brief document addressing single technical aspect. Can be seen as a complement of AN or UM documents
ES	Errata sheet	Contained known issues and device limitation.
PM	Programmer manual	Target software developer with a full description of the STM32 Cortex®-M processor programming model, instruction set and core peripherals
RN	Release note	It describes new features, known limitations and corrections on a specific software release for an evaluation board, a reference design, a software or programming or debugging tool.

Tip: The MCU Finder application can be useful for document access and bookmarking in addition to its primary usage for identifying the suitable STM32 product. The MCU Finder application is available for use on PC, smartphone, and tablet. More information is available on *www.st.com*.

Trick: When an Internet search engine is used to get access to STMicroelectronics documents, it is advised to search with an explicit mention of STMicroelectonics web site so that references to genuine documents are obtained. In the Google Toolbar™ search bar, the following syntax can be used:
“[Document reference or key word]” site:*www.st.com* filetype:pdf

2.4.2 Wiki platform

STMicroelectronics offers the wiki platform to help developers to use its STM32 devices.

This user guide aims at assisting developers to use STM32 MCU devices from STMicroelectronics.

It contains articles to discover STM32 MCUs, as well as examples and helps:

- The Getting started to easily start with an STM32 MCU board in
- The Development zone to help developing applications and share projects
- The Software tools zone for a first contact with the tools
- The Training zone to get trained on STM32 MCUs

The home page of wiki is https://wiki.st.com/stm32mcu/index.php/

Note: Two wiki spaces are currently proposed: one dedicated to STM32 Microcontroller (MCU) products and one for STM32 Microprocessor (MPU) products.
The focus of this application note is on the those dedicated to STM32 microcontrollers.

2.4.3 Github

STMicroelectronics is now publishing STM32Cube embedded software on GitHub, the popular cloud-based service. The aim is to open up the STM32 integrated software offering to collaborative and community-friendly development and take advantage of faster and more efficient distribution of updates.

Publishing all STM32Cube original code through GitHub lets users of more than 1000 STM32 Arm® Cortex®-M microcontroller variants and heterogeneous Cortex-M/-A microprocessors to easily store, manage, track, and control their code. GitHub features such as Pull requests promote co-development, enabling the community to propose alternate solutions and new features taking advantage of GitHub's change-handling structures. In addition, GitHub Issues - the privileged communication channel between developers - lets users submit problems, share solutions, and contribute to fixes.

The move to GitHub also ensures that the developers can receive all software updates as soon as they are published, more quickly than traditional means of updating MCU packages.

All current STM32Cube MCU packages are already online, as well as hardware abstraction layer (HAL) code and MCU-independent CMSIS drivers. The remaining STM32Cube embedded-software components will be added over the coming months.

All STM32Cube embedded software on GitHub is available free of charge. Please visit https://github.com/STMicroelectronics for more information or to get started.

2.4.4 ST Community

STMicroelectronics new community is now live and ready for receiving questions, sharing projects and collaborating among fellow community members. The focus is on collaboration because the primary purpose of this community is to share with peers and help them in a transparent way that showcases the world of STMicroelectronics products, activities and achievements.

The home page of ST Community is https://community.st.com/welcome.

For any problem met, it is interesting to first browse the STM32 Forum for related topics and eventually to post a new one if no relevant thread is found.

2.4.5 STM32 Education

STM32 education material is available on-line at *www.st.com* (search for STM32 Education).

This site provides free educational resources created by STMicroelectronics engineers for bringing an STM32 project to life.

On this site, a user learns at his own pace, watches classes as per his own schedule, anytime, anywhere, on any device, or apply to one of the live learning sessions led by STMicroelectronics experts at a nearby location.

Content:

- Online Training
- MOOC
- Videos
- Webinar
- Textbooks
- ST training courses
- Partner training courses

3 Compiling for debug

This chapter reviews the various options for debug-friendly compiling solutions.

3.1 Optimization

Compiler are usually configured by default to optimize performance and/or code size. In most cases, this reduces or even prevents program debugging.

The most common symptoms resulting from code optimization are:

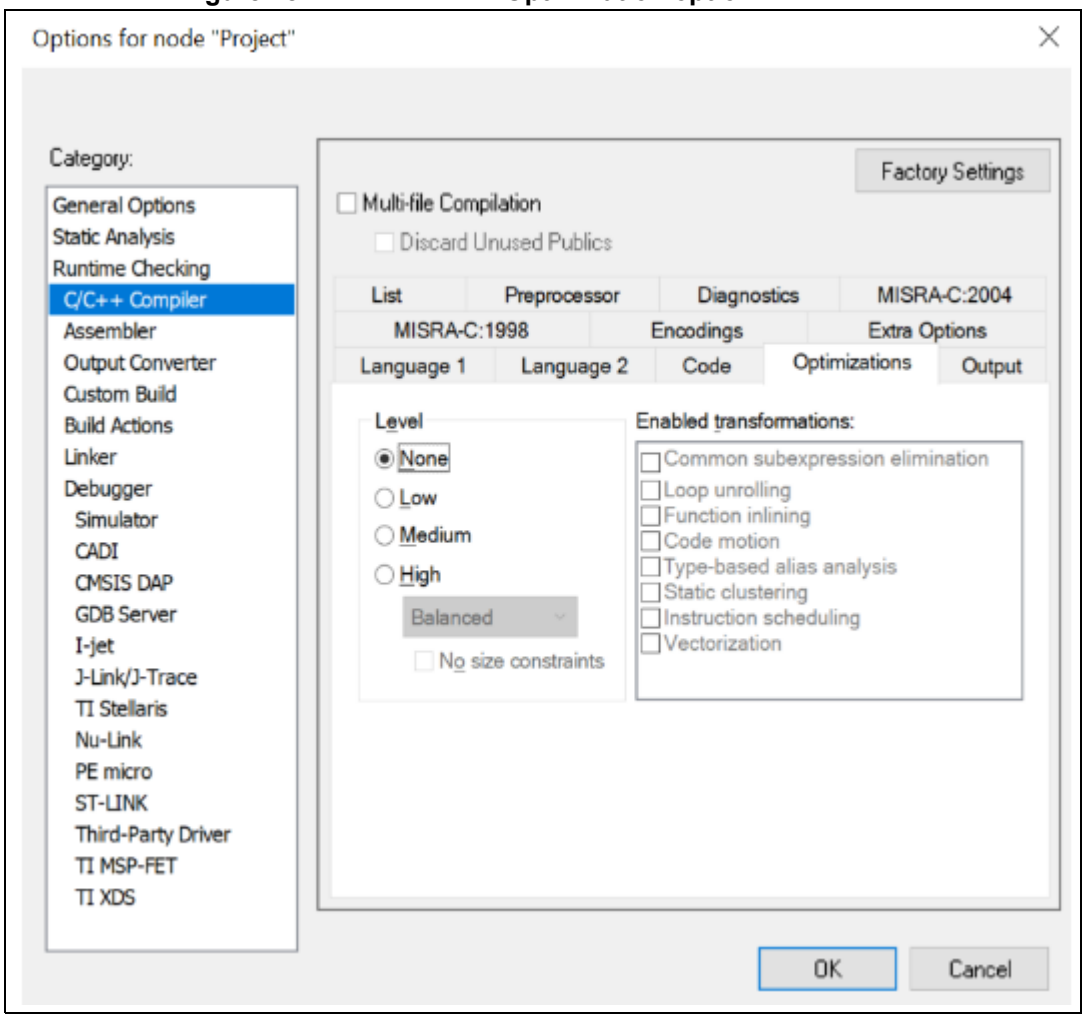
- Problem to set or reach a breakpoint. Some lines are not accessible.
- Impossibility to evaluate a variable (watch feature).
- Inconsistency while stepping (what I get, is not what I see).

Therefore, for efficient debugging it is recommended to modify the code optimization option.

3.1.1 IAR™ EWARM

In ***Project->option->C/C++Compiler->Optimization***

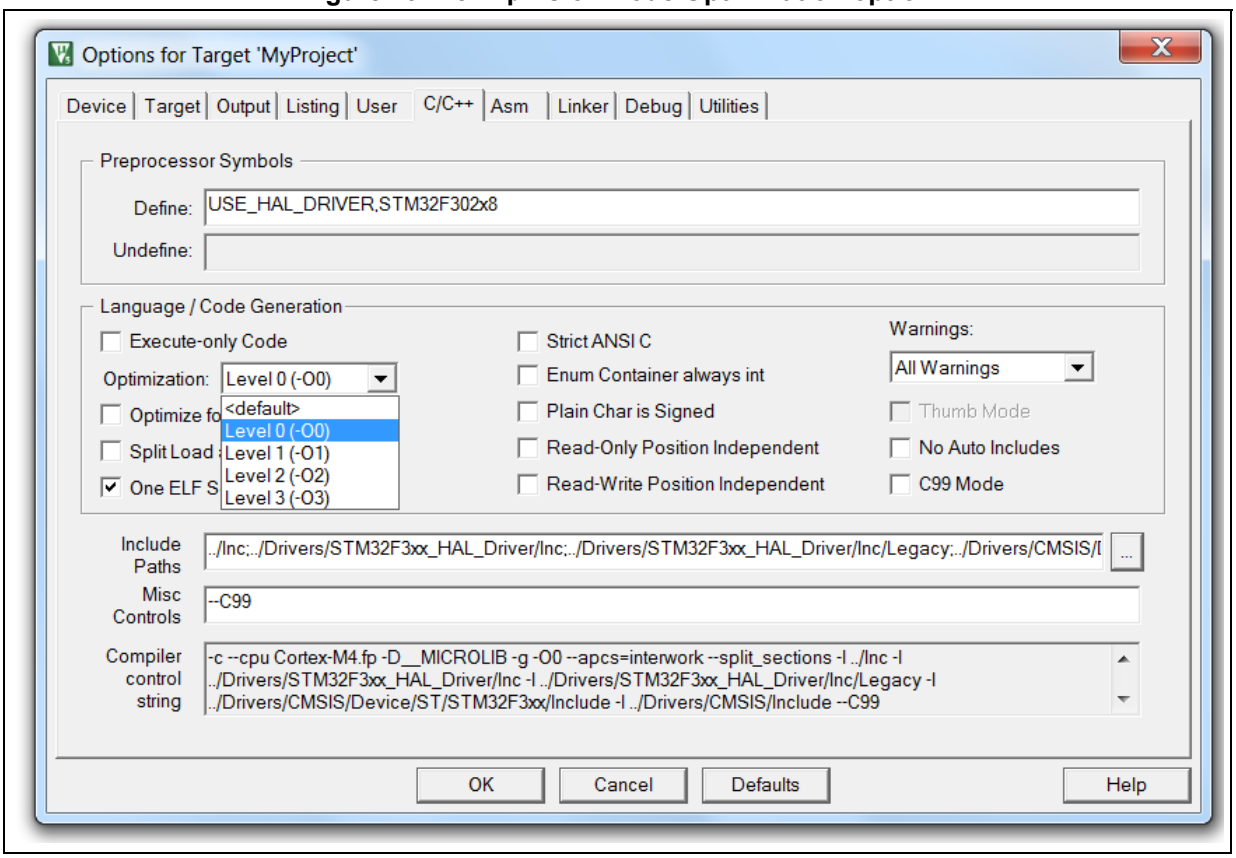
Figure 18. IAR™ EWARM Optimization option



3.1.2 Keil® MDK-Arm µVision

In Project ***Option for Target->C/C++->Optimization***

Figure 19. Keil® µVision Code Optimization option



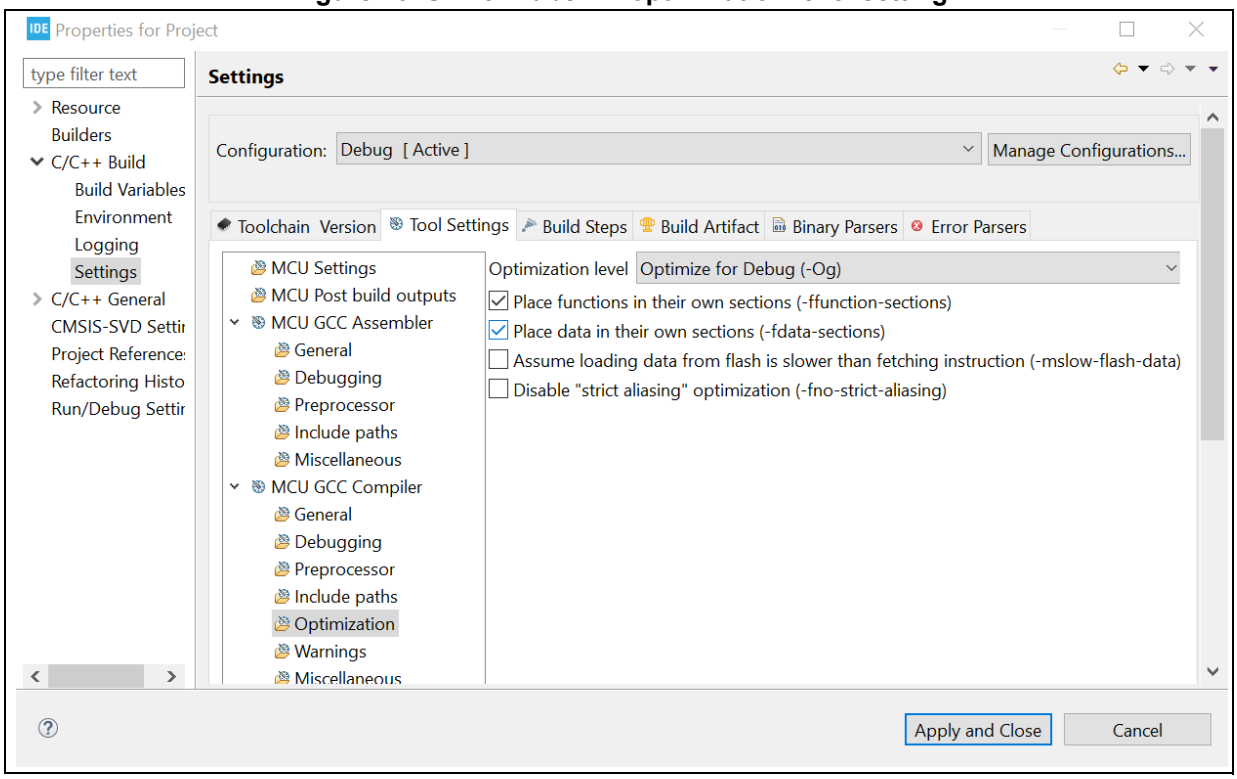
Keil® documentation suggests that Level1 (-O1) can be a suitable alternative for debug.

Refer to www.keil.com support page *Compiler optimization levels and the debug view* for details.

3.1.3 STM32CubeIDE

In project ***Properties->Settings->Tool Settings->MCU GCC Compiler->Optimization***

Figure 20. STM32CubeIDE optimization level setting



gcc also provides the -Og option:

-Og enables optimizations that do not interfere with debugging. It offers a reasonable level of optimization while maintaining fast compilation and a good debugging experience.

3.2 Debugging information

Debugging information is generated by the compiler together with the machine code. It is a representation of the relationship between the executable program and the original source code. This information is encoded into a pre-defined format and stored alongside the machine code.

Debugging information is mandatory to set breakpoint or get the content of a variable.

This chapter presents the location of the Debugging Information related option in IAR Systems®, Keil®, and STM32CubeIDE.

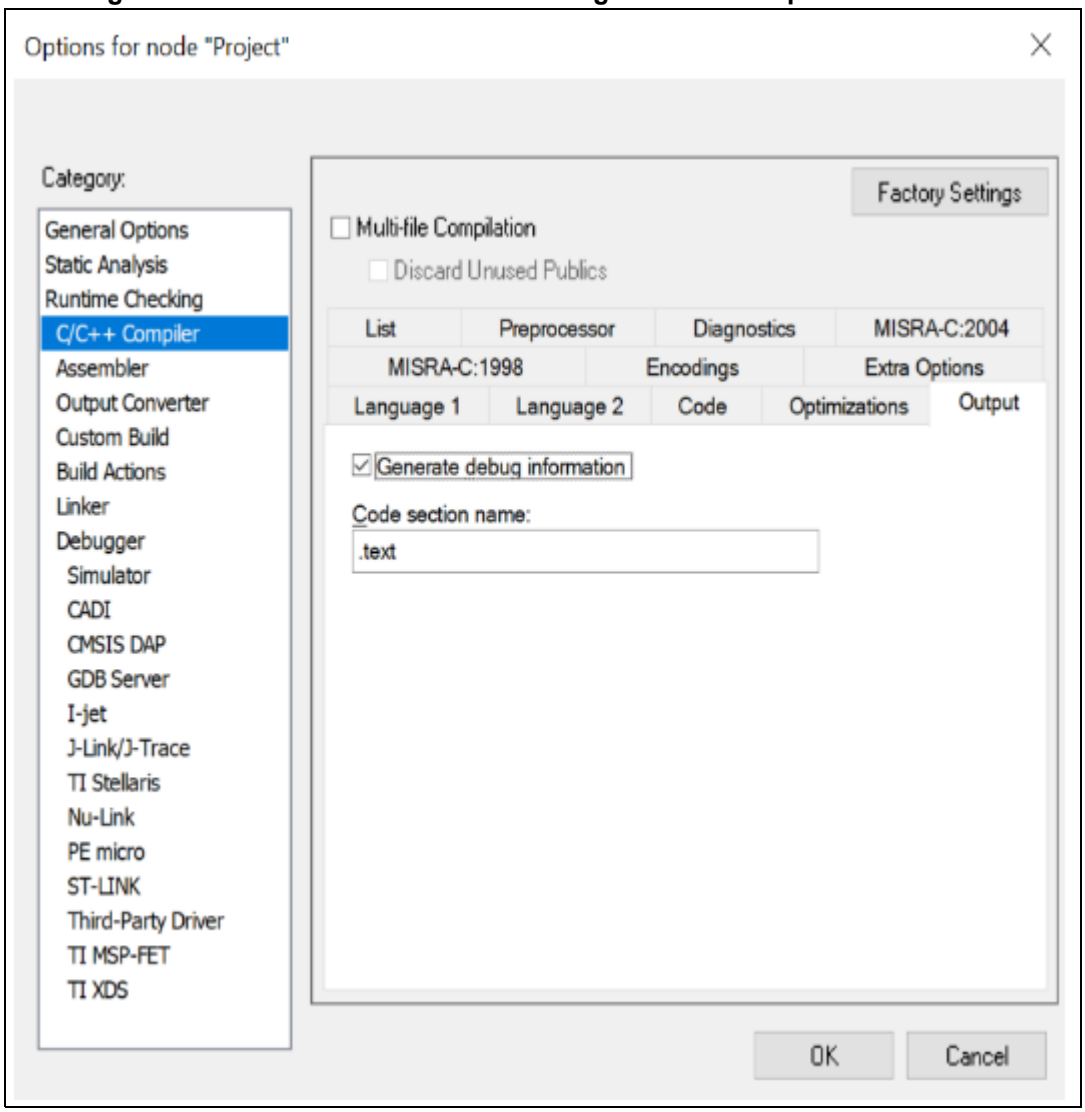
3.2.1 IAR™ EWARM

“Generate debug information” option tick box is accessible in

Project -> Options -> C/C++ Compiler -> Output Pane

It is set by default.

Figure 21. IAR™ EWARM Generate debug Information option



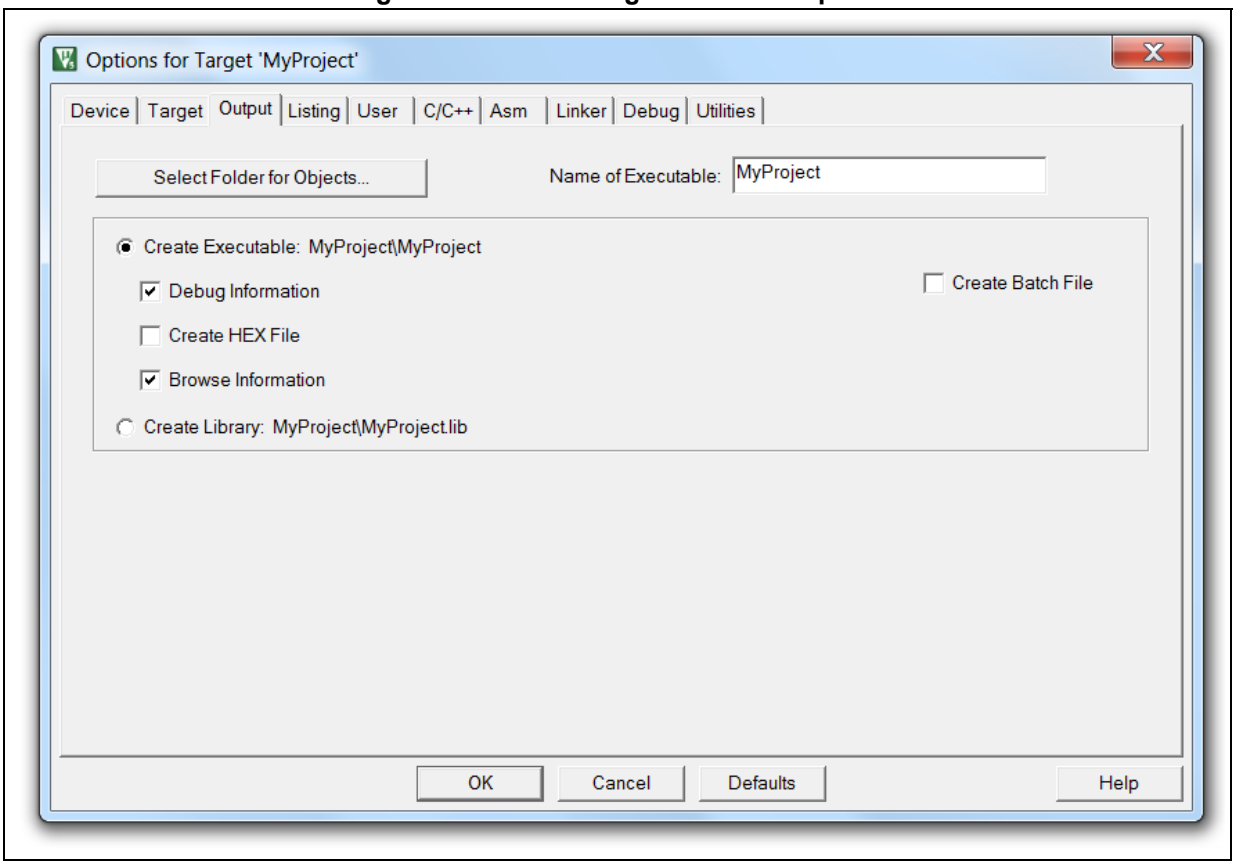
3.2.2 Keil®-MDK-Arm µVision

Debug Information Tick box is accessible in

Project -> Options -> Output Pane.

It is set by default.

Figure 22. Keil® Debug Information option

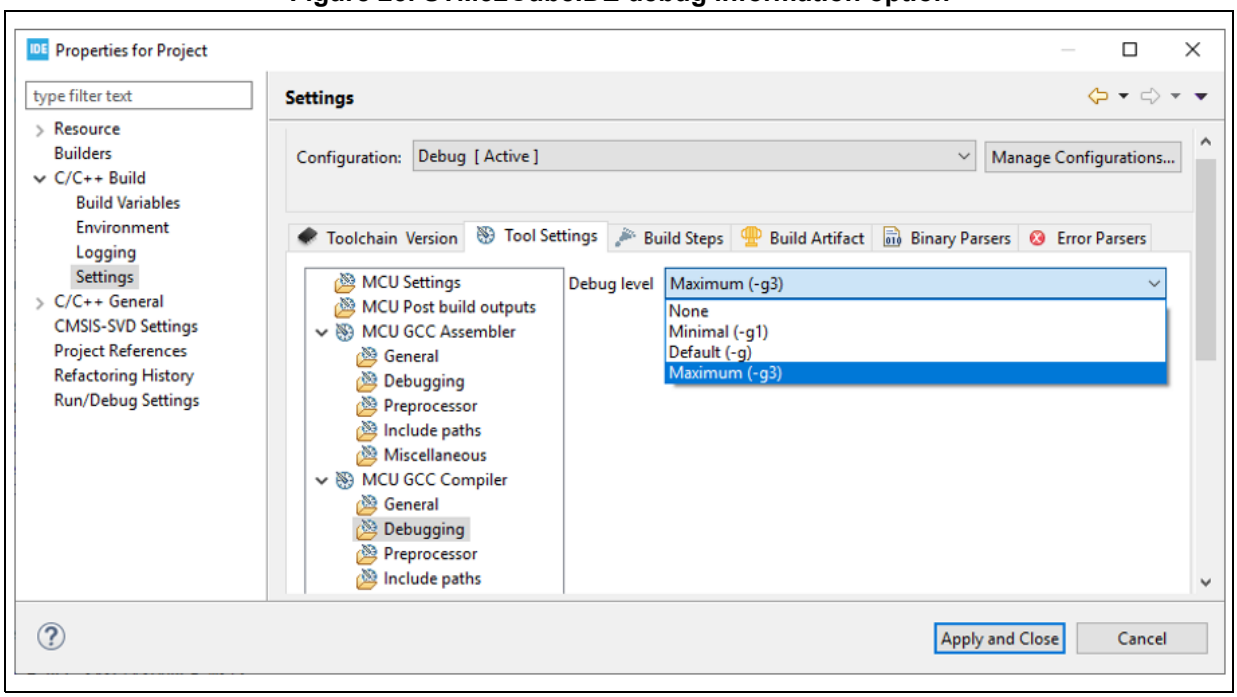


3.2.3 STM32CubeIDE

Option to manage Debugging Information are in

Properties -> C/C++ Build -> Settings -> Tool Settings -> Debugging.

Figure 23. STM32CubeIDE debug information option



Debug Level can be set among four levels:

- None: Level 0 produces no debug information at all;
- Minimal (-g1): Level 1 produces minimal information, enough for making backtraces in parts of the program for which no debug is planned. This includes descriptions of functions and external variables, and line number tables, but no information about local variables.
- Default (-g): Produce debugging information in the operating system's native format (stabs, COFF, XCOFF, or DWARF). GDB can work with this debugging information.
- Maximal (-g3): Level 3 includes extra information, such as all the macro definitions present in the program. Some debuggers support macro expansion when -g3 is used.

The same pane contains the options to add profiling information.

For further information, refer to *Section 3.1 Option Summary* available at http://gcc.gnu.org.

4 Connecting to the board

The way IDEs get connected to the boards is not always known. In case of trouble, a basic knowledge about this topic can save time in identifying and fixing the issue.

This chapter intends to provide the minimal set of information in order to prevent or quickly fix issues related to connection.

4.1 SWD/JTAG pinout

On STMicroelectronics hardware kits, SWD must be made available for connection with ST-LINK.

SWD is always mapped on PA13 (SWDIO) and PA14 (SWCLK). This is the default state after reset.

Nothing specific is required in the application code to make SWD work.

Special attention must be paid to make sure that, voluntarily or accidentally, the SWD pins are not switched to some alternate functions or affected by I/O settings modifications.

Hint: For instance, STM32Cube PWR examples switch all GPIO (including SWD) in an analog state in order to minimize consumption. This disconnects the debugger. A Connect Under Reset using NRST is required to take back the control of the board. (Refer to *Section 4.2*).

When using STM32CubeMX at configuration stage, PA13 and PA14 can be in one of three states upon selection of Serial Wire in SYS/Debug configuration list:

- Reset, shown by the pins colored in gray in *Figure 24*
- Reserved but inactive shown by the pins colored in orange in *Figure 25*
- Active shown by the pins colored in green in *Figure 26*

Figure 24. SWD pins PA13 and PA14 in Reset state under STM32CubeMX

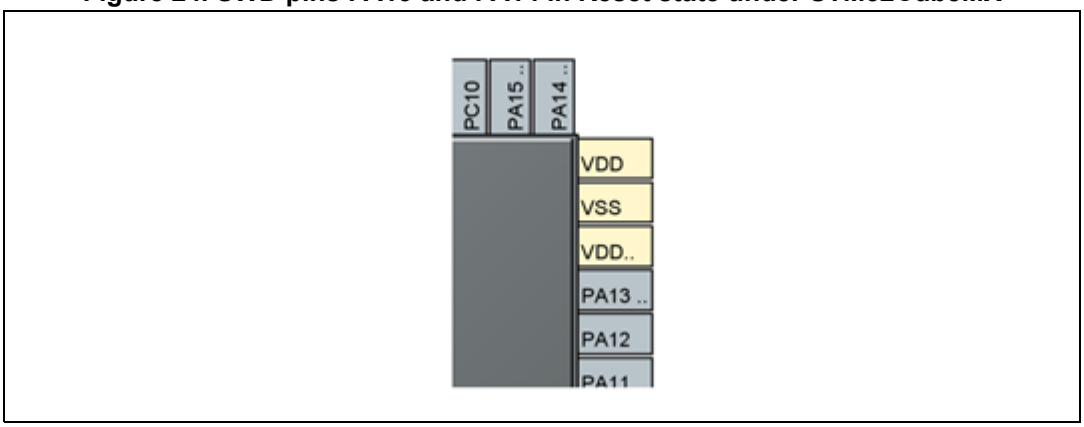


Figure 25. SWD pins PA13 and PA14 in Reserved but inactive state under STM32CubeMX

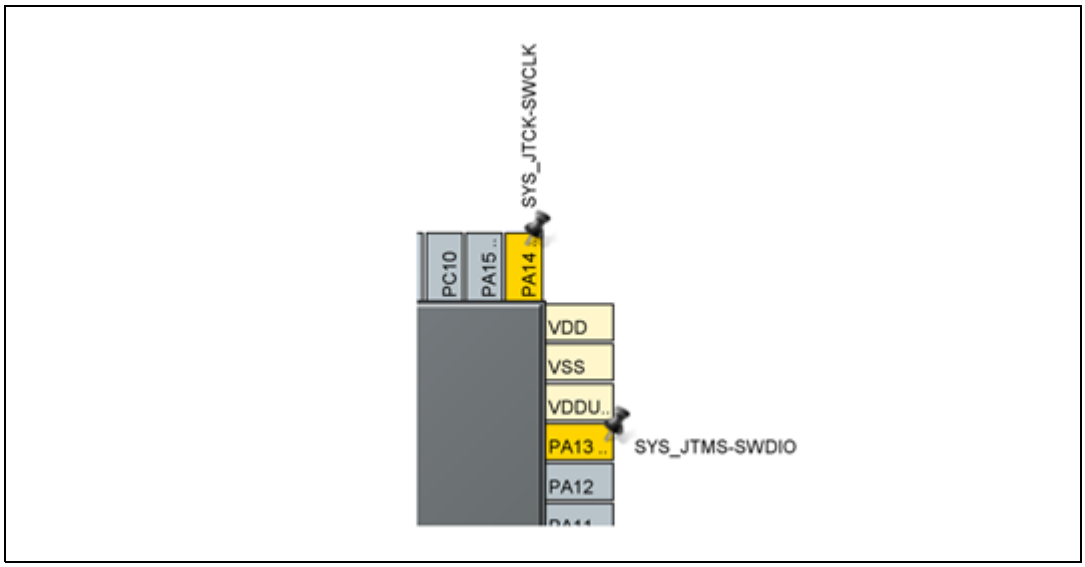
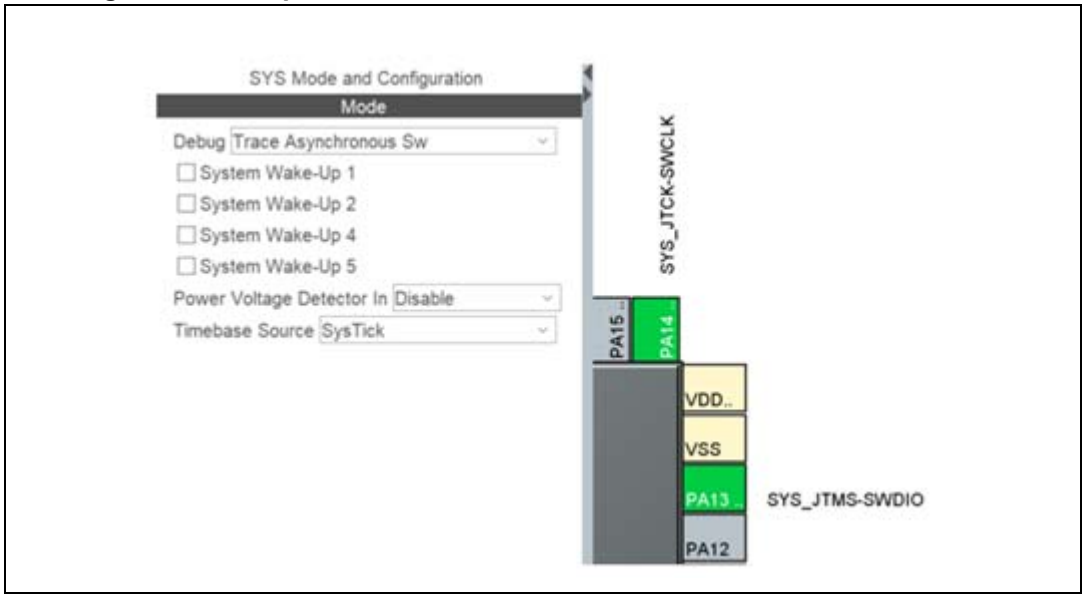


Figure 26. SWD pins PA13 and PA14 in Active State under STM32CubeMX



All three states are functional from SWD connection point of view.

It is anyway recommended to explicitly activate the SWD pins by selecting “Serial Wire” or “Trace Asynchronous SW” (together with SWO. Refer to *Section 7.3 on page 71*). This is the only way by which STM32CubeMX protects the I/O from being selected for another use during the configuration process by highlighting the conflict to the user.

JTAG is not available on Nucleo and Discovery boards.

On EVAL boards, it is available through a dedicated 20-pin connector.

Nevertheless, in STM32CubeMX, SWD remains the default and preferred debug port. For this reason, extra JTAG pins are not reserved. It is then strongly advised to explicitly enable the desired JTAG configuration.

Especially since JTAG is using more pins, users should be aware that it is at the expense of using some IPs.

Refer to the product datasheet for a detailed presentation of the default and alternative function mapping for each pin.

4.2 Reset and connection mode

This section reviews the reset and connection mode available while using ST-LINK/V2or STLINK/V3 debug interface.

4.2.1 Presentation

Connection mode and reset mode are 2 different but dependent concepts:

Reset mode can be either:

- Hardware: drive the NRST pin of the MCU. In all STMicroelectronics hardware kits, the debugger can drive this NRST through ST-LINK/V2 or ST-LINK-V3.

Hint: On Nucleo, check that relevant Solder Bridge SB12 is not OFF.

- Software (write to core register)
  - System: Core and all Peripheral SOC IPs are reset
  - Core: Only Arm® Cortex® is reset

Connection mode can be either:

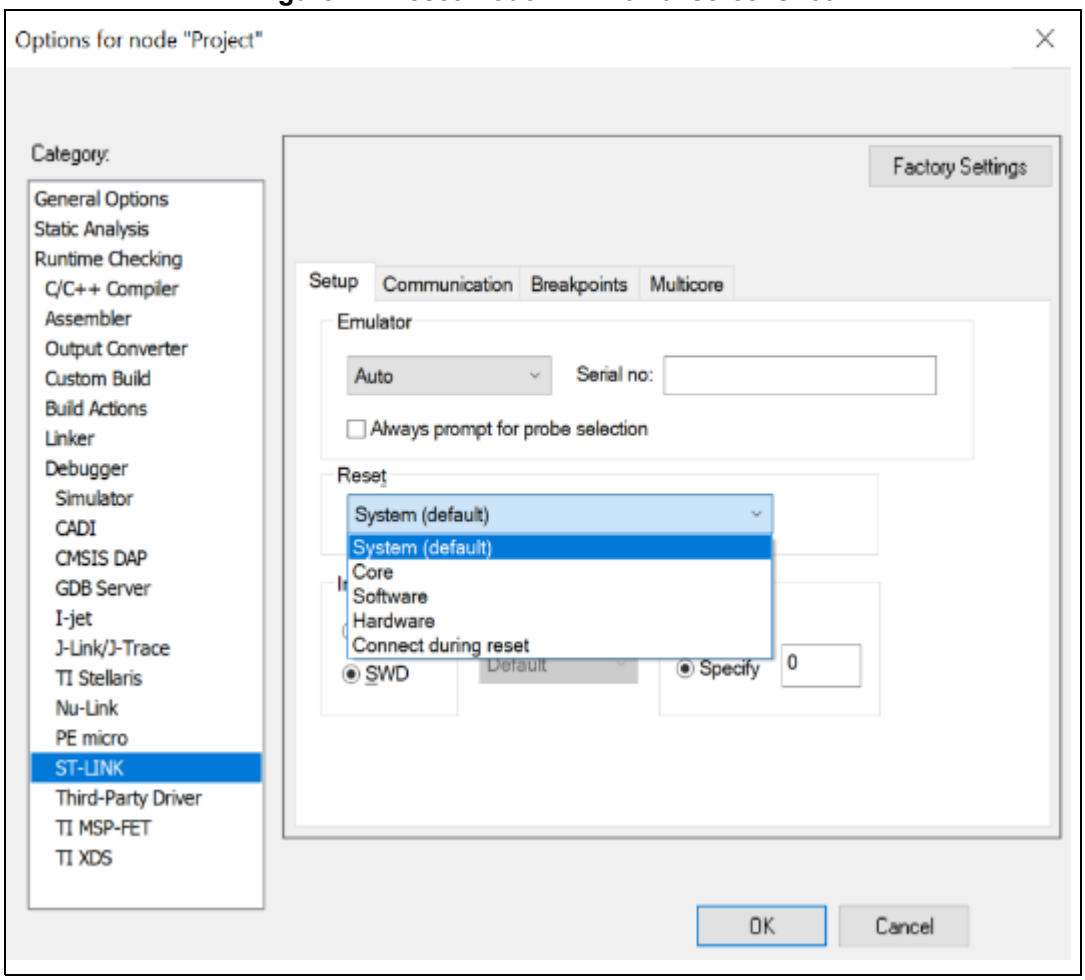
- Normal: Debugger takes control through JTAG/SWD port and starts execution after a software reset.
  This is working only if JTAG/SWD is available:
  - GPIO correctly configured and clocked
  - FCLK or HCLK enabled
  - Main Power domain or Low-Power debug active
- ConnectUnderReset: Debugger takes control while asserted NRST pin, setting GPIO and clock into there default state.
  This is required in case of a reconnection to a system in Low-Power mode or which has changed SWD pin to alternate functions.
- Hotplug: Debugger connect without reset nor halt. Once connected, the user can chose to perform the required action (typically halt to get where the program stands and read registers or memory for instance).

Reset and Connection mode are differently accessible and exposed depending on tool and IDE.

4.2.2 IAR™ EWARM

Reset and Connection mode are seen as a single reset mode option as shown in *Figure 27*.

Figure 27. Reset Mode in IAR8.10: screenshot



- System (default): Normal Connection. Software System Reset prior to jump at main.
- Core: Normal Connection. Software Core Reset prior to jump at main.
- Software: Normal Connection. No Reset prior to jump and stop at main.
- Hardware: Normal Connection. Assert NRST MCU pin prior to jump to main.
- Connect during reset: Connection while asserted Hardware NRST.

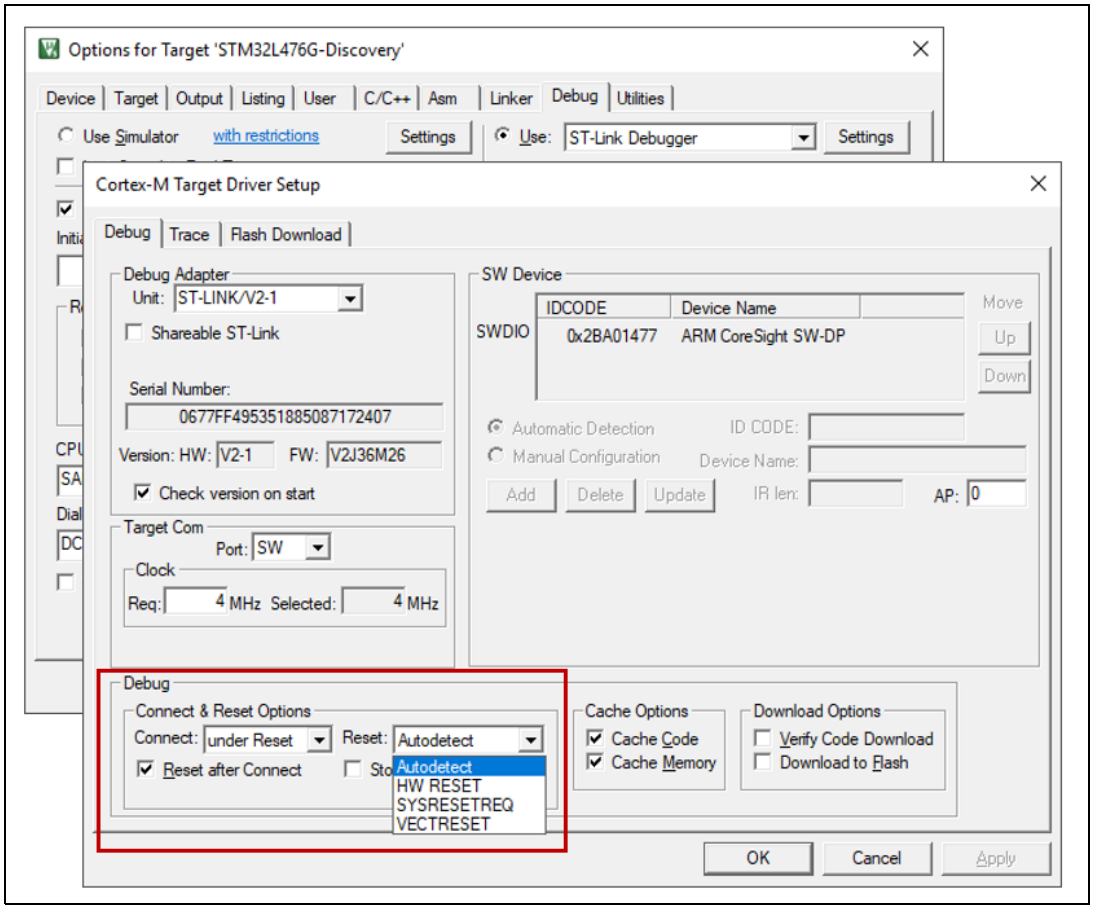
Hotplug connection is accessible with “Attach to running Target” function in project menu.

4.2.3 Keil® MDK-Arm µVISION

Can be set through

Project -> Options -> Debug -> Settings -> Debug

Figure 28. Connect and Reset option Keil®



Connect: controls the operations that are executed when the µVision debugger connects to the target device. The drop-down has the following options:

- Normal just stops the CPU at the currently executed instruction after connecting.
- with Pre-reset applies a hardware reset (HW RESET) before connecting to the device.
- under Reset holds the hardware reset (HW RESET) signal active while connecting to the device. Use this option when the user program disables the JTAG/SW interface by mistake.

Reset after Connect: performs (if enabled) a reset operation as defined in the Reset drop-down list (see below) after connecting to the target. When disabled, the debugger just stops the CPU at the currently executed instruction after connecting the target.

Reset: controls the reset operations performed by the target device. The available options vary with the selected device.

- Autodetect selects the best suitable reset method for the target device. This can be a specialized reset or standard method. If Autodetect finds an unknown device, it uses the SYSRESETREQ method.
- HW RESET performs a hardware reset by asserting the hardware reset (HW RESET) signal.
- SYSRESETREQ performs a software reset by setting the SYSRESETREQ bit. The Cortex®-M core and on-chip peripherals are reset.
- VECTRESET performs a software reset by setting the VECTRESET bit. Only the Cortex®-M core is reset. On-chip peripherals are not reset. For some Cortex®-M devices, VECTRESET is the only way they may be reset. However, VECTRESET is not supported on Cortex®-M0, Cortex®-M0+, Cortex®-M1, and Arm®v8-M cores.

Refer to http://www.keil.com/

Hotplug

If all of the following options are disabled, no hardware reset is performed at debugger start:

Options for **Target -> Debug -> Load Application at startup**

Options for **Target -> Debug -> Settings -> Reset after connect** (with Options for **Target -> Debug -> Settings -> Connect** selected as NORMAL)

Options for **Target -> Utilities -> Update Target before Debugging**

Figure 29. Keil® hotplug step1

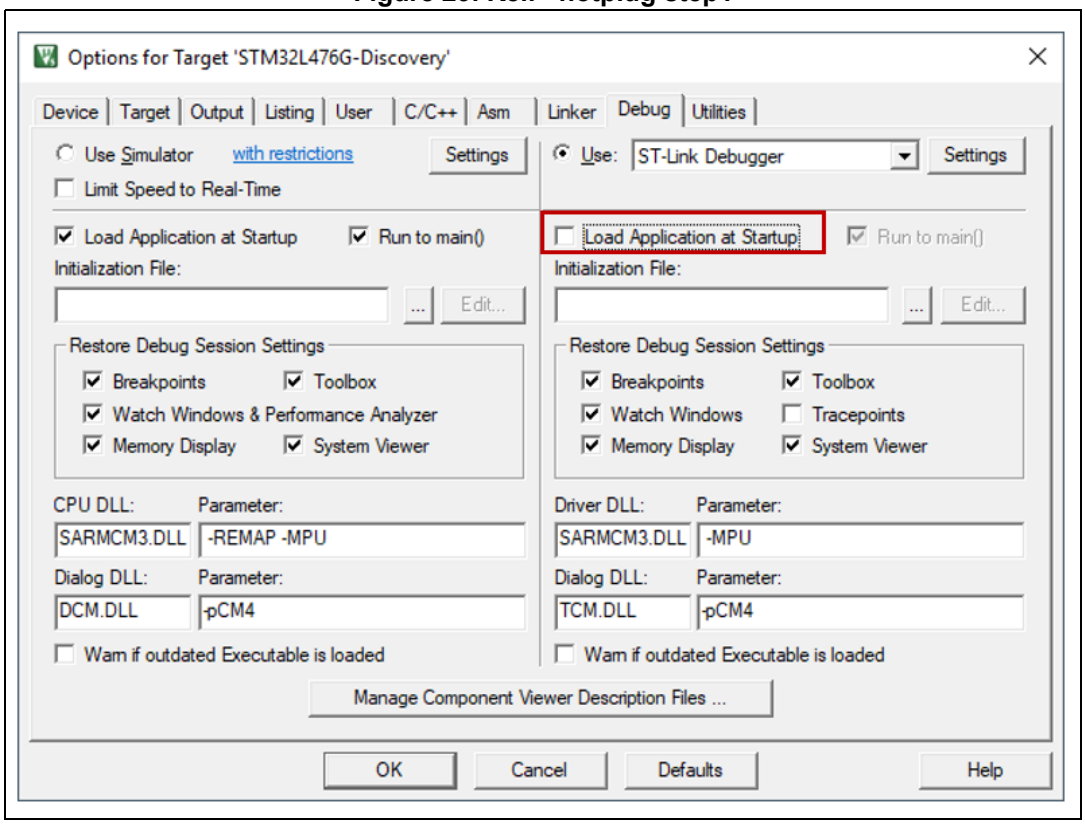


Figure 30. Keil® hotplug step2

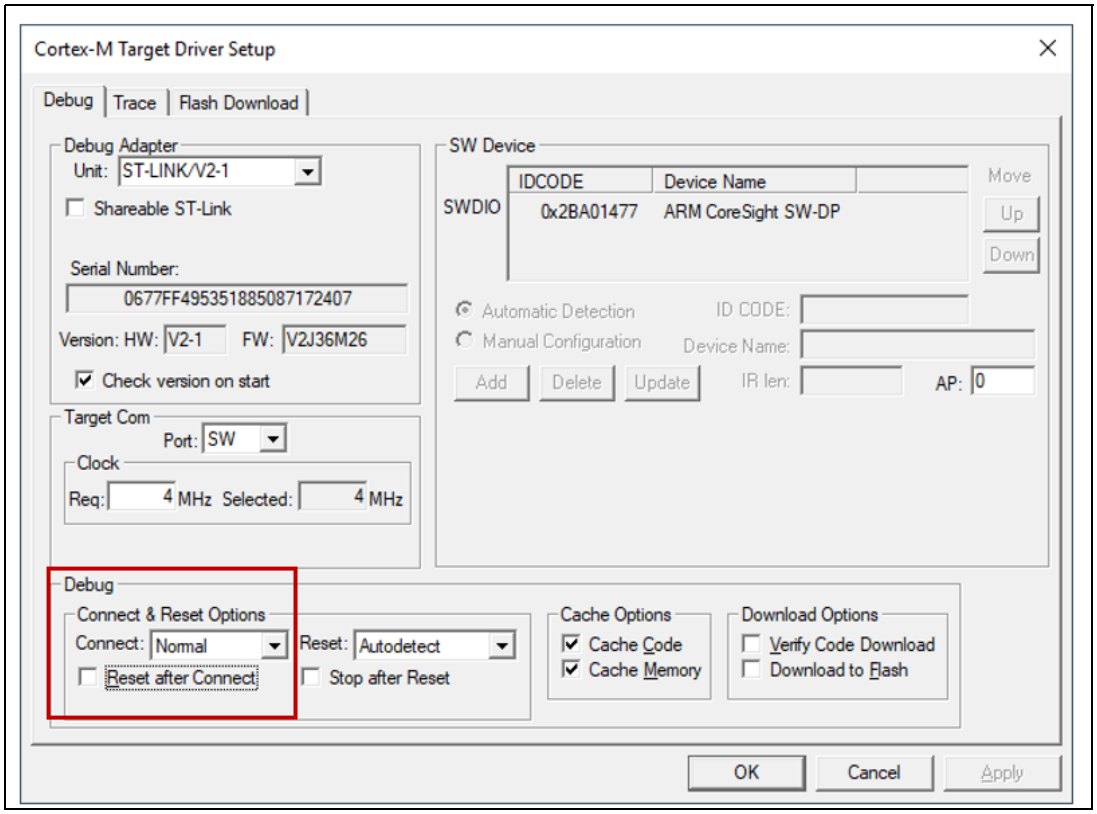
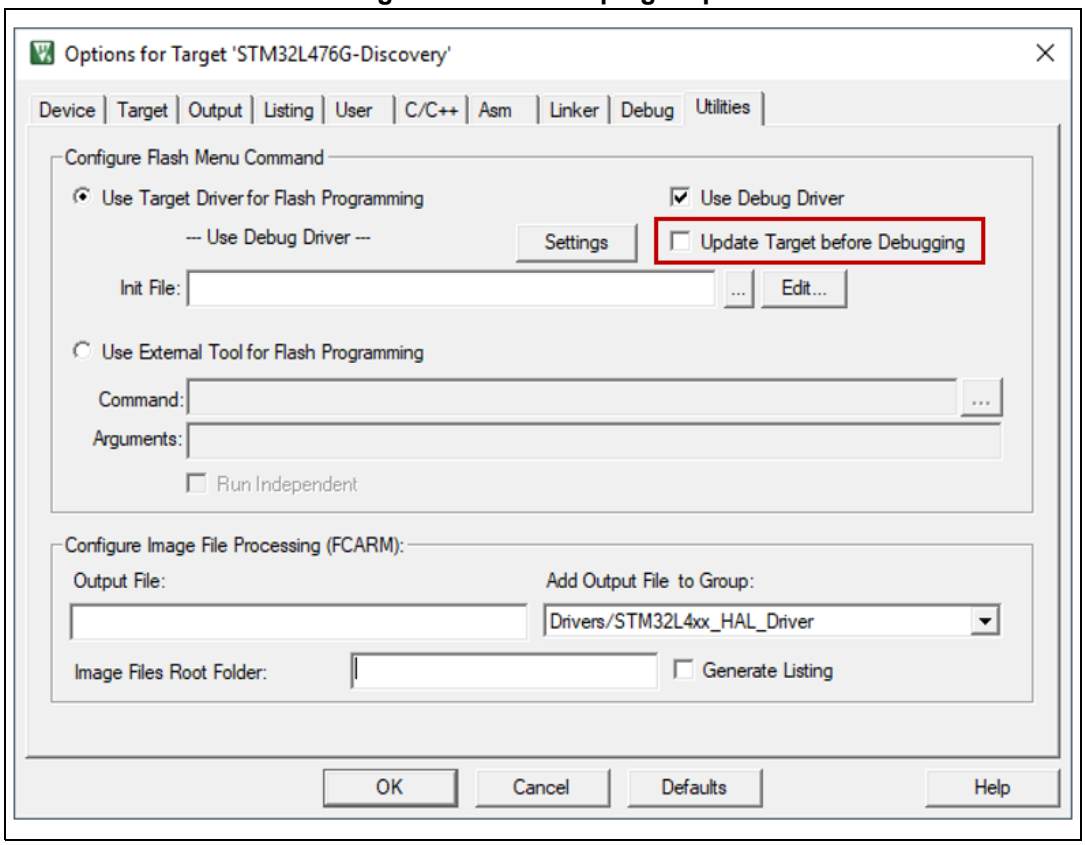


Figure 31. Keil® hotplug step3



With these options disabled, the debugger starts, and the target hardware stops at the current location of the program counter. This allows to analyze the memory and register content.

Because Options For Target - Debug - Load Application at startup is disabled, the debugger does not have any application program and debug information. To load this information into the debugger, use the LOAD debugger command with the option NORESET or INCREMENTAL.

LOAD can be automated using an Initialization File under Options For Target - Debug.

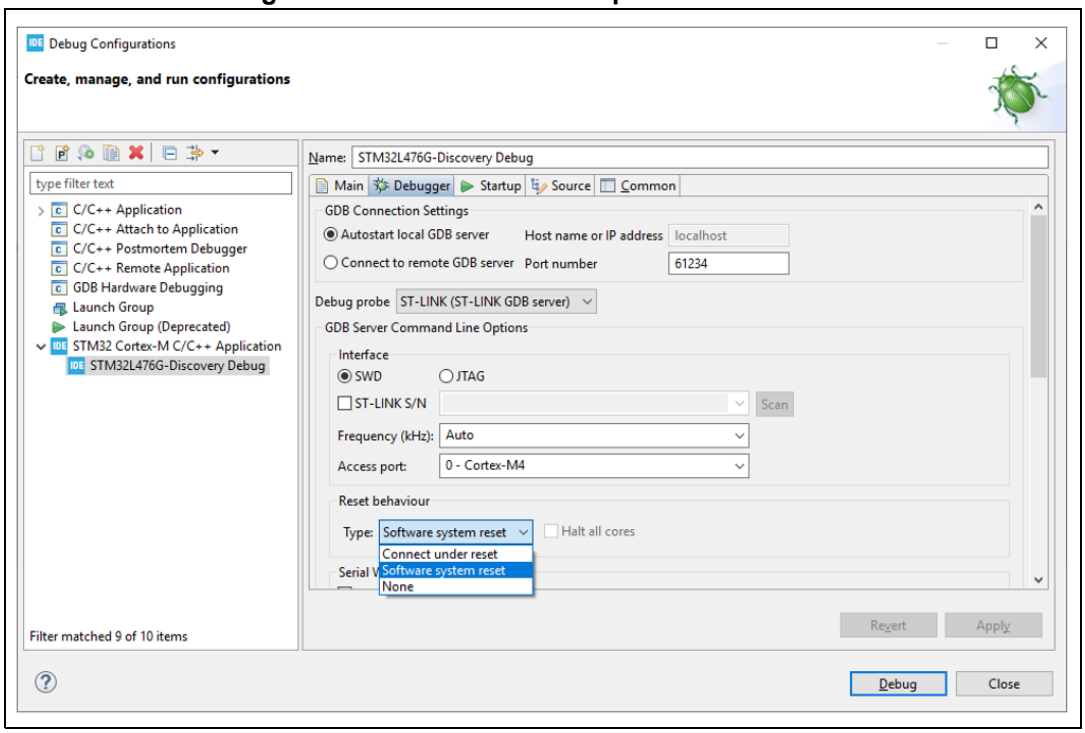
To go further, refer to http://www.keil.com/.

4.2.4 STM32CubeIDE

Reset and connection modes can be changed through

Run -> Debug Configurations -> Debugger

Figure 32. Select Generator Options Reset Mode



The Mode Setup group allows to set up the Reset Mode along with other debug behaviors.

- Reset Mode as Connect under reset: asserts hardware reset and then connects to the target (under reset).
- Reset Mode as None: performs a hardware reset and then connects to the target.
- Reset Mode as Software system reset: does not perform any hardware reset but connects to the target and performs a software system reset.

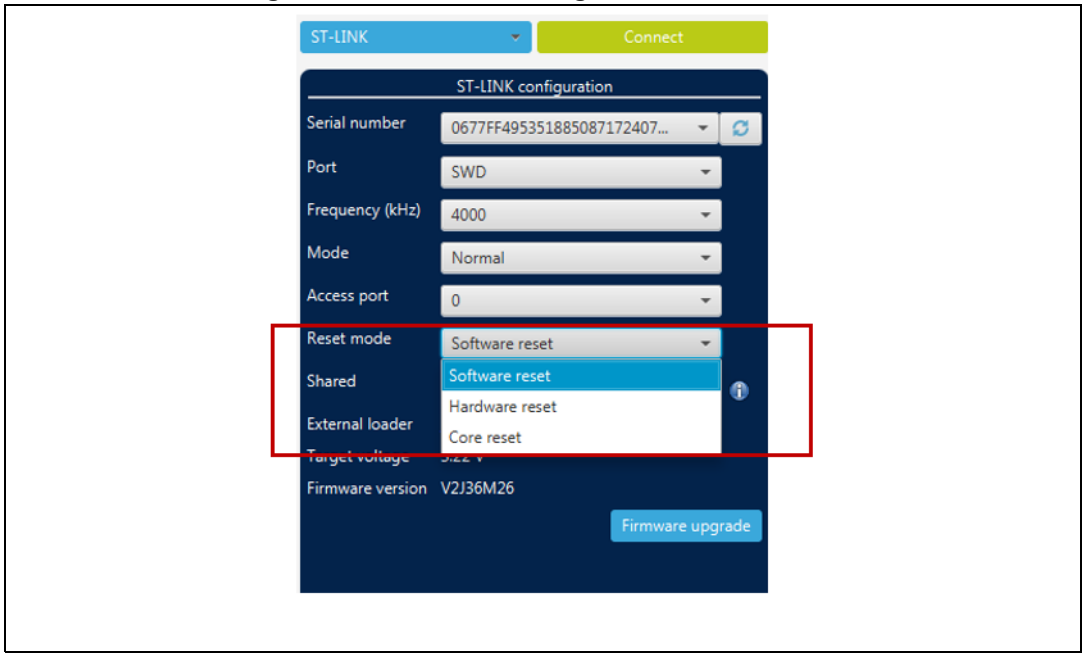
In case of problem to connect to the board with STM32CubeIDE, make sure that NRST from ST-LINK is properly connected to STM32 NRST.

Hotplug mode is not proposed by STM32CubeIDE. STM32CubeProgrammer can be used instead.

4.2.5 STM32CubeProgrammer

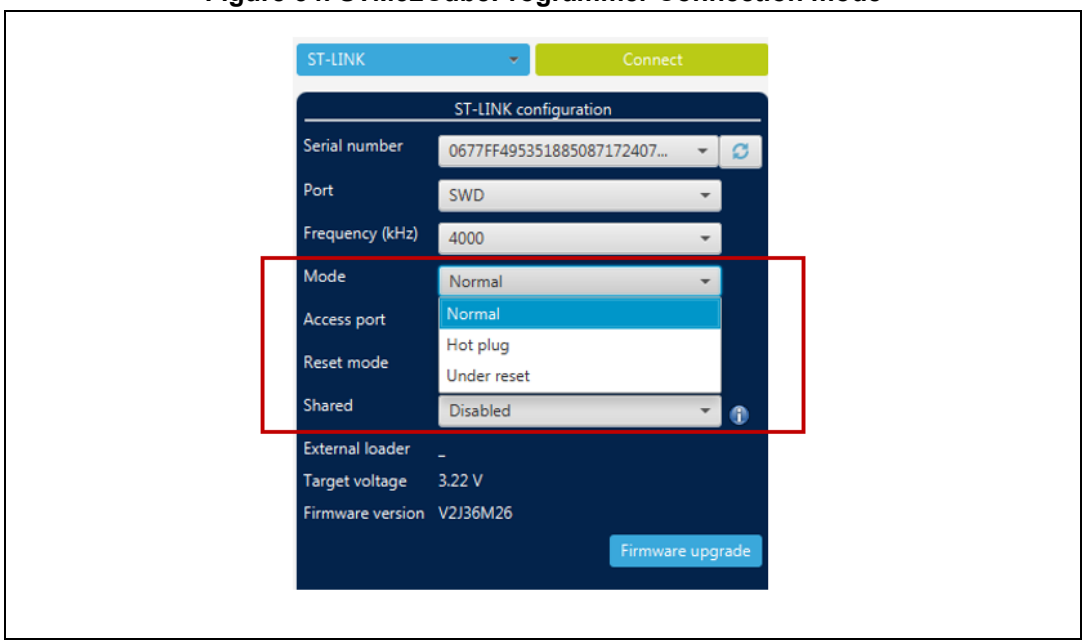
Reset and Connection modes can be selected in the ST-LINK configuration Pane.

Figure 33. STM32CubeProgrammer Reset mode



- Software system reset: Resets all STM32 components except the Debug via the Cortex-M application interrupt and reset control register (AIRCR).
- Hardware reset: Resets the STM32 device via the nRST pin. The RESET pin of the JTAG connector (pin 15) must be connected to the device reset pin.
- Core reset: Resets only the core Cortex-M via the AIRCR

Figure 34. STM32CubeProgrammer Connection mode



- With ‘Normal’ connection mode, the target is reset then halted. The type of reset is selected using the ‘Reset Mode’ option.
- The ‘Connect Under Reset’ mode enables connection to the target using a reset vector catch before executing any instructions. This is useful in many cases, for example when the target contains a code that disables the JTAG/SWD pins.
- The ‘Hot Plug’ mode enables connection to the target without a halt or reset. This is useful for updating the RAM addresses or the IP registers while the application is running.

In Keil® MDK-Arm µVISION, IAR™ EWARM and STM32CubeProgrammer, in case NRST is not connected on the board or PCB a silent fallback operates with a System Reset. In case of failure to take control of a board despite the use of Connection UnderReset / Hardware, check the NRST connection on the board.

4.3 Low-power case

By default, the debug connection is lost if the application puts the MCU in Sleep, Stop, or Standby mode while the debug features are used. This is due to the fact that the Cortex®-M core is not clocked in any of these modes.

However, the setting of dedicated configuration bits in the DBGMCU_CR register allows software debug even when the low-power modes are used extensively.

Refer to the PWR and DBG sections of the reference manual for details.

Appendix A: Managing DBGMCU registers on page 95 guides the user through the various means to manage DBGMCU depending on IDE and needs.

Caution: In order to reduce power consumption, some applications turn all GPIOs to analog input mode, including SWD GPIOs. This is the case for all PWR examples provided in STM32Cube (debug connection is lost after `SystemPower_Config()` which sets all GPIOs in Analog Input State).
Enabling low-power debug degrades power consumption performance by keeping some clocks enabled and by preventing to optimize GPIO state. Even if this is useful for functional debugging, it has anyhow to be banned as soon as the target is to measure/enhance power consumption.
All DBGMCU registers values are kept while reset. Users must pay attention not to let debug or unwanted states when returning to normal execution (refer to *Section 9: Dual-Core microcontroller debugging on page 92*).

5 Breaking and stepping into code

This chapter provides users with highlights about a few points affecting system behavior at code break.

5.1 Debug support for timers, RTC, watchdog, BxCAN and I^2C

During a breakpoint, it is necessary to choose how the counter of timers, RTC and watchdog should behave:

- They can continue to count inside a breakpoint. This is usually required when a PWM is controlling a motor, for example.
- They can stop counting inside a breakpoint. This is required for watchdog purposes.

For the BxCAN, the user can choose to block the update of the receive register during a breakpoint.

For the I^2C, the user can choose to block the SMBUS timeout during a breakpoint.

Those options are accessible in DBGMCU freeze registers (DBGMCU_APB1FZR1, DBGMCU_APB1FZR2) which can be written by the debugger under system reset.

If the debugger host does not support these features, it is still possible to write these registers by software.

Refer to *Appendix A: Managing DBGMCU registers on page 95* to find suitable ways to handle debug options depending on IDEs and needs.

5.2 Debug performance

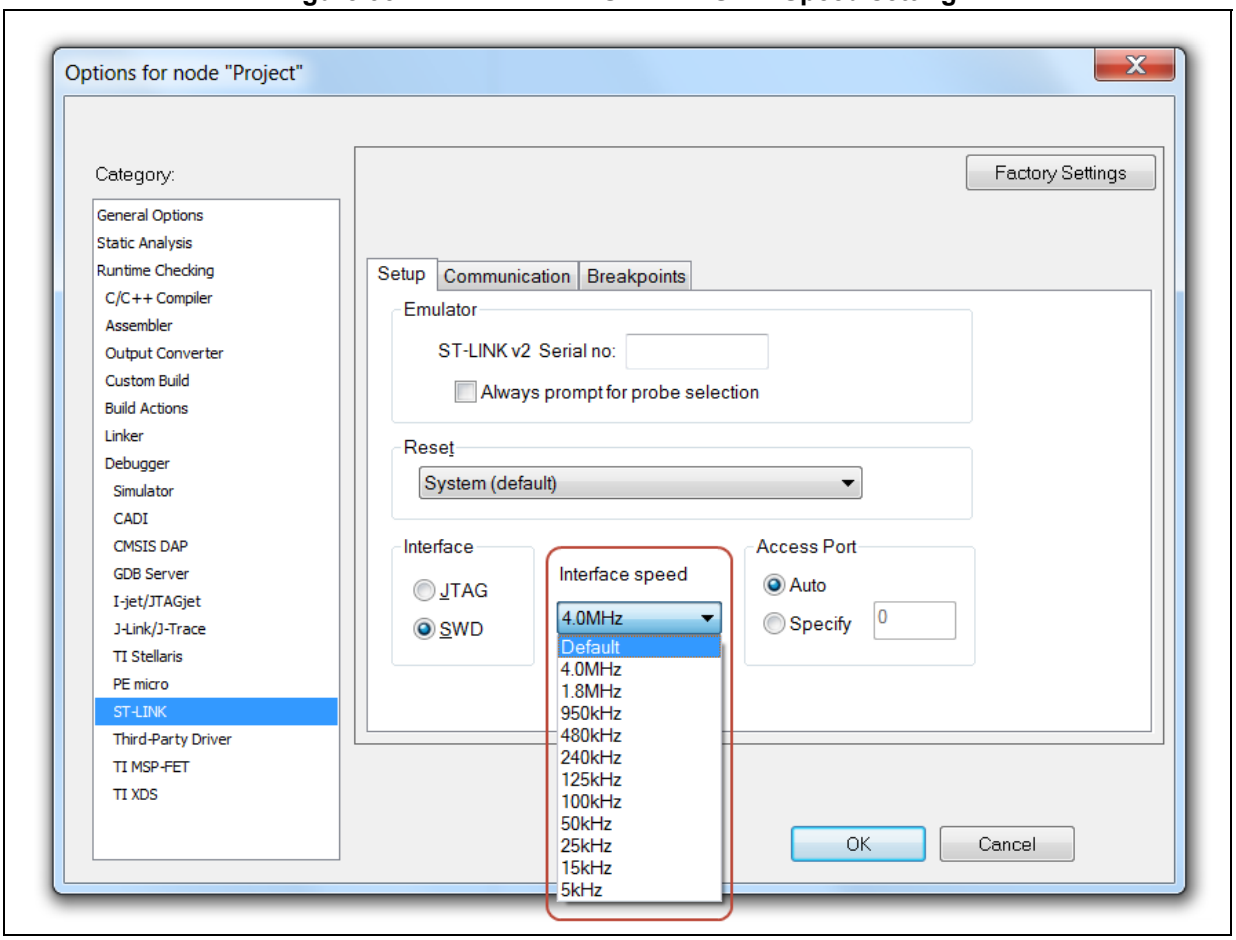
To save flashing time and improve debugger reactivity when stepping, make sure that the higher SWD frequency possible is used with the probe.

When using IAR™ EWARM, or Keil® MDK-Arm µVISION speed is set at speed is set at 1.8 MHz by default. On system with a core clock greater than 1 MHz, it is safe to use the highest 4 MHz SWD speed.

5.2.1 IAR™ EWARM

In **Project -> Option -> ST-LINK -> Interface speed**

Figure 35. IAR™ EWARM ST-LINK SWD Speed setting

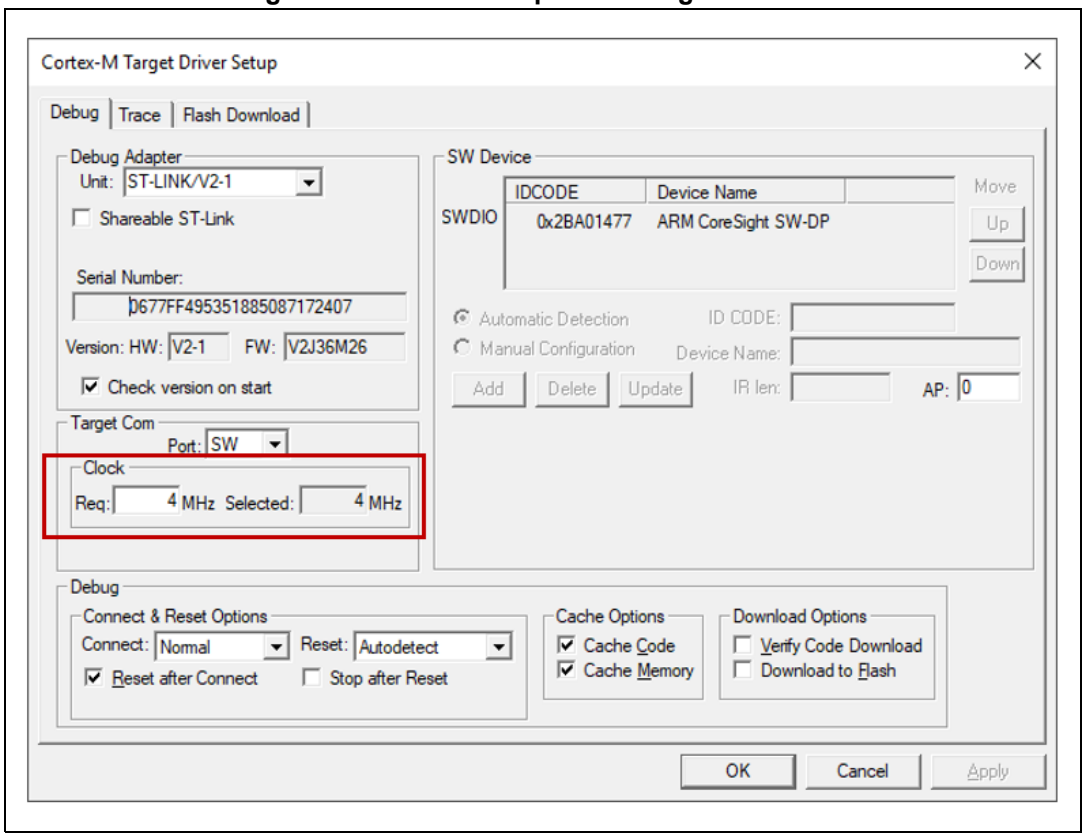


5.2.2 Keil® MDK-Arm µVISION

SWD Speed setting is accessible in

Project -> Options for Target.. -> Debug -> Settings -> Target Com

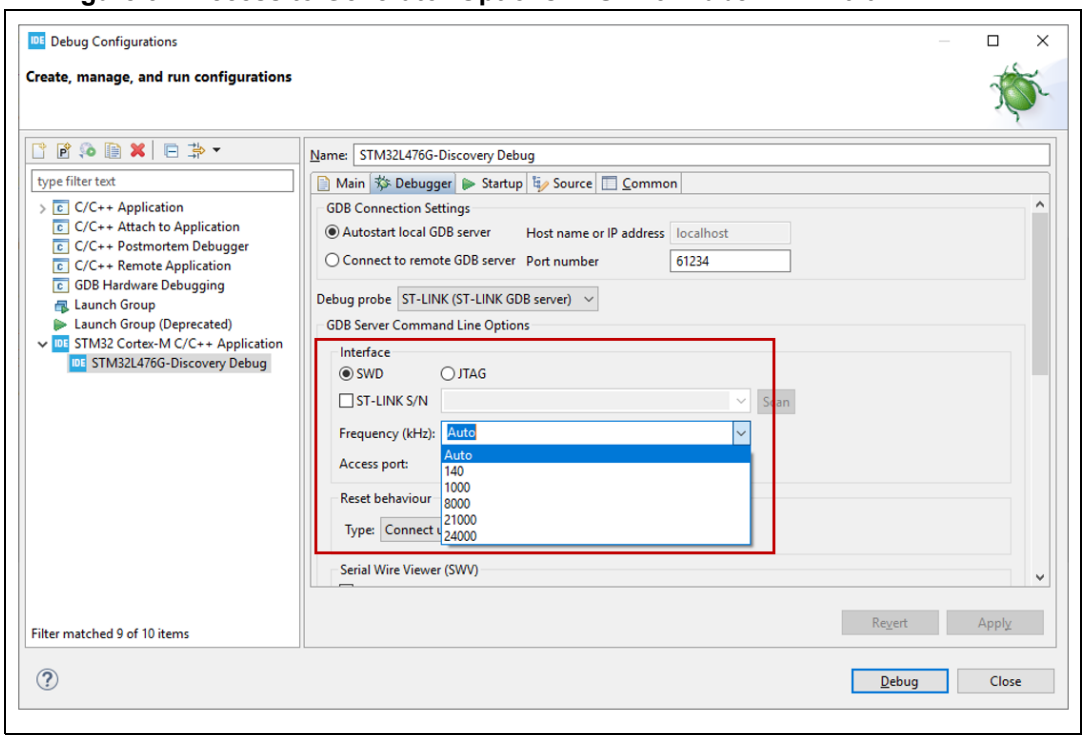
Figure 36. Keil® SWD Speed Setting



5.2.3 STM32CubeIDE

In **Run -> Debug Configuration -> Debugger Pane**

Figure 37. Access to Generator Options in STM32CubeIDE V2.0.0



Note: SWD communication is always possible on all ST boards whereas JTAG is only present on EVAL boards.

SWD communication is always present on all Cortex®-M devices whereas JTAG is not present on Cortex®-M0(+) devices. Refer to Appendix D on page 116 for a complete overview of debug capabilities for each Cortex®-M type.

5.3 Secure platform limitation

The STMicroelectronics platform provides the following code protection means.

RDP: ReadOut Protection

Prevents Flash Memory access through the JTAG for ALL Flash memory.

PcROP: Proprietary Code ReadOut Protection

Prevents read access of configurable Flash memory areas performed by the CPU execution of malicious third-party code (Trojan Horse).

WRP: Prevents accidental or malicious write/erase operations.

For further details please refer to the reference manual or section *Training L4* on STMicroelectronics website *www.st.com*.

The next sections provide additional details on the expected behavior of the secure applications.

5.3.1 RDP

- Level 0: No Protection.

  This is the factory default mode allowing all accesses.
- Level 1: Read Protection.

  Any access to Flash or protection extension region generates a system hard-fault which blocks all code execution until the next power-on reset. A simple reset does re-enable code execution; power must be switched off and on so that power-on reset enables code execution. The restriction depends on the STM32 Series as described in *Table 4*.

Table 4. STM32 Series RDP protection extension

Product	RDP protection extension
F0	+ backup registers
F2	+ backup SRAM
F3	+ backup registers
F4	+ backup SRAM
L0	+ EEPROM
L1	+ EEPROM
L4	+ backup registers + SRAM2
L5	RDP 4 levels + backup registers + SRAM2
F7	+ backup SRAM
H7	+ backup SRAM

Thus, any attempt to load, or connect to, an application running from Flash crashes.

It is still possible to load, execute and debug an application in SRAM.

Option Bytes management can be done with ST-LINK utility or with an application running from SRAM.

Going back to RDP Level 0 completely erases the Flash.

- Level 2: No Debug.

  JTAG/SWD connection is killed. There is no way back. In this case, nobody - even STMicroelectronics - can perform any analysis of defective parts.
- Level 0.5 is an additional protection level associated with TrustZone(only available in STM32L5 Serie). RDP 0.5 is available only when TrustZone is enabled. Debug of secure domain is forbidden, only non-secure domain can be accessed for debug. Regression from level 0.5 to level 0 triggers a Flash mass erase, as well as backup registers and all SRAMs.Regression from RDP Level 1 to RDP Level 0.5 leads to a partial Flash memory erase: only the non-secure part is erased

Note: Refer to AN5421 and AN5347 for more informations about Trustzone development on STM32L5 Series.

5.3.2 PCROP

Proprietary Code ReadOut Protection is the ability to define secure area in Flash where user can locate a proprietary code.

This prevents malicious software or debugger from reading sensitive code.

In case an application with third party code in PCROP area needs to be debugged, the following points must be considered:

- Step-into PCROP function is tolerated but ignored (Step-over)
- Access to protected memory through debugger trigs Flash Interruption (Instrument NMIHandler) and return default pattern for the whole area

For further details refer to section *Memory Protection* in the reference manual of the device.

6 Exception handling

It is usually helpful, or even mandatory in complex project, to properly trap and find root cause of software exception like HardFault and NMI. This chapter intends to make the user aware of a few techniques used to help investigating such issue.

In order to get deeper into the subject, the user can usefully refer to Joseh Yiu's work and book collection *The Definitive Guide to Arm-Cortex-M*, and to Carmelo Noviello's recent on-line guide *Mastering STM32*.

6.1 Default weak Handlers

By default Handlers are implemented as __weak functions which perform endless loops:

```
__vector_table
        DCD     sfe(CSTACK)
        DCD     Reset_Handler             ; Reset Handler
        DCD     NMI_Handler               ; NMI Handler
        DCD     HardFault_Handler         ; Hard Fault Handler
        DCD     0                         ; Reserved
        DCD     0                         ; Reserved
        DCD     0                         ; Reserved
        DCD     0                         ; Reserved
        DCD     0                         ; Reserved
        DCD     0                         ; Reserved
        DCD     0                         ; Reserved
        DCD     SVC_Handler               ; SVCall Handler
        DCD     0                         ; Reserved
        DCD     0                         ; Reserved
        DCD     PendSV_Handler            ; PendSV Handler
        DCD     SysTick_Handler           ; SysTick Handler
```

Nothing is trigged on debugger side and application looks hanged / stuck.

In that case, code break is needed and the PC must be at the address of the Handler.

Some IDEs provide the faulty calling code through Call stack window. (Keil® MDK-Arm µVision, STM32CubeIDE).

If it is not the case, display registers and find the faulty code address in SP + 0x18

In STM32CubeIDE all weak default handlers point to the same DefaultHandler which can be confusing.

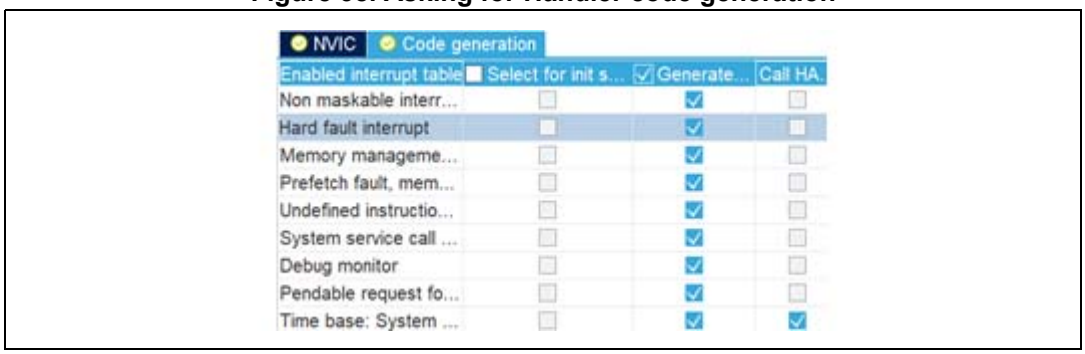
A more efficient approach is to trap the exception by instrumenting Handlers.

6.2 Custom Handlers

One way to generate templates of Handler functions is to use STM32CubeMX.

In ***Configuration -> NVIC Configuration -> Code Generation***, use Generate IRQ handler tick boxes as shown in *Figure 38*.

Figure 38. Asking for Handler code generation



When Non maskable interrupt and Hard fault interrupt are selected, the following code is generated:

```
void NMI_Handler(void)
{
  /* USER CODE BEGIN NonMaskableInt_IRQn 0 */

  /* USER CODE END NonMaskableInt_IRQn 0 */
  /* USER CODE BEGIN NonMaskableInt_IRQn 1 */

  /* USER CODE END NonMaskableInt_IRQn 1 */
}

/**
* @brief This function handles Hard fault interrupt.
*/
void HardFault_Handler(void)
{
  /* USER CODE BEGIN HardFault_IRQn 0 */

  /* USER CODE END HardFault_IRQn 0 */
  while (1)
  {
  }
  /* USER CODE BEGIN HardFault_IRQn 1 */

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

This simple declaration overriding the default weak function,removes ambiguity and clarifies the call stack.

In order to trap the exception, a hardware or a software breakpoint can be set in the IDE or directly programmed in the source code using Arm® instruction BKPT.

Caution: BKPT is not tolerated if no debugger is connected (refer to *Chapter 9: Dual-Core microcontroller debugging on page 92*). it is advised to set it under `#ifdef` statement.

In-line insertion of assembly instruction in application C code depends on the IDE.

- IAR™ EWARM and STM32CubeIDE

```
void NMI_Handler(void)
{
#ifdef DEBUG
    asm ("BKPT 0");
#endif
}
```

- Keil®

```
void NMI_Handler(void)
{
#ifdef DEBUG
    __asm
    {
    BKPT 0
    }
#endif
}
```

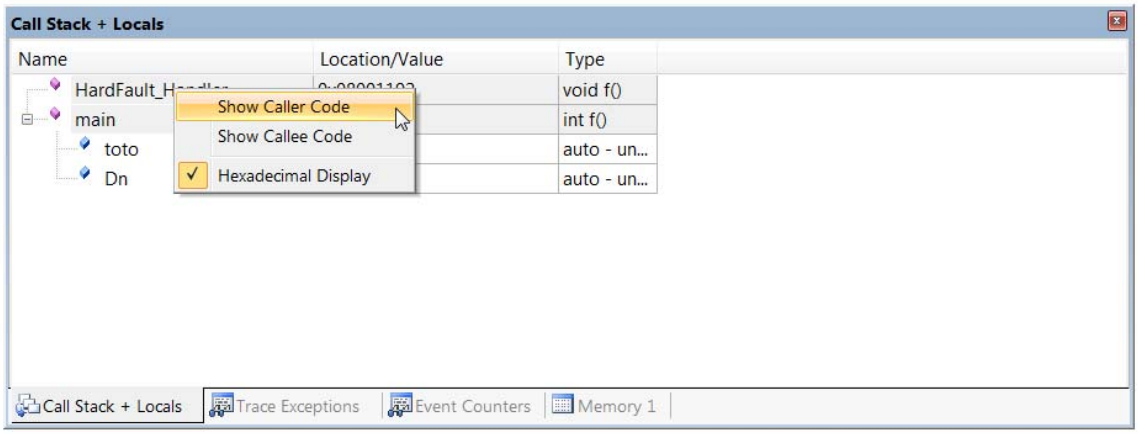
For each IDE, it is also possible to use the abstraction function defined in the CMSIS library and provided in STM32Cube software pack.

```
void NMI_Handler(void)
{
#ifdef DEBUG
__BKPT(0);
#endif
}
```

In all cases, the Halt Debug-Mode is entered; it allows to investigate the issue by inspecting Call Stack and Registers content.

Tip: On Keil® MDK-Arm µVISION, the caller code is not directly accessible in the Call Stack Window. Right clicking "Show Caller Code" as in *Figure 39* leads to the faulty line.

Figure 39. Keil® Access to Show Caller Code in Contextual menu



6.3 Trapping div/0 exception

Most often, code execution causing a division by zero are difficult to investigate:

- Nothing is neither triggered nor trapped.
- Erroneous returned value generates an unexpected and unpredictable behavior that is very difficult to analyze.

This chapter gives several tips in order to properly trap div/0 exceptions.

6.3.1 Cortex®-M0/M0+ case

For targets that do not support hardware division instructions (SDIV/UDIV), integer division-by-zero errors can be trapped and identified by means of the appropriate C library helper functions:

`__aeabi_idiv0()`

When integer division by zero is detected, a branch to `__aeabi_idiv0()` is made. A breakpoint placed on __aeabi_idiv0() allow to trap the division by zero.

To ease the breakpoint application, override the default function:

```
void __aeabi_idiv0()
{
  #ifdef DEBUG
  __BKPT(0);
  #endif
}
```

This way, and depending on IDE, the call stack or registers can be examined and the offending line in the source code can be rapidly found.

To go further refer to section 7.7 of *Arm® Compiler Software Development Guide*.

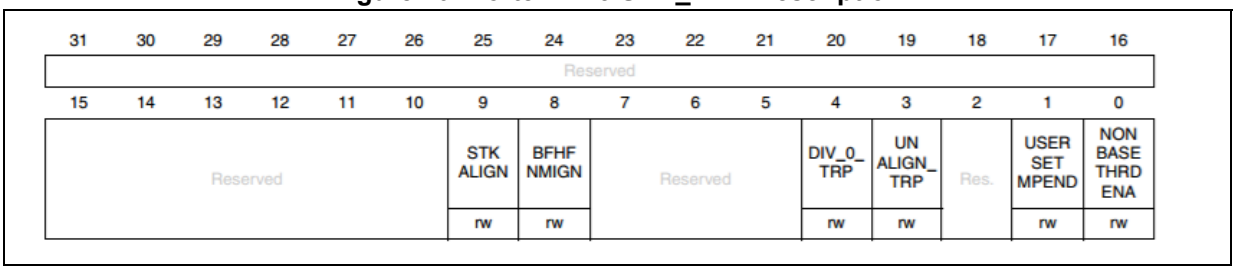
6.3.2 Cortex®-M3/4/7 case

For targets that support hardware division instructions, Trapping of DIV0 operation is possible by configuring System Control Block (SCB) registers, accessible through CMSIS library.

For example on Cortex®-M3:

SCB_CCR register description is provided in *Figure 40*.

Figure 40. Cortex®-M3 SCB_CCR Description



Refer to *STM32F10xxx/20xxx/21xxx/L1xxxx Cortex-M3 programming manual* (PM0056).

Setting bit 5 of SCB_CCR register

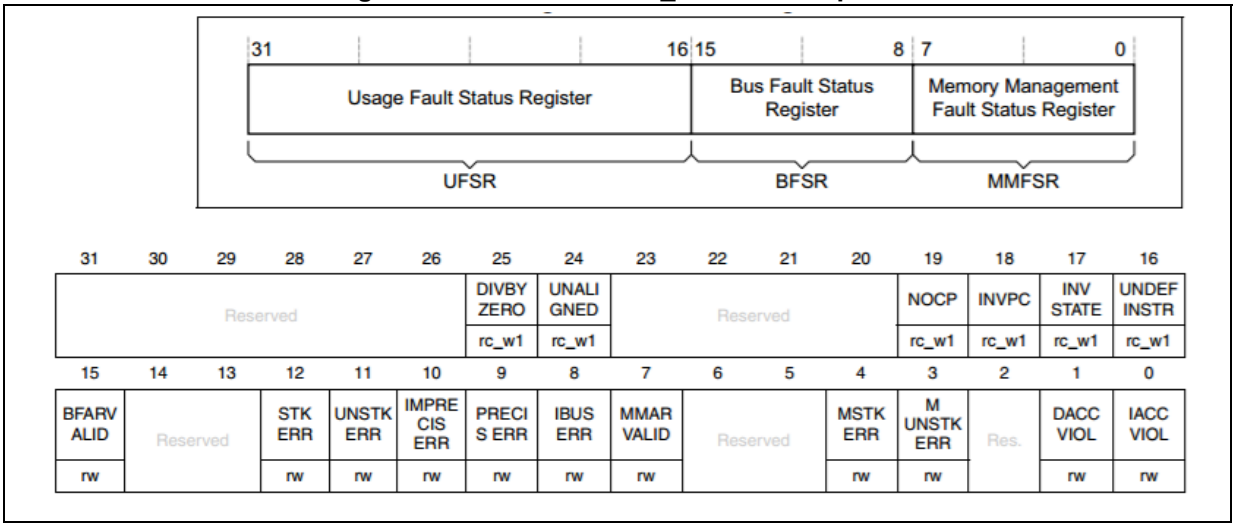
```
SCB->CCR |= 0x10; // enable div-by-0 trap
```

When Div0 occurs it is trapped in HardFault_Handler.

With breakpoint on while instruction into HardFault_Handler, CallStack point to the offended line and SCB->CFSR register explicits the type of fault

SCB_CFSR register description is provided in *Figure 41*.

Figure 41. Cortex-M3 SCB_CFSR Description

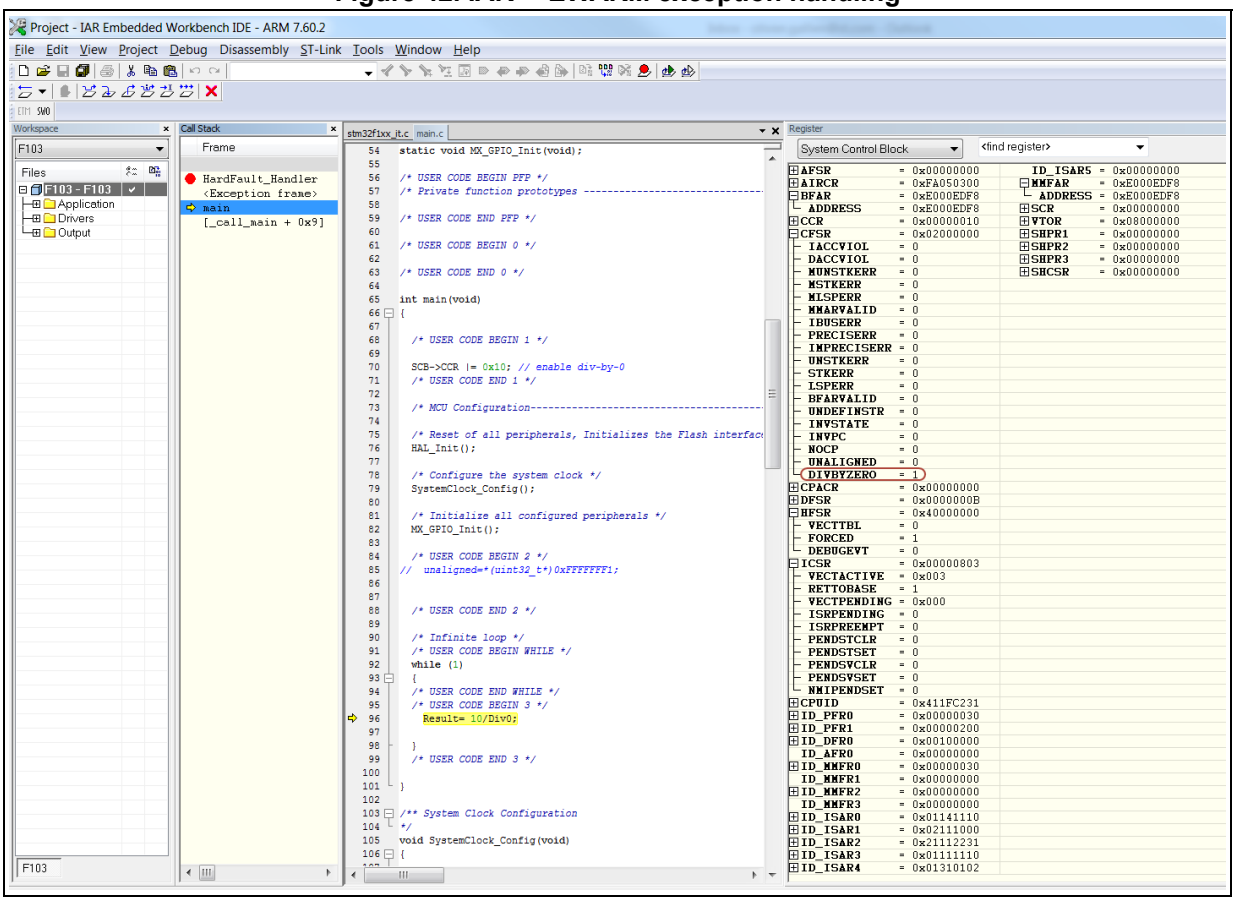


The following sections describe the management of SCB registers as a function of the selected IDE.

IAR™ EWARM

Detailed R/W access to the values of each SCB registers bits at runtime can be obtained through ***View -> Register*** -> ***System Control Block*** (from Pick List) as shown in *Figure 42*.

Figure 42. IAR™ EWARM exception handling

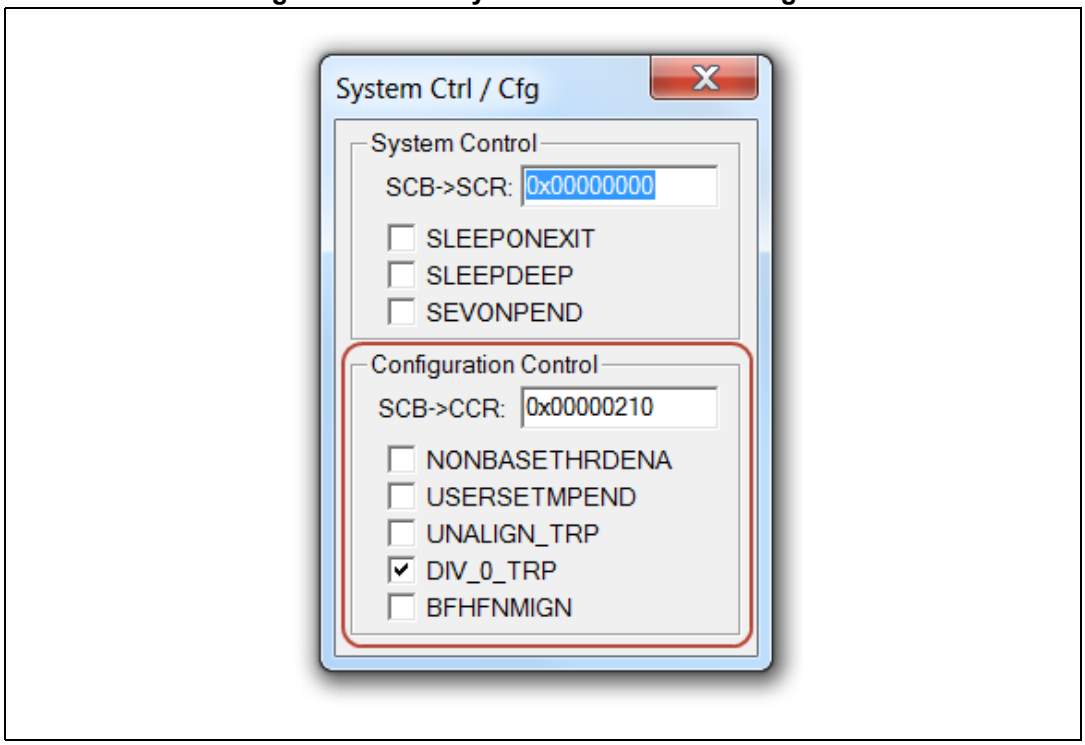


Keil® MDK-Arm µVISION

SCB->CCR can be managed at run time through ***View*** -> ***System Viewer*** -> ***Core Peripheral*** -> ***System Control and Configure***.

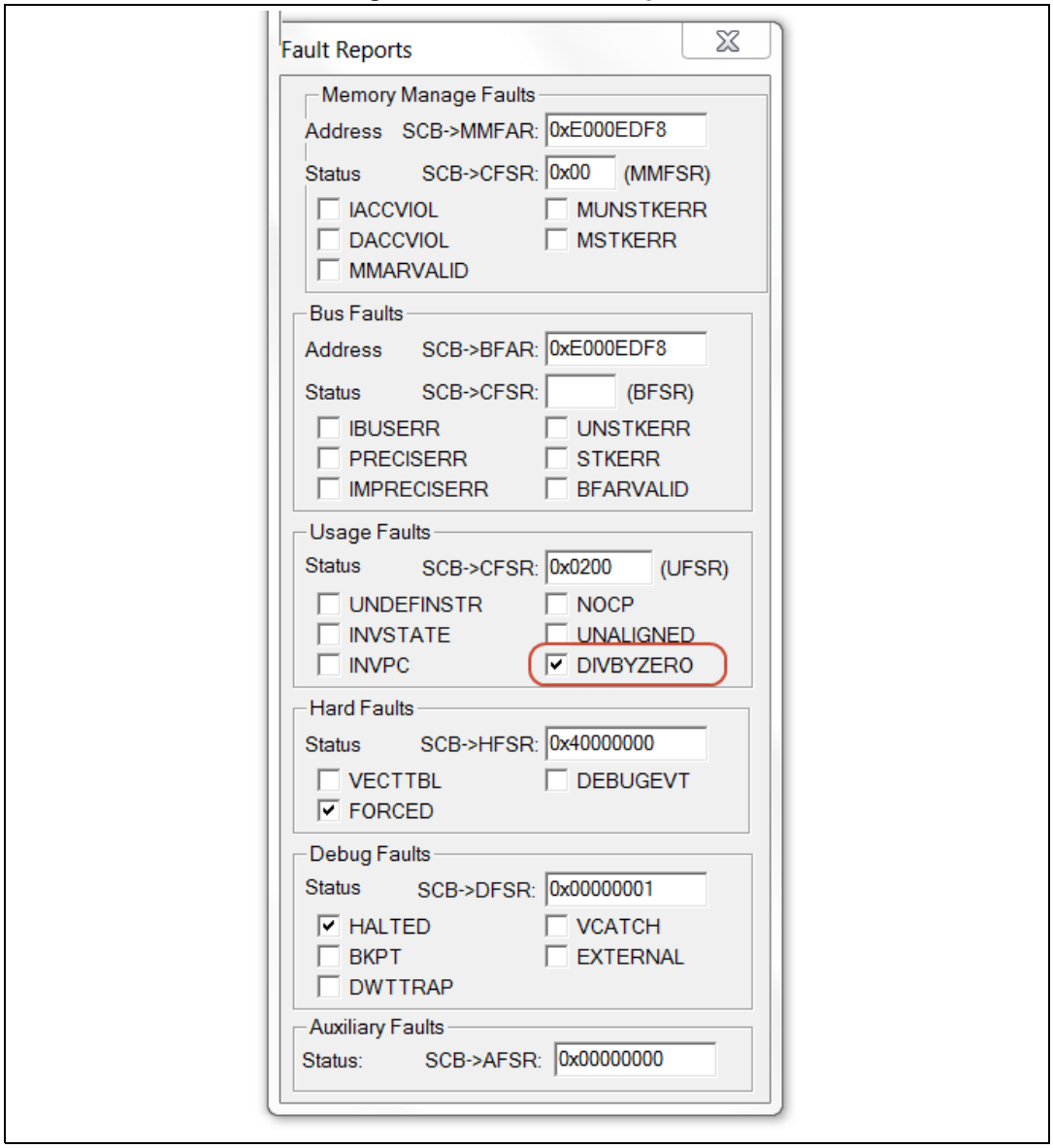
Refer to *Figure 43* for details.

Figure 43. Keil® System Control and Configure



The fault type can be investigated using ***Peripherals -> Core Peripherals*** -> ***Fault Reports*** as shown in *Figure 44*.

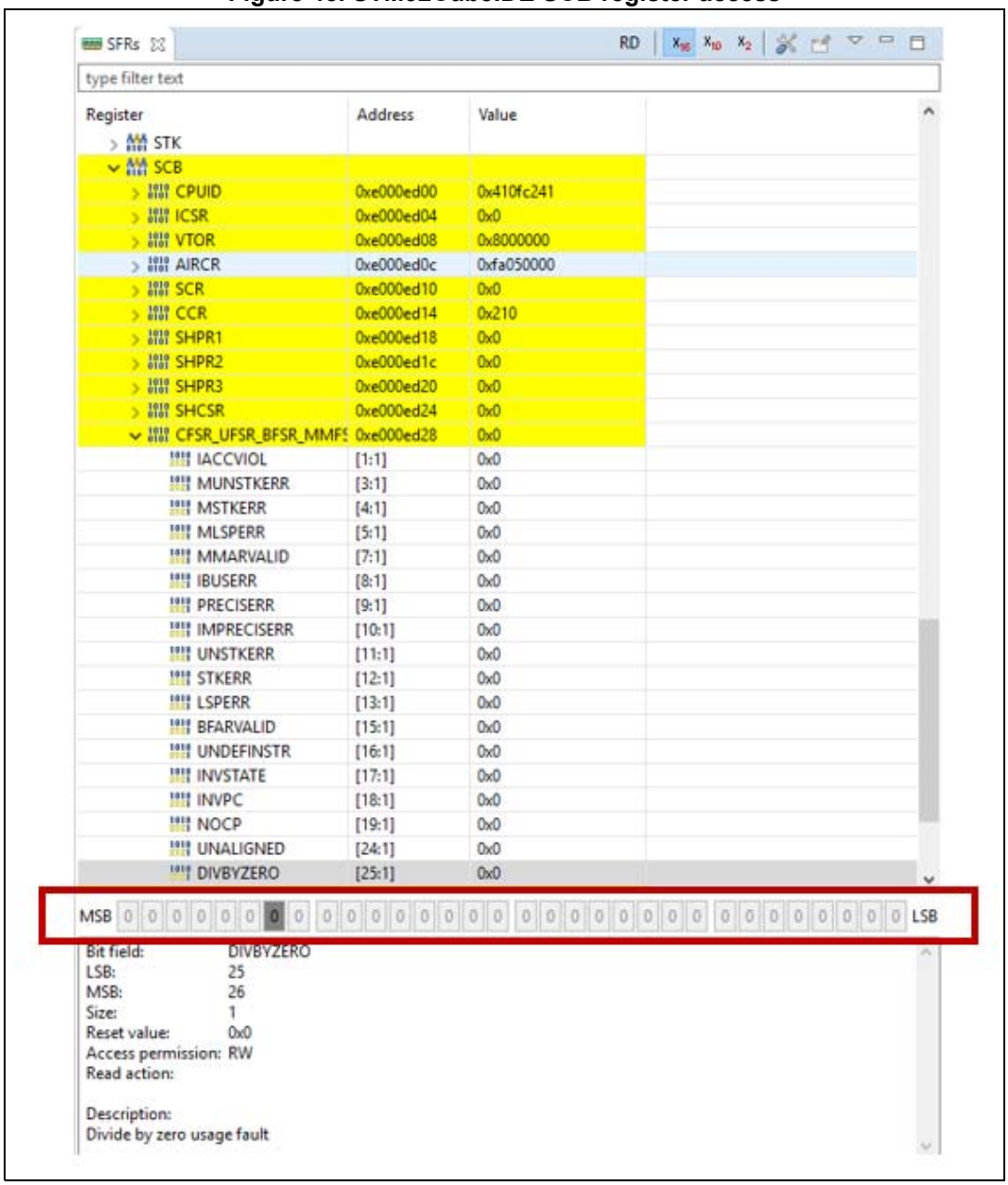
Figure 44. Keil® Fault Reports



STM32CubeIDE

At runtime, while debug is in break state, the SCB register can be accessed in read mode or in write mode through: **Window -> Show View -> SFRs** as shown in *Figure 45*.

Figure 45. STM32CubeIDE SCB register access



If you need more informations about the Hardfault,you can first enable faulting or halting when the processor executes an SDIV or UDIV instruction with a divisor of 0:

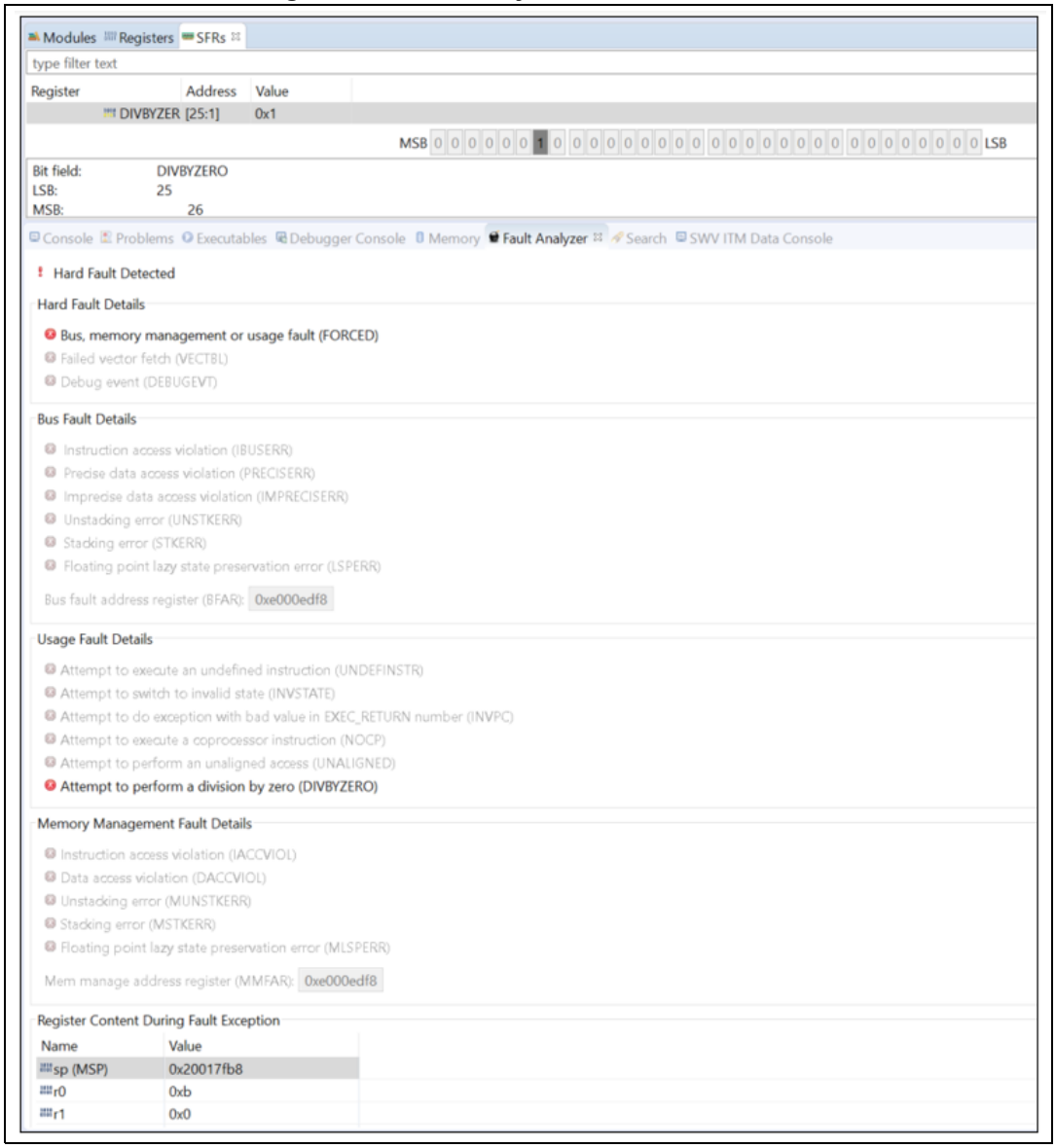
SCB->CCR | = 0x01.

At runtime, before pressing “resume” button, open “Fault Analyzer” in

Window ->Show View -> Fault Analyzer

If the code performs an SDIV or UDIV instruction with a divisor of 0, code stops with informations in “Fault Analyzer” (see *Figure 46*).

Figure 46. Fault Analyzer in STM32CubeIDE



Independently from the IDE, for projects including the CMSIS library, the content of the registers in the code can also be printed:

```
void HardFault_Handler(void)
{
  volatile uint32_t csfr= SCB-> CSFR ;  // load into variable
  printf ( "SCB-> CSFR 0x%08x \n", SCB-> CSFR) // print
  while (1)
  {
  }
}
```

The same content can as well be obtained directly from the memory with any memory browser.

Other faults like UNALIGNED, UNDEFINSTR can be managed in a similar way.

For more details, refer to the relevant programming manual:

- *STM32F4 and STM32L4 Series Cortex®-M4 programming manual*
- *(PM0214)*
- *STM32F7 Series Cortex®-M7 processor programming manual (PM0253)*

Relevant information is also available on partners websites:

- https://www.iar.com
- http://www.keil.com

7 Printf debugging

Printf debugging is one of the most straight-forward and used solution in order to start investigating a non-working system.

This chapter is a getting started guide to quickly setup a printf data path through semihosting, USART or SWO, benefiting from facilities offered by STMicroelectronics hardware kits and ecosystem tools.

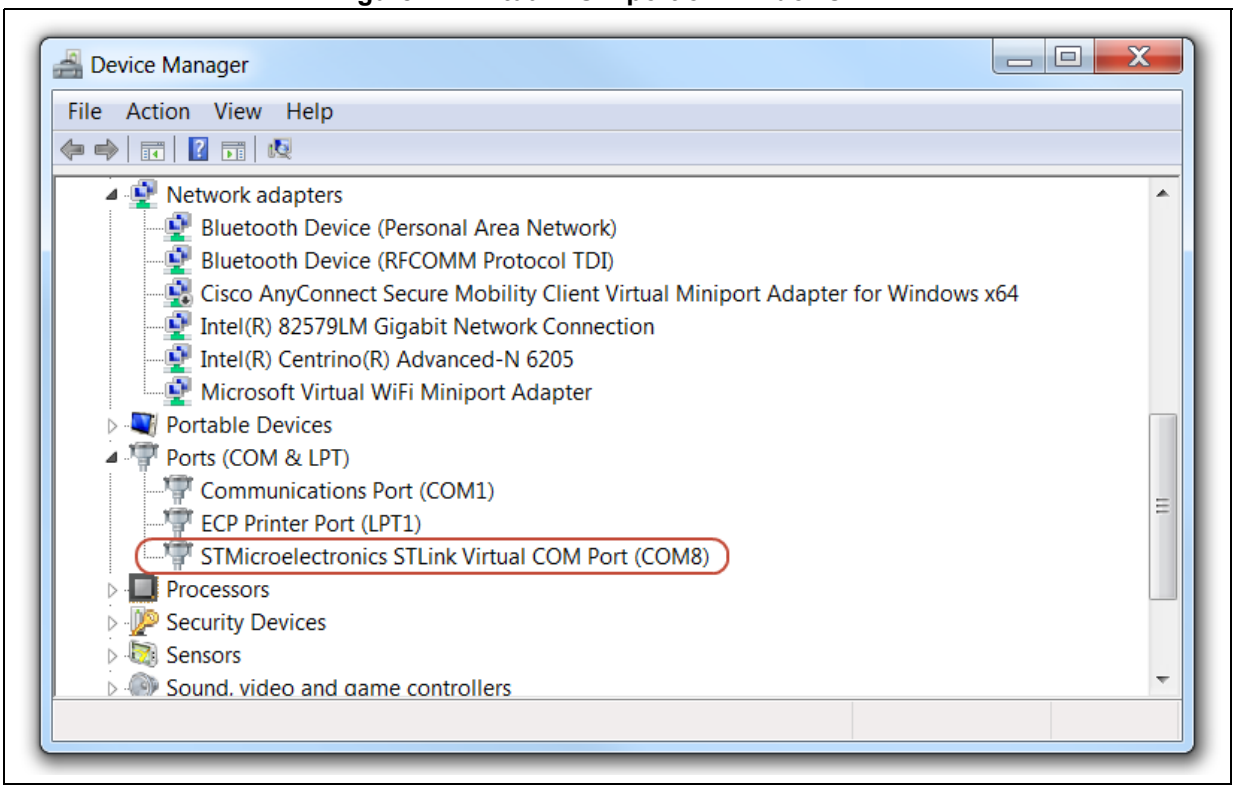
7.1 STM32 Virtual COM port driver

STM32 Virtual COM Port Driver (VCP) is a feature supported by ST-LINK/V2-B embedded in most of recent hardware kits (refer to *Section 2.1: Hardware development tools on page 9*). It is a RS232 emulation through ST-LINK USB connection.

On the PC side, this requires driver software package (STSW-STM32102) included in ST-LINK driver (STSW-0009).

Once the target is connected, it is seen as a serial port on the PC. An example is presented in *Figure 47*.

Figure 47. Virtual COM port on Windows® PC



7.2 Printf via UART

Direct connection from PC UART to board pinout does not work due to signal level incompatibility.

Take care to use external adapter (such as MAX232, ST3241EB, FTDI USB/UART) or the USART connected to Virtual COM port.

Trick: *Appendix B: Use Nucleo "cuttable" ST-LINK as stand-alone VCP on page 106* explains how to use ST-LINK Nucleo stand-alone part as VCP.

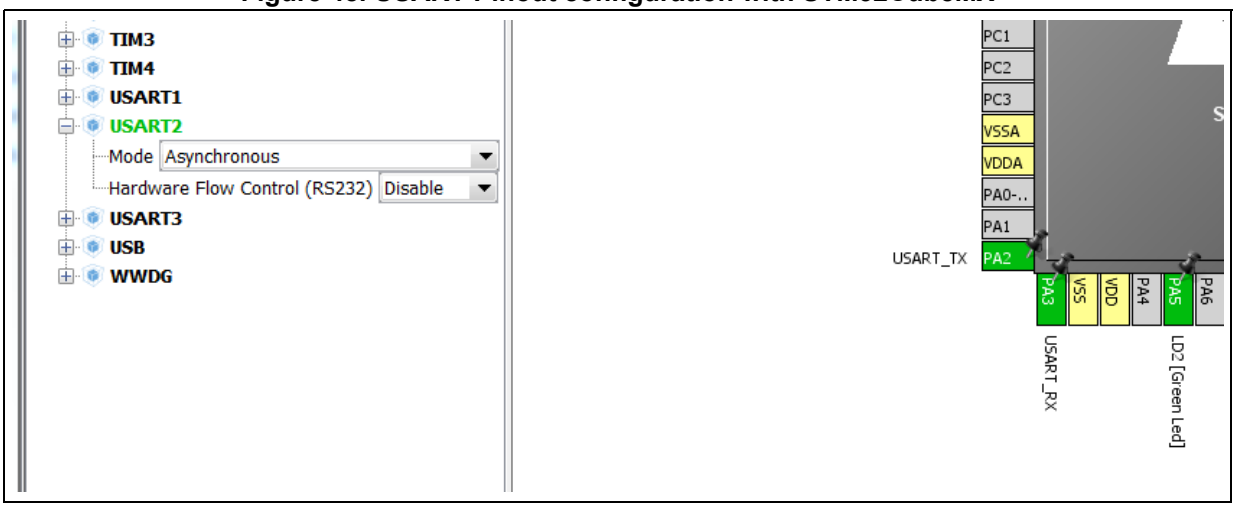
The straight-forward way to set a Serial Com port with PC host is to use the USART connected to VCP.

USART connected to VCP depends on the hardware kit:

- Nucleo-32/Nucleo-64: USART2 - PA2/PA3
- Nucleo-144: USART3 - PA9/PA10
- Discovery: not standard. Refer to the board schematics
- EVAL: not standard. Refer to the board schematics. Either the VCP or the RS232 connector can be used

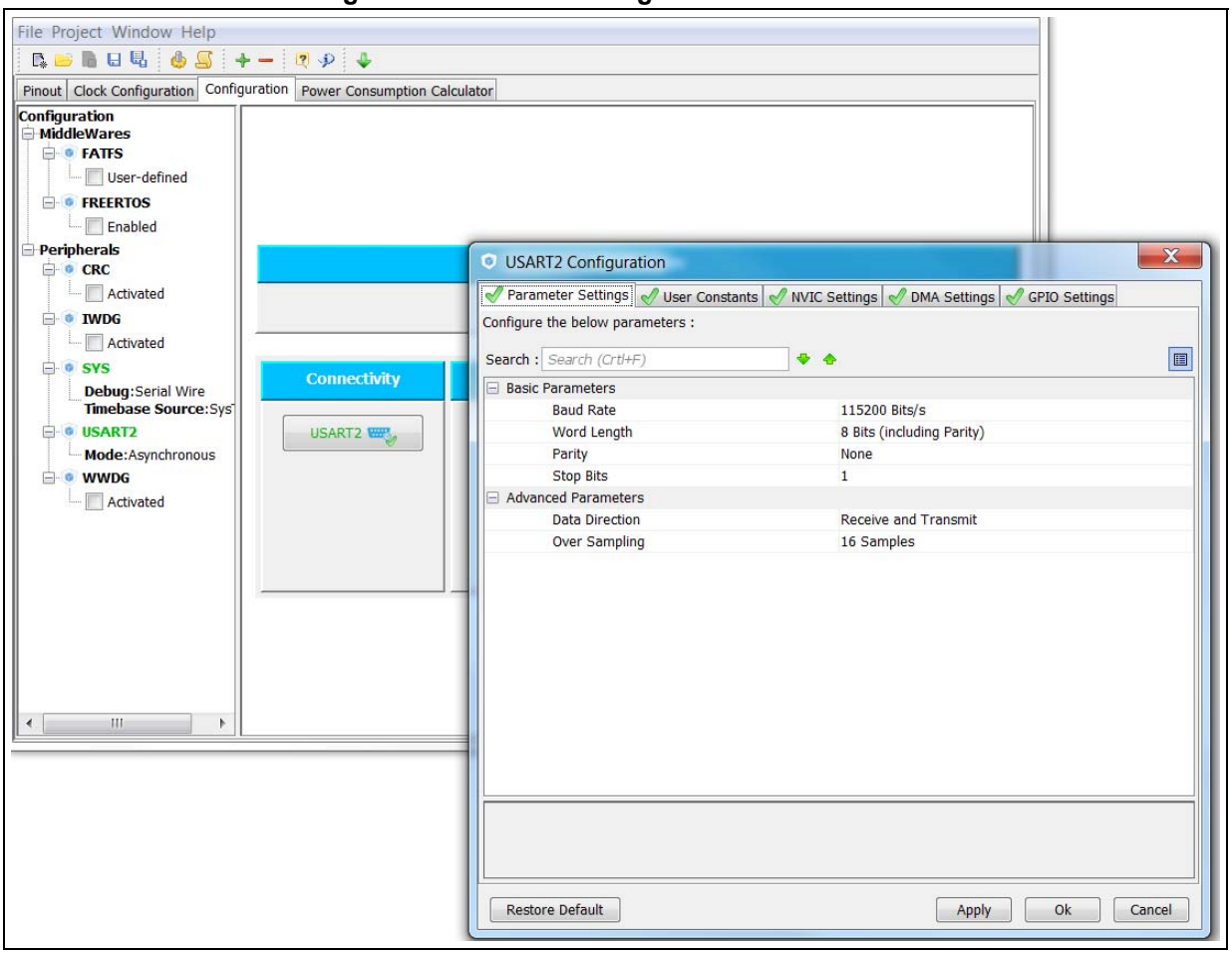
In STM32CubeMX, for Nucleo board, the VCP USART pins (PA2/PA3) are reserved by default, but required to be enabled by selecting "asynchronous" in USART mode selection box as shown in *Figure 48*.

Figure 48. USART Pinout configuration with STM32CubeMX



Then, set the UART communication settings in ***Configuration*** -> ***USART2 Configuration*** -> ***Parameter Settings*** as shown in *Figure 49*.

Figure 49. USART2 setting with STM32CubeMX



Retargeting printf to UART depends on the toolchain.

For IAR™ EWARM and Keil® MDK-Arm µVISION this is done by overriding the stdio fputc function

```
#include "stdio.h"

int fputc(int ch, FILE *f)
{

  HAL_UART_Transmit(&UartHandle, (uint8_t *)&ch, 1, 0xFFFF);

  return ch;
}
```

For GCC based toolset like STM32CubeIDE, two cases can be met.

With syscall.c integrated to the project:

```
#include "stdio.h"

int __io_putchar(int ch)
{
```

```
  HAL_UART_Transmit(&UartHandle, (uint8_t *)&ch, 1, 0xFFFF);

  return ch;
}
```

Without syscall.c integrated, a customized _write function has to be defined:

```
int _write(int file, char *ptr, int len)
{
int DataIdx;
   for (DataIdx = 0; DataIdx < len; DataIdx++){ __io_putchar( *ptr++ );}
   return len;
}
```

Refer to STM32Cube provided example `UART_Printf()` available for almost all STM32 Series. An example is available in STM32Cube_FW_F3_V1.7.0\Projects\STM32F303ZE-Nucleo\Examples\UART\UART_Printf.

Caution: USART word length includes parity which is not the case for most of UART terminal. Word length 8 with parity require 7 bits + parity on terminal side to match.

VCP does not support Word length of 7 bits and below (whatever the parity). *Table 5* gives examples of compatible configurations:

Table 5. STM32 USART vs. PC terminal WordLength example

STM32 UART	PC Terminal
Word Length: 8, Parity: Odd	Data: 7, Parity: Odd
Word Length: 8, Parity: None	Data: 8, Parity: None
Word Length: 9, Parity: Odd	Data 8, Parity: Odd
Word Length: 7, Parity: Odd/None	Not Working with VCP

7.3 Printf via SWO/SWV

Serial Wire Output (SWO) is single pin, asynchronous serial communication channel available on Cortex-M3/M4/M7 and supported by the main debugger probes.

It is using the ITM (instrumentation trace macrocell) module of the Cortex Core-Sight.

The asynchronous mode (SWO) requires 1 extra pin and is available on all packages for STM32 based on Cortex-M3, -M4, and -M7.

It is only available if a Serial Wire mode is used. It is not available in JTAG mode.

By default, this pin is NOT assigned. It can be assigned by setting the TRACE_IOEN and TRACE_MODE bits in the Debug MCU configuration register (DBGMCU_CR). This configuration has to be done by the debugger host.

Refer to the related chapter of STMicroelectronics reference manual.

In debug context it can be a good alternative to UART in system where pinout constraints are strong (alternate function preempting UART GPIOs).

It has to be used in combination with a Serial Wire Viewer (SWV) on host side which provides the following features:

- PC (Program Counter) sampling
- Event counters that show CPU cycle statistics
- Exception and Interrupt execution with timing statistics
- Trace data - data reads and writes used for timing analysis
- ITM trace information used for simple printf-style debugging

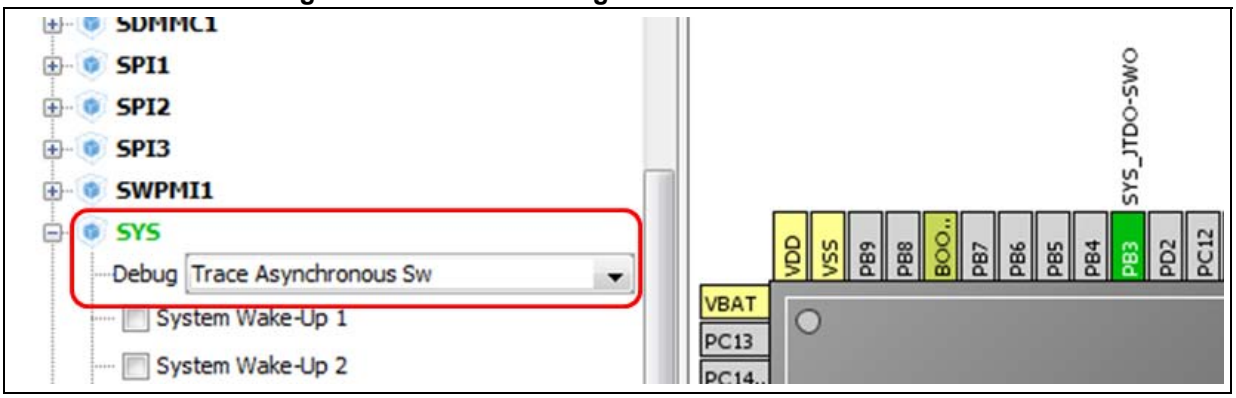
This chapter only addresses the printf-style debugging feature.

In order host debugger can manage flexible pin assignment ensure SWO pin is not used for other purpose.

In STM32CubeMX:

Select "Trace Asynchronous Sw" in ***SYS*** -> ***Debug*** selection box as shown in *Figure 50*.

Figure 50. SWO Pin configuration with STM32CubeMX



This secures that the PB3 is not allocated to another use. No specific code is generated.

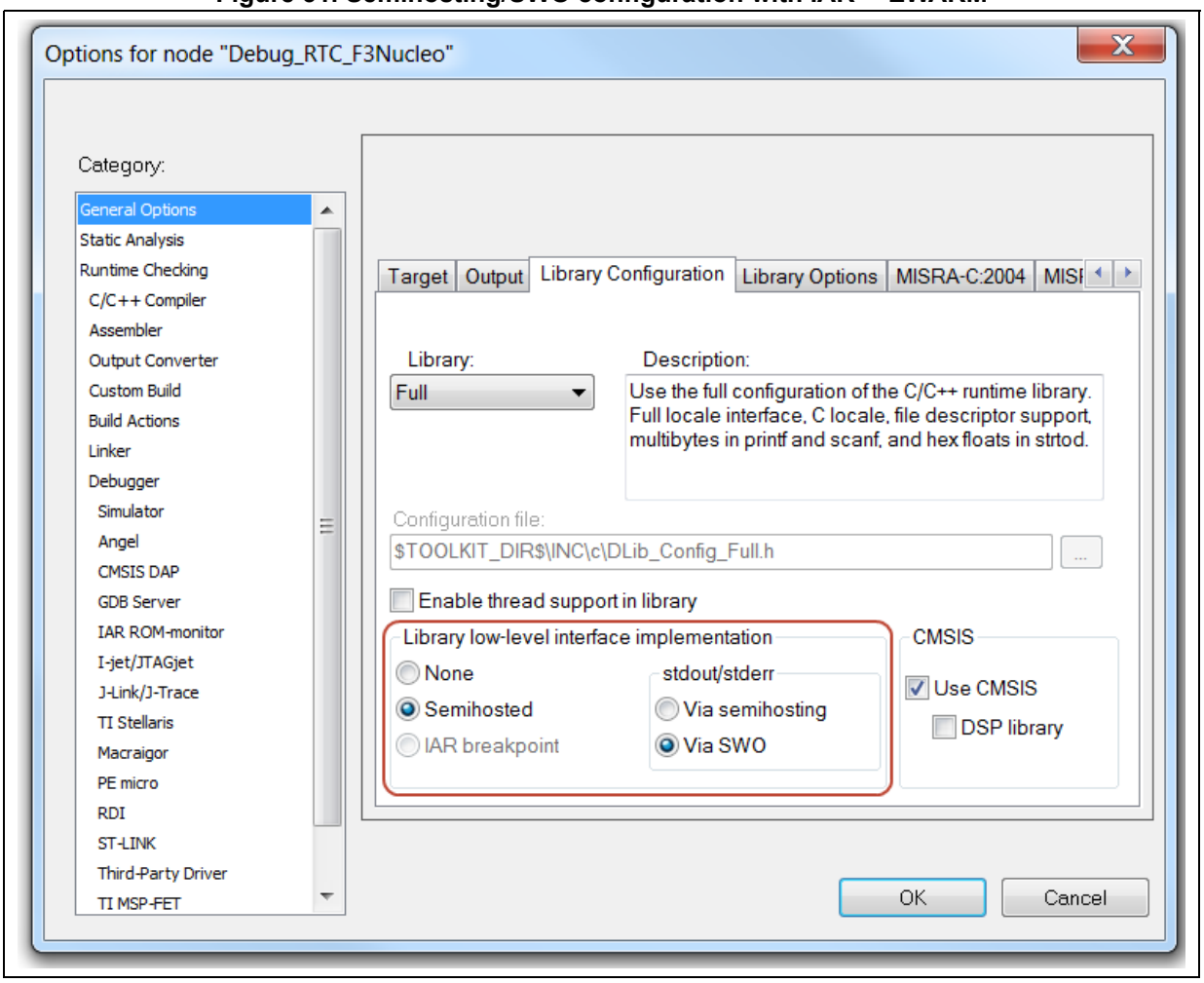
Other init steps are performed by the SWV integrated in the IDE or in the ST-LINK utility.

IAR™ EWARM

IAR™ EWARM provides an integrated access to SWO.

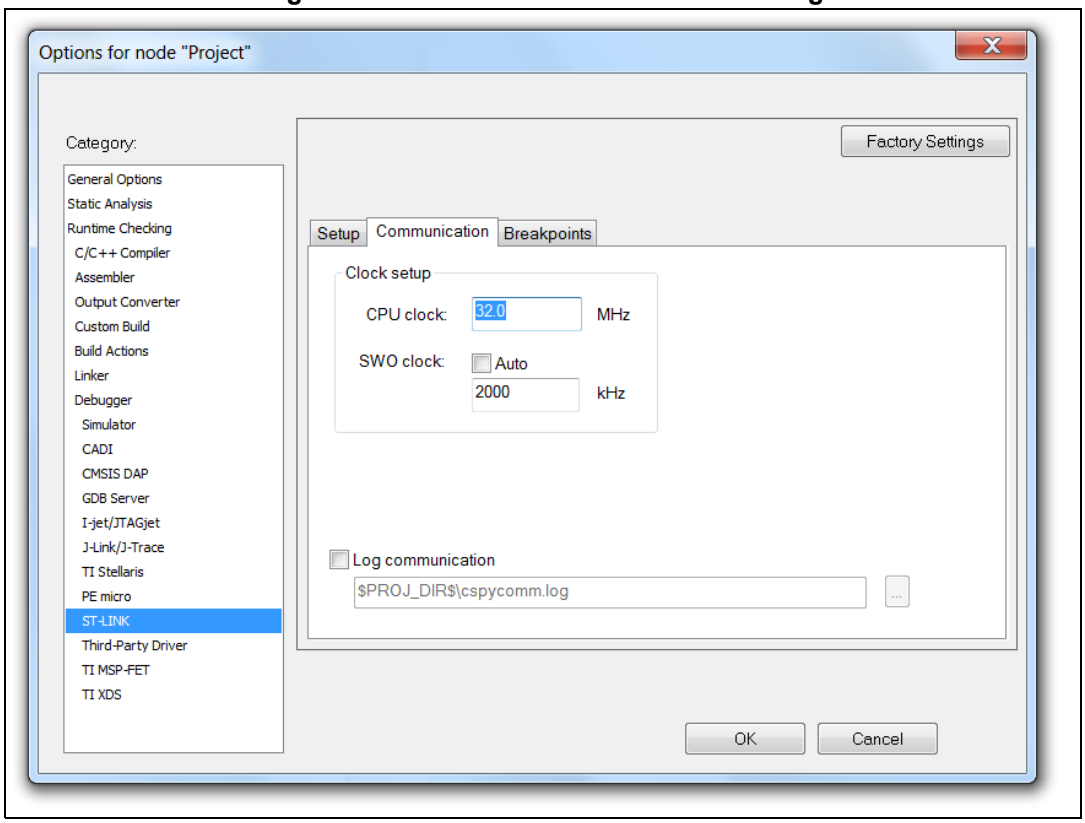
Redirection of printf and scanf is possible using Library Configuration options as shown in *Figure 51*.

Figure 51. Semihosting/SWO configuration with IAR™ EWARM



Care must be taken that clock setup is correct by using ***ST-LINK*** -> ***Communication Pane*** as illustrated in *Figure 52*.

Figure 52. IAR™ EWARM SWO Clock setting



Once configured, IAR™ EWARM properly sets TRACE_IOEN and TRACE_MODE and configures the related GPIO.

SWO printf occurrences are visible in Terminal I/O windows.

Port Stimulus 0 is used by printf and scanf. It is not configurable.

Keil® MDK-Arm µVISION:

In MDK-Arm it is required to redirect printf to SWO by some piece of code following same model as for UART (Refer to *Section 7.2: Printf via UART on page 69*)

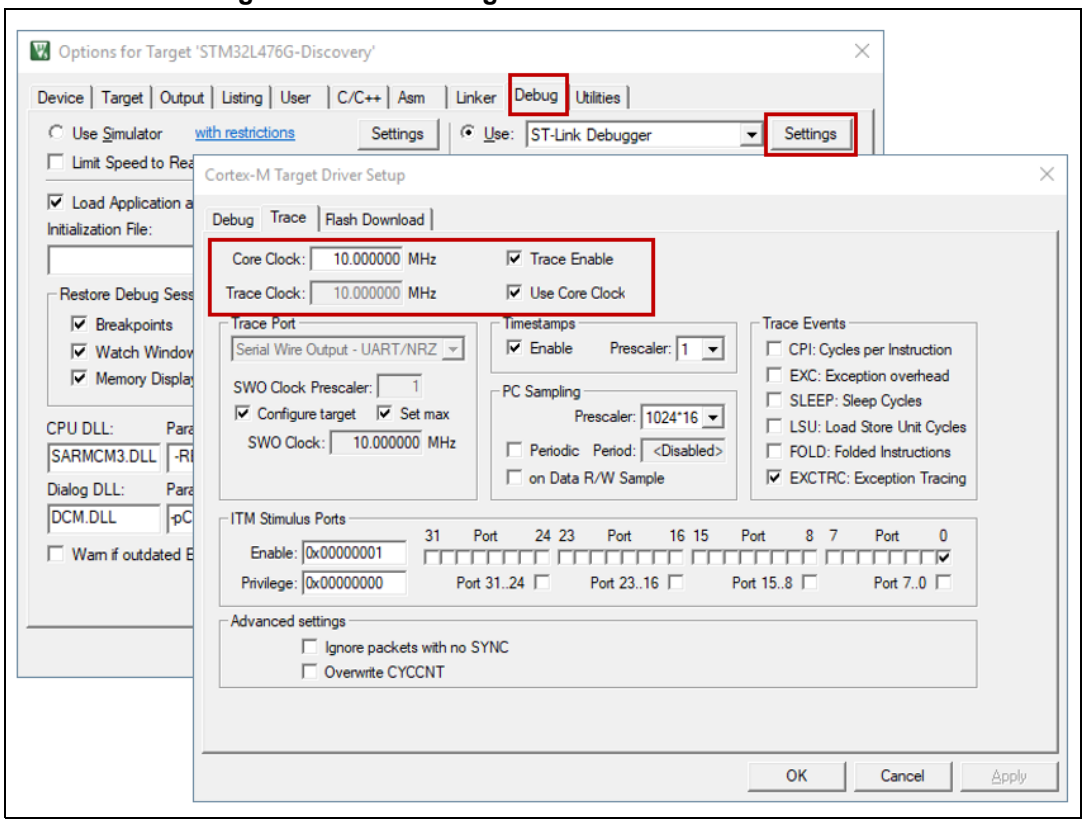
```
#include "stdio.h"

int fputc(int ch, FILE *f)
{
  ITM_SendChar(ch);
  return(ch);
}
```

Keil® must be properly configured for the SWO communication to be properly set. An example is given in *Figure 53*.

In ***Projet Option*** -> ***Debug*** -> ***Probe Settings*** -> ***Trace Pane:***

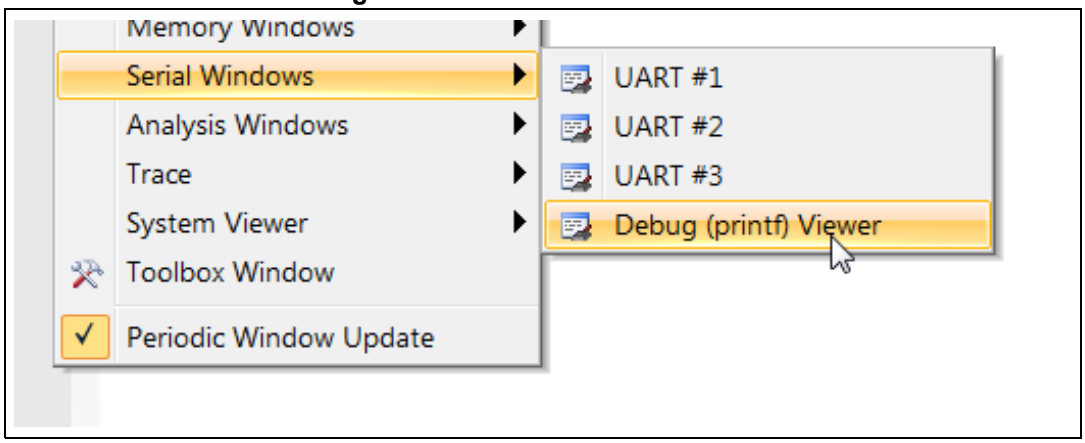
1. Tick Trace Enable
2. Enter correct Core Clock
3. Enable ITM Stimulus Port 0

Figure 53. SWO configuration with Keil®



SWV viewer is called “Debug (printf) Viewer” and is accessible while in debug through ***View*** -> ***Serial Windows*** -> ***Debug (prinf) Viewer*** as shown in *Figure 54*.

Figure 54. Access to SWV in Keil®



Tip: Keil® MDK-Arm µVISION allow to select the Stimulus to display. On the other hand it is quite straight forward to make some clone of ITM_SendChar() function using any of the 31 stimulus port. Can be useful in a very verbose system to set a trace

library which split trace between stimulus based on their importance (info, debug, error) or there source.

STM32CubeIDE

With STM32CubeIDE you also have to redirect printf to SWO by some piece of code.

With syscall.c integrated to the project:

```
#include "stdio.h"
int __io_putchar(int ch)
{
 ITM_SendChar(ch);
 return(ch);
}
```

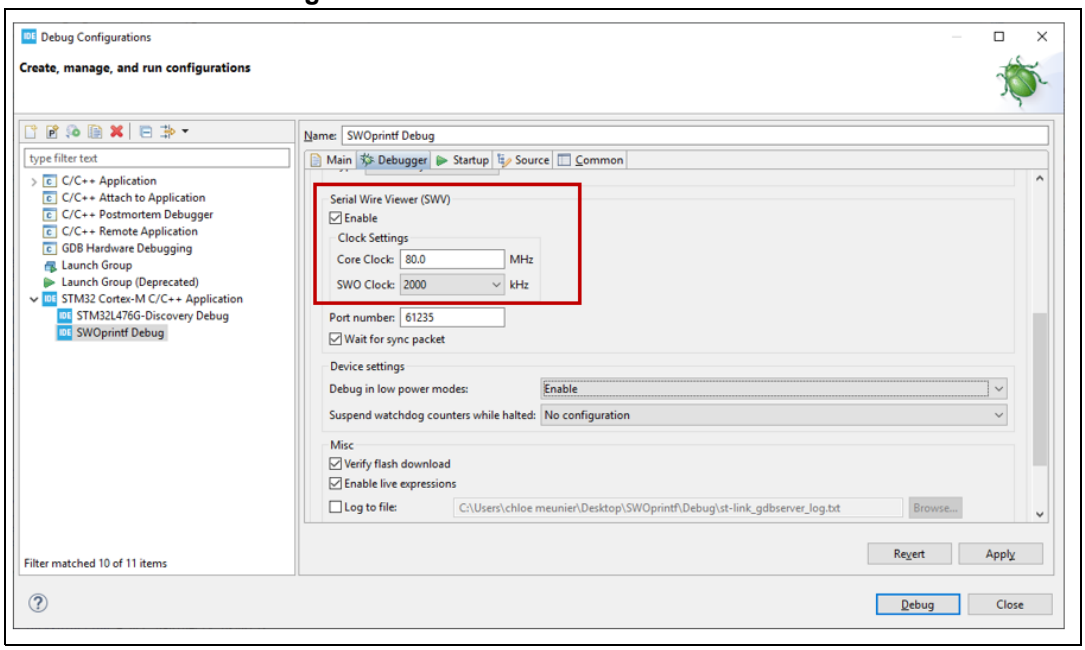
Without syscall, add:

```
int  _write(int file, char *ptr, int len)
 {
  int DataIdx;
  for (DataIdx = 0; DataIdx < len; DataIdx++)
  {
  __io_putchar(*ptr++);
  }
  return len;
 }
```

Enable SWD in Debug configuration ? Debugger pane (see *Figure 55*).

Core clock must be the same as Cortex clock. You can then start the debug session.

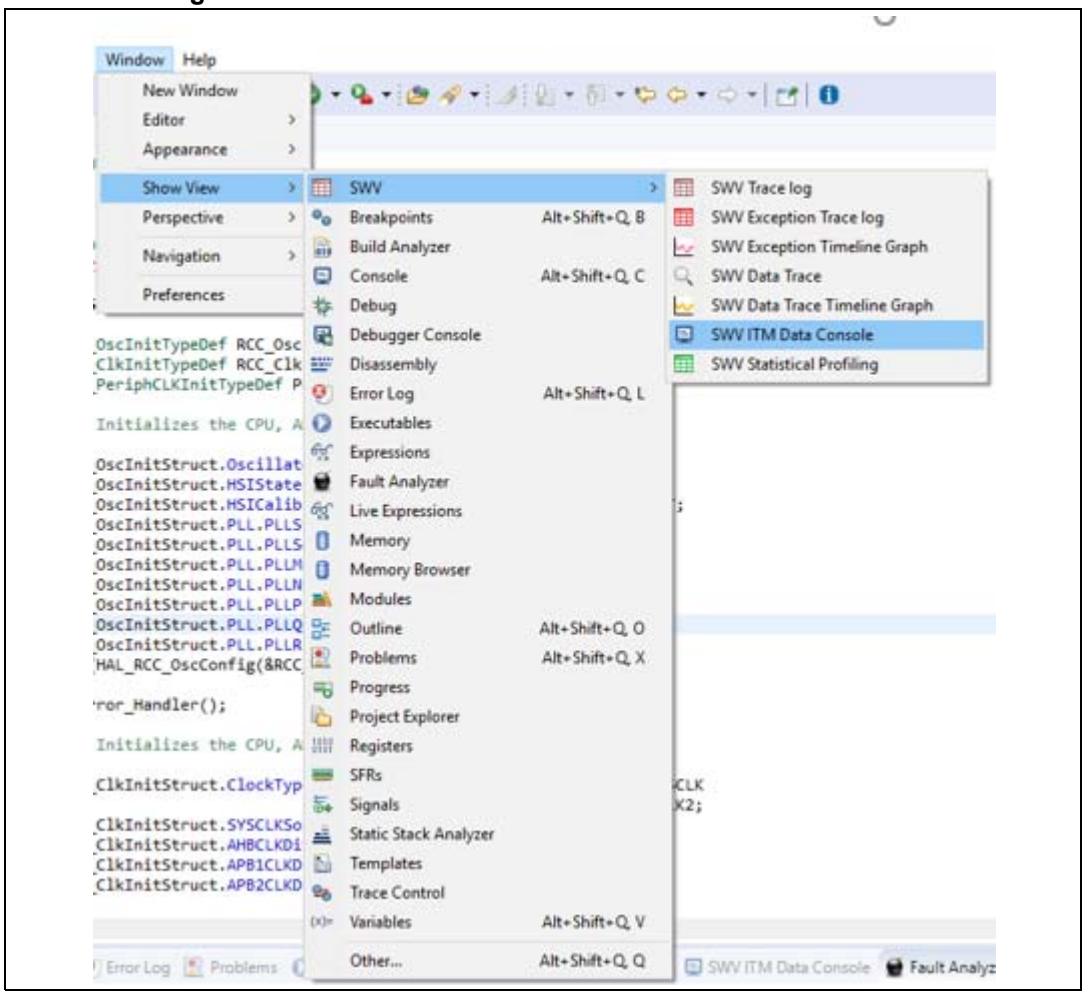
Figure 55. Enable SWD in STM32CubeIDE



Enable SWV ITM Data Console in

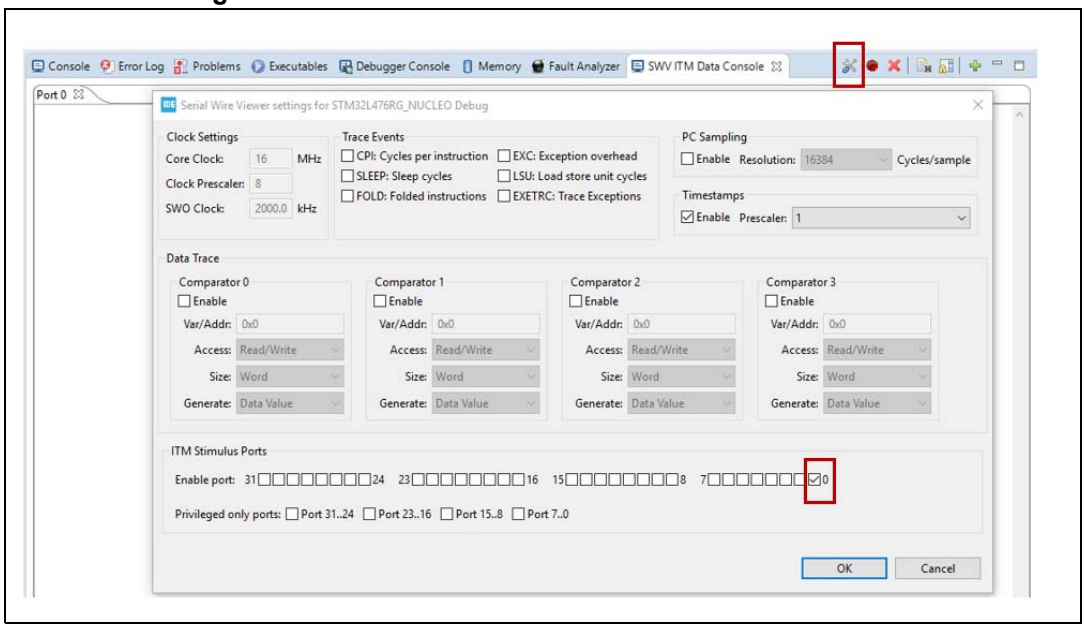
Window ->Show View -> SWV -> SWV ITM Data Console as shown in the figure below:

Figure 56. Enable SWV ITM Data Console in STM32CubeIDE



Enable ITM Stimulus Port 0 after clicking on “Configure trace” as shown in the below figure.

Figure 57. Enable ITM stimulus Port 0 in STM32CubeIDE



Click on “Start Trace” button

Figure 58. Start Trace button in STM32CubeIDE

Press “Resume” button, and your printf message is printed in SWV ITM Data Console.

7.4 Semihosting

Semihosting is a mechanism that enables code running on an Arm® target to communicate and use the Input/Output facilities on a host computer that is running a debugger.

Examples of these facilities include keyboard input, screen output, and disk I/O. For example, this mechanism can be used to enable functions in the C library, such as `printf()` and `scanf()`. It can also allow to use the screen and keyboard of the host instead of having a screen and keyboard on the target system.

This is useful because development hardware often does not have all the input and output facilities of the final system. Semihosting enables the host computer to provide these facilities.

However, the user has to be aware of the following drawbacks:

- Semihosting only works during a debug session. Otherwise, the program gets stuck in the first `printf()` routine reached.
- Since semihosting uses breakpoint instruction and host dependent code, it has significant and unpredictable impact on performance.

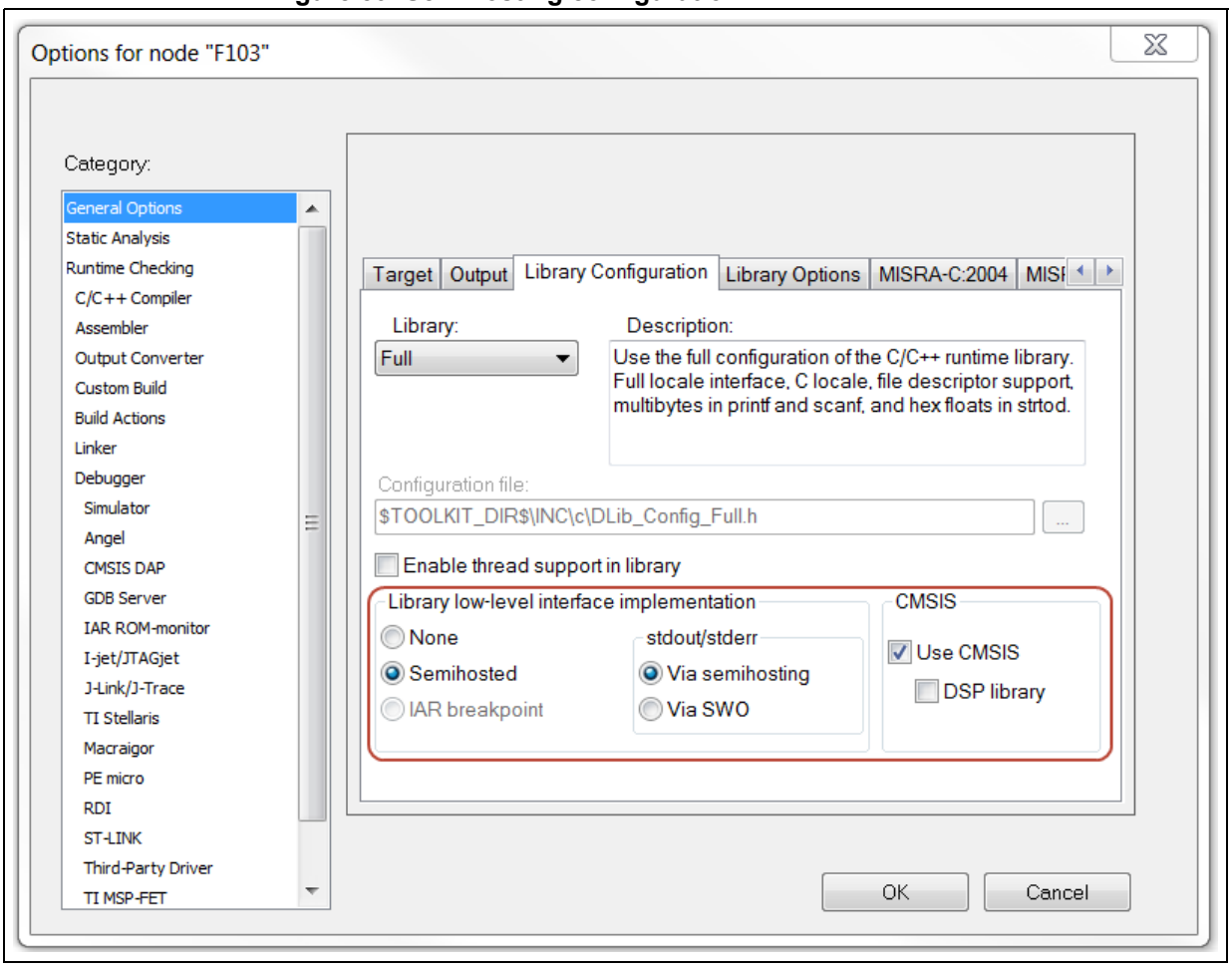
Semihosting depends on the library provided by the IDE. The next sections present how to set semihosting using the three main IDEs covered in this application note.

7.4.1 IAR™ EWARM

IAR™ EWARM provides a highly integrated semihosting feature, enabled by default.

Figure 59 shows how to check if it is the case for the project in ***Options -> General options -> Library Configuration Pane.***

Figure 59. Semihosting configuration in IAR™ EWARM



In such a case, simply use `printf()` / `scanf()` functions in the code.

Input and output of the program are displayed in the Terminal I/O window.

7.4.2 Keil® MDK-Arm µVISION

Keil® has no semihosting capability.

7.4.3 STM32CubeIDE

Set linker parameters

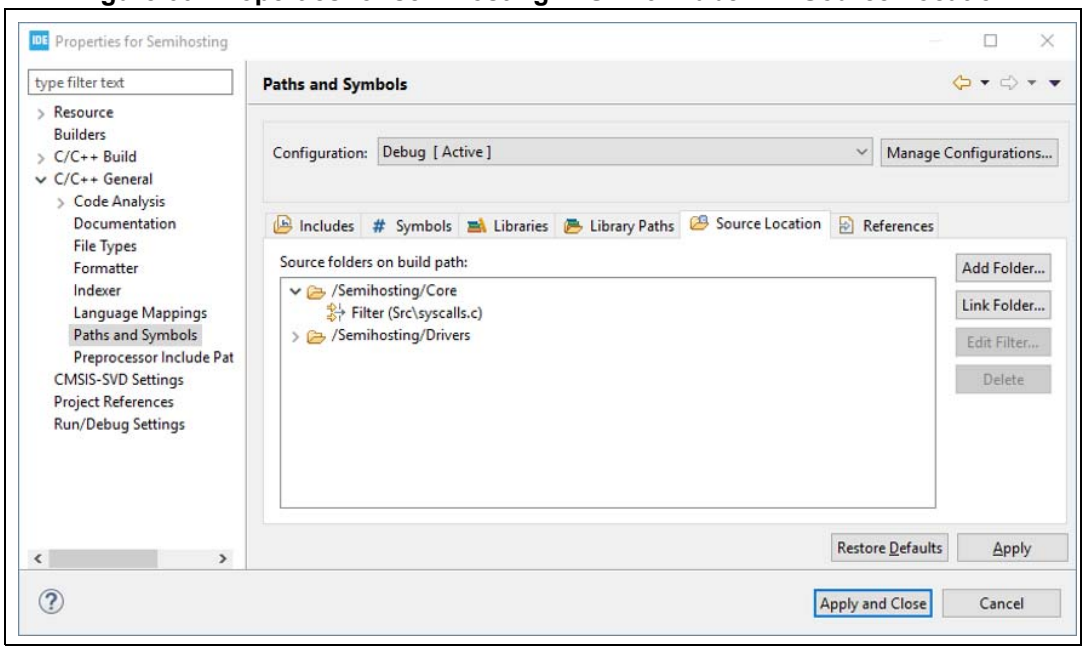
First the linker must ignore the default syscalls.c file and has to use the newlib-nano librairies, which contains printf() function.

in ***Project -> Properties -> C/C++ General -> Paths and Symbol***

Click on the Source Location tab. Click on the arrow near to “[Project name]/Core”, and select “Filter(empty)”.

Then click on “Edit filter” button and add “syscall.c” to the Exclusion patterns list.

Figure 60. Properties for semihosting in STM32CubeIDE- Source Location

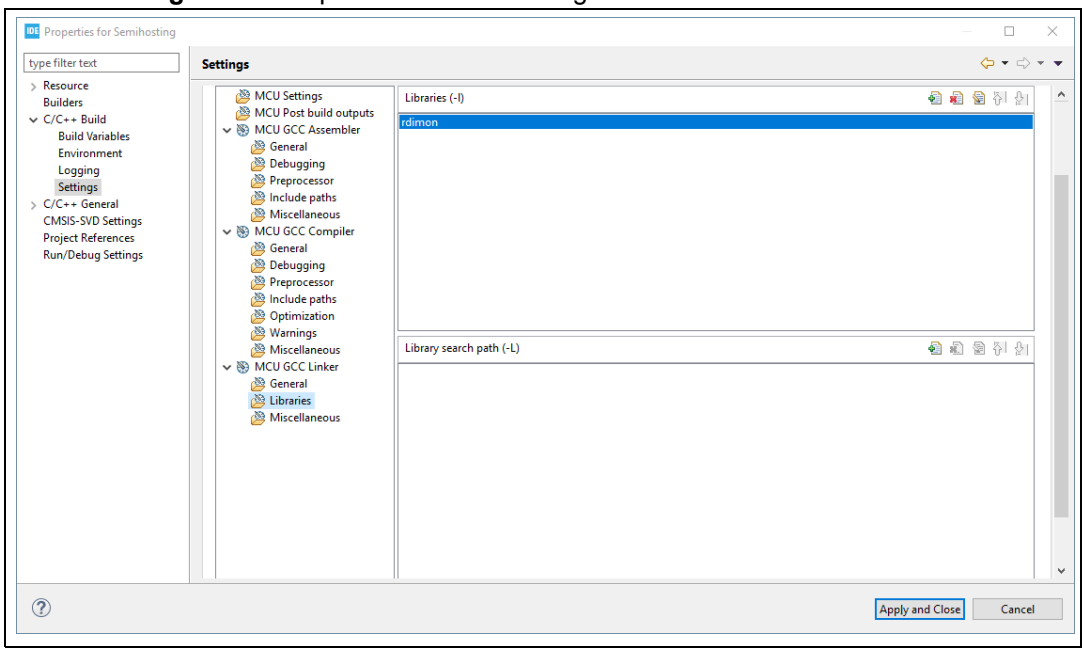


To use semihosting the librdimon must be enabled. Librdimon implements the semihosting versions of syscalls from newlib.

On the left-side pane, go into **C/C++ Build -> Settings** and select the Tool Settings tab.

Then, select MCU GCC **Linker -> Libraries**. In the libraries pane, click the “Add” button and enter rdimon.

Figure 61. Properties for semihosting in STM32CubeIDE- Librairies

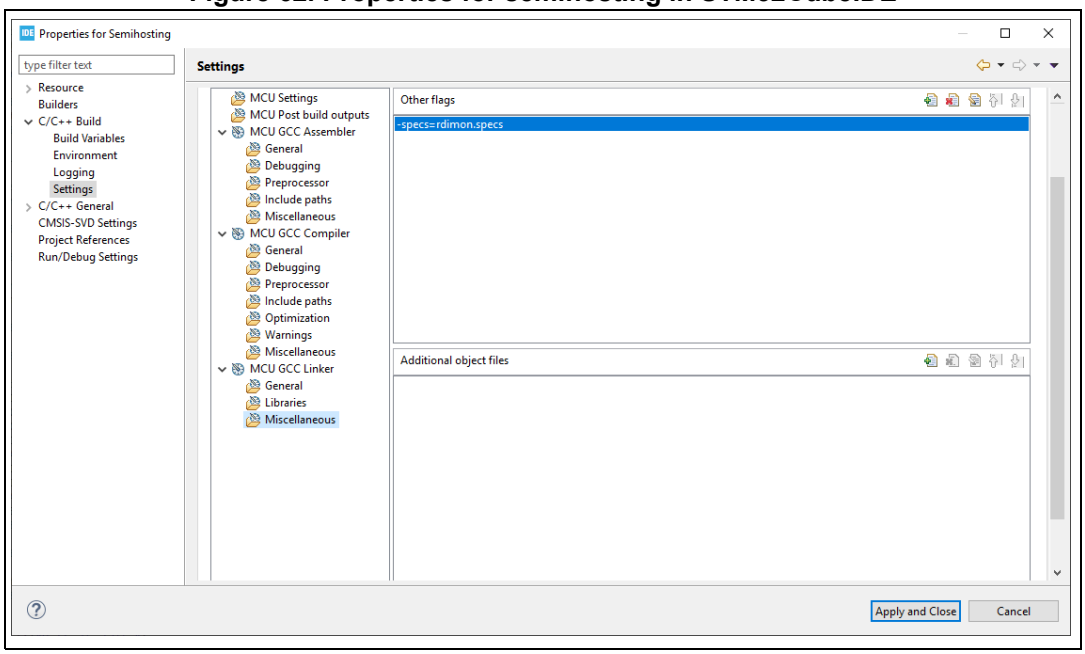


Next, select **MCU GCC Linker -> Miscellaneous** while still in the Tool Settings tab.

Click the Add… button and enter -specs=rdimon.specs into the dialog box.

This add the linker flags in order to include the librdimon library.

Figure 62. Properties for semihosting in STM32CubeIDE



Add printf Code

Above int main(void) (USER CODE 0 section), add:

```
extern void initialize_monitor_handles(void);
```

Then configure the semihosting system call: In int main(void) before the while(1) loop (USER CODE 1 section) add:

```
Initialise_monitor_handles();
```

Then inside the while(1) loop, add:

```
Printf("Hello World!\n");
HAL_Delay(1000);
```

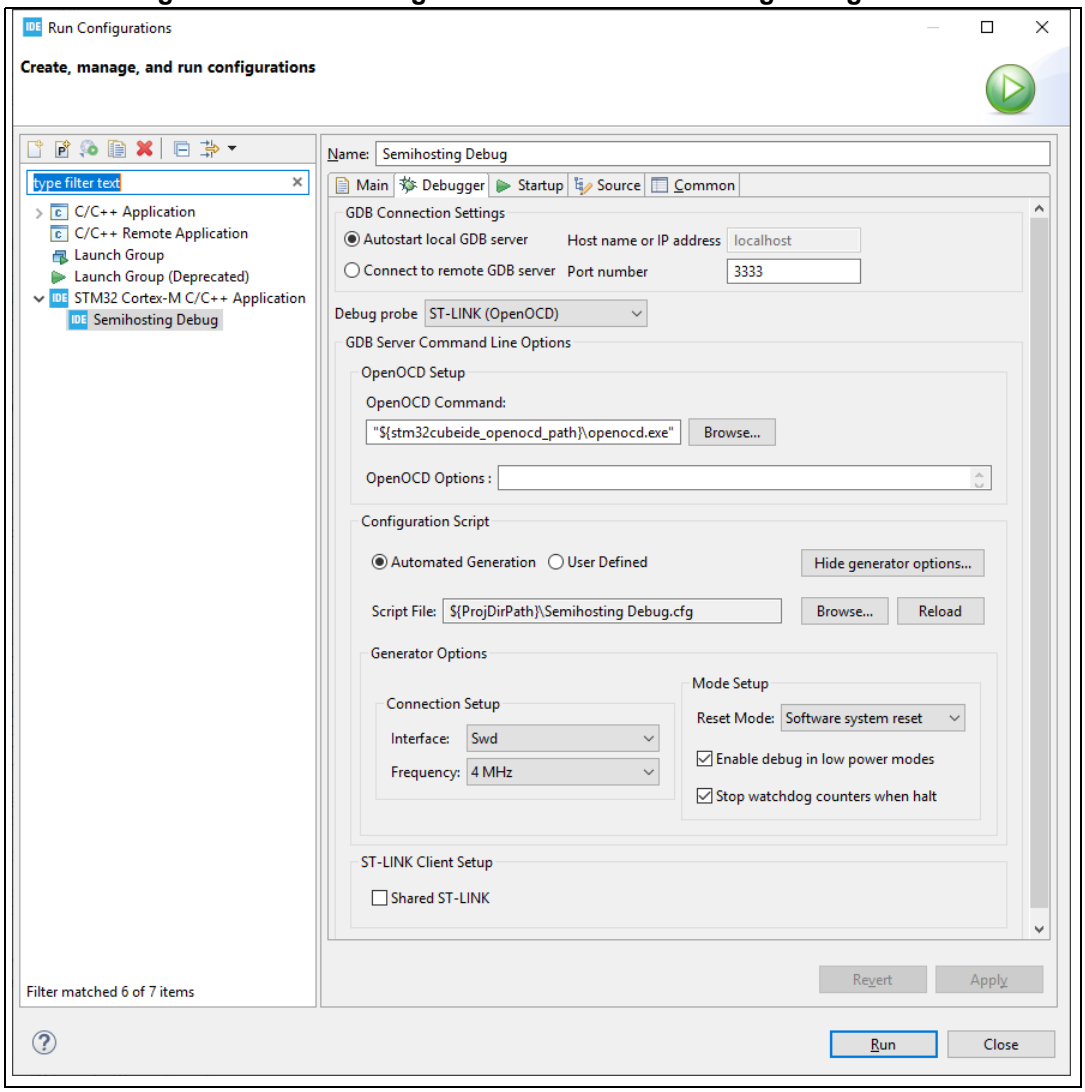
Click **Project -> Build Project** to compile and link everything.

Debug configuration

In **Run -> Debug configuration -> Debugger tab**, change the debugger probe to ST-LINK (Open OCD).

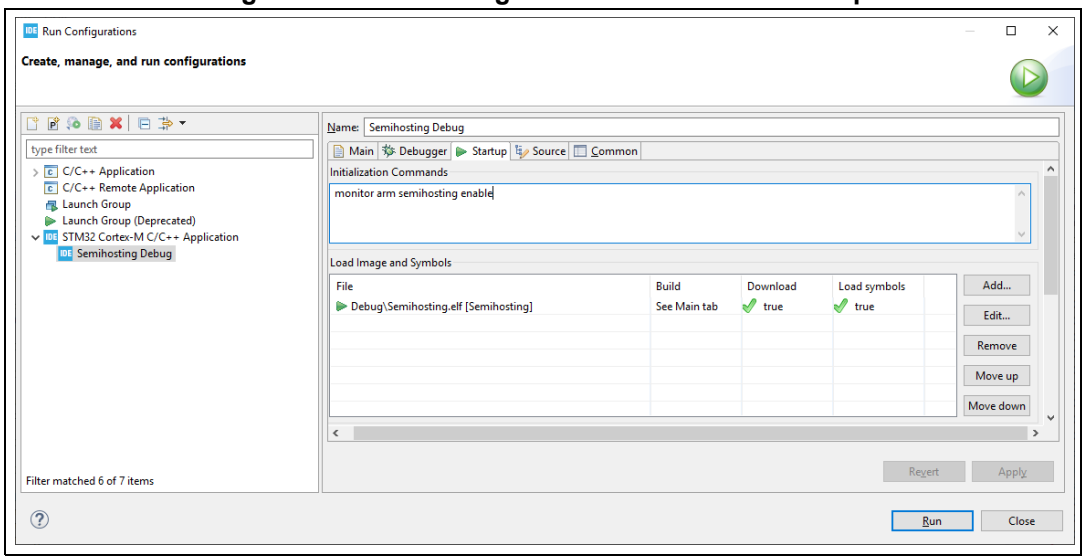
In Generator options, choose “Software system reset“as reset Mode.

Figure 63. Semihosting in STM32CubeIDE – Debug configuration



In the Startup tab enter the command: monitor arm semihosting enable.

Figure 64. Semihosting in STM32CubeIDE – Startup

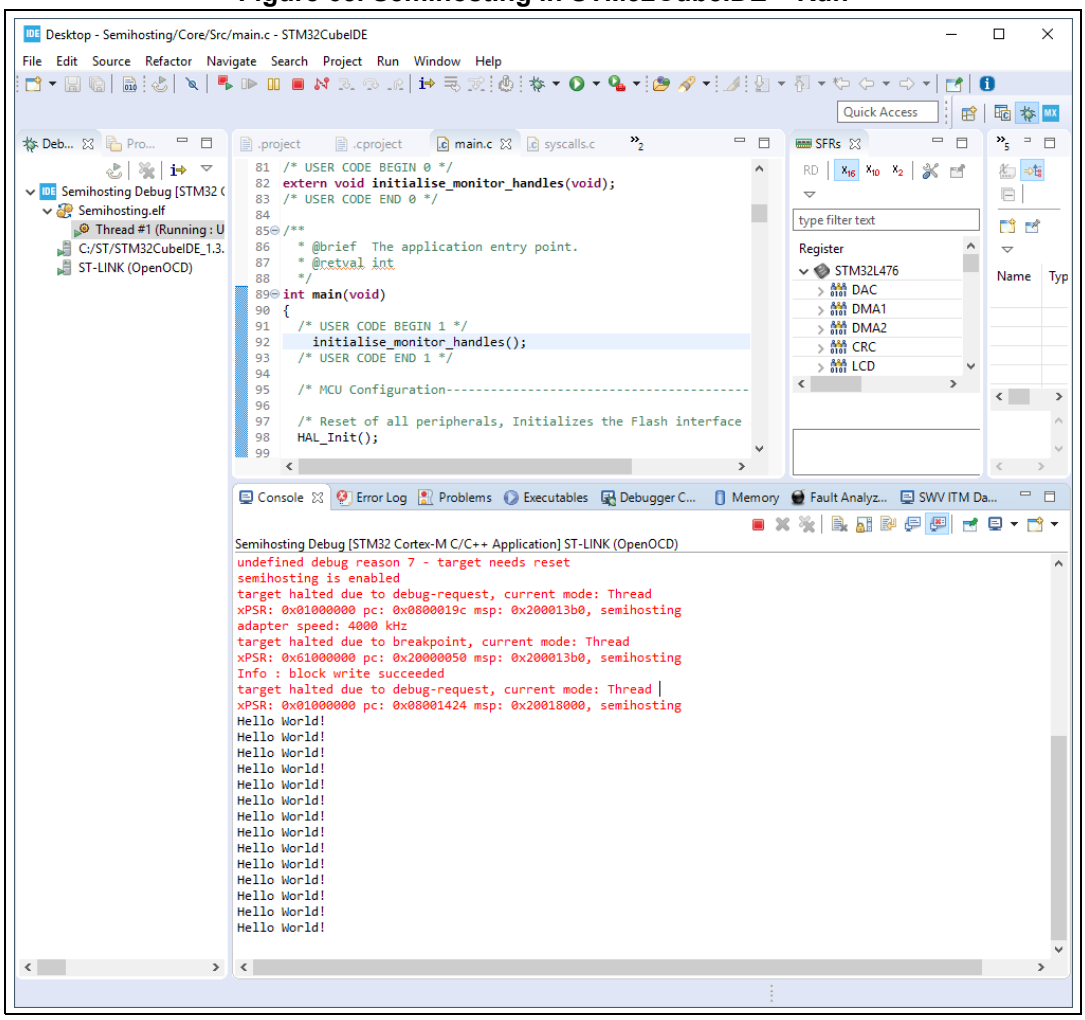


Click on Debug button.

Run

In the debugging perspective, click **Run -> Resume**, and you should see “Hello, World!” being printed at the bottom of the console once per second.

Figure 65. Semihosting in STM32CubeIDE – Run



8 Debug through hardware exploration

As a complement to software instrumentation, a user facing a non-working system may take great advantage to monitor STM32 pin states (GPIO or clock among others) with external tools such as oscilloscopes or logic analyzers.

This chapter presents the possibilities offered by STMicroelectronics hardware kits and integrates a complete tutorial to setup the microcontroller clock output (MCO)

8.1 Easy pinout probing with STMicroelectronics hardware kits

All STMicroelectronics hardware kits presented in *Section 2.1.1 on page 9* offers easy pinout access thanks to their Morpho or ARDUINO® connectors.

The coverage of the pinout by the connectors depends on the board itself as well as on the MCU type. In most cases, a large number of GPIOs are covered.

In order to use this coverage at best, the user is advised to study the board schematics that show the connections between the MCU pins and the connectors. In association with the schematics, the board user manual presents the jumper and solder bridge configurations that modify the routing of pins to connectors.

8.2 Microcontroller clock output (MCO)

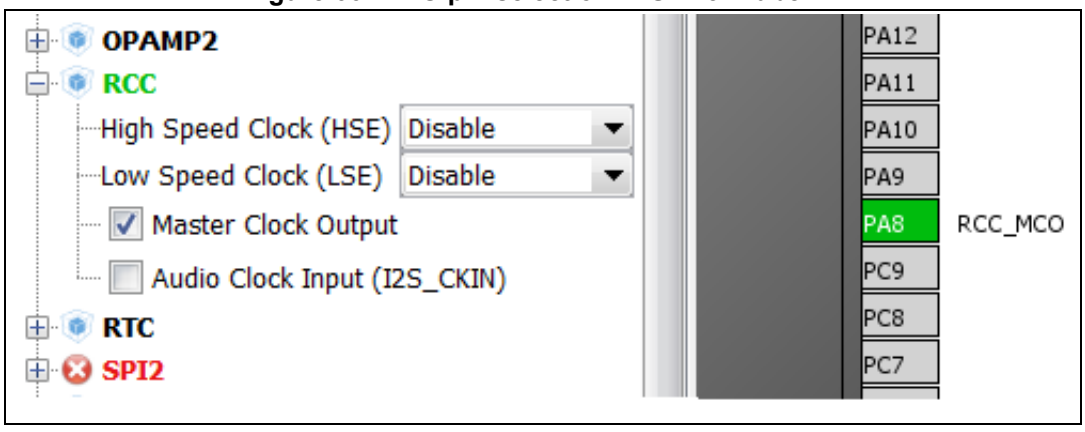
This feature allows to output one or more internal clock to one or more pins in order to enable measurement through an external tool, typically an oscilloscope.

It can be useful in debug context in order to check that clock settings is as per expectation and help to investigate potential error in clock tree initialization code.

8.2.1 Configuration with STM32CubeMX

In STM32CubeMX, MCO stands for master clock output. It is enabled by ticking the Master Clock Output option in the RCC section as shown in *Figure 66*.

Figure 66. MCO pin selection in STM32CubeMX



This allocates a pin labeled RCC_MCO.

This is typically pin PA8 for all STM32 families.

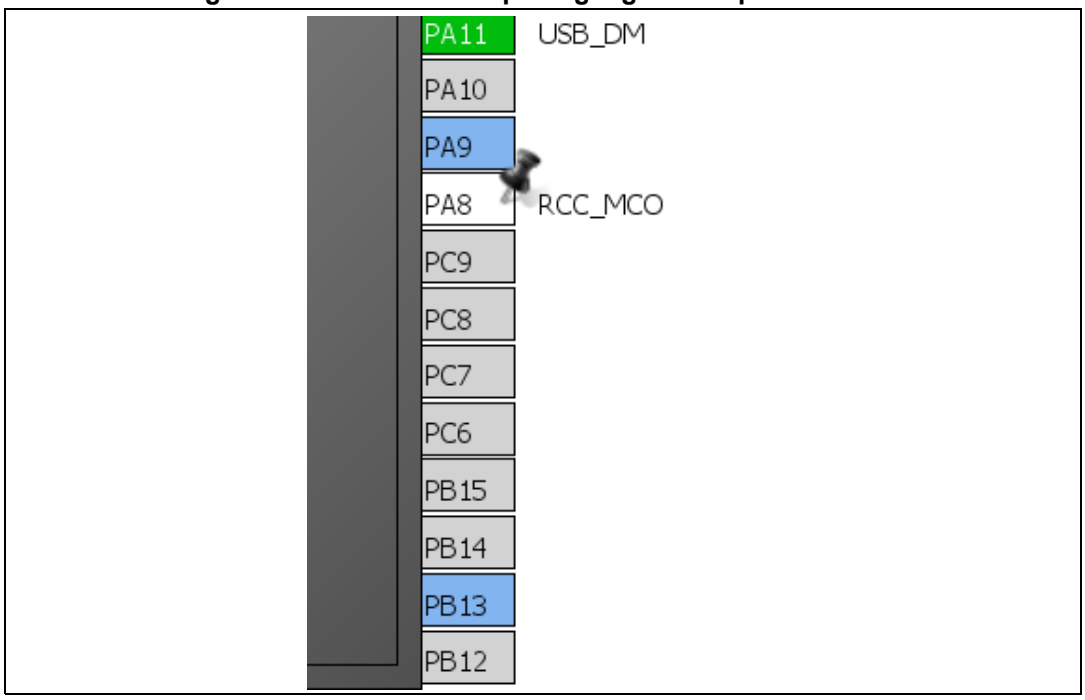
For Nucleo kits, the PA8 pin is accessible on the D7 pin of the ARDUINO® connector.

For other board pin configuration, please refer to the board schematics.

Depending on board or and chip families, other pins can be used if needed and available.

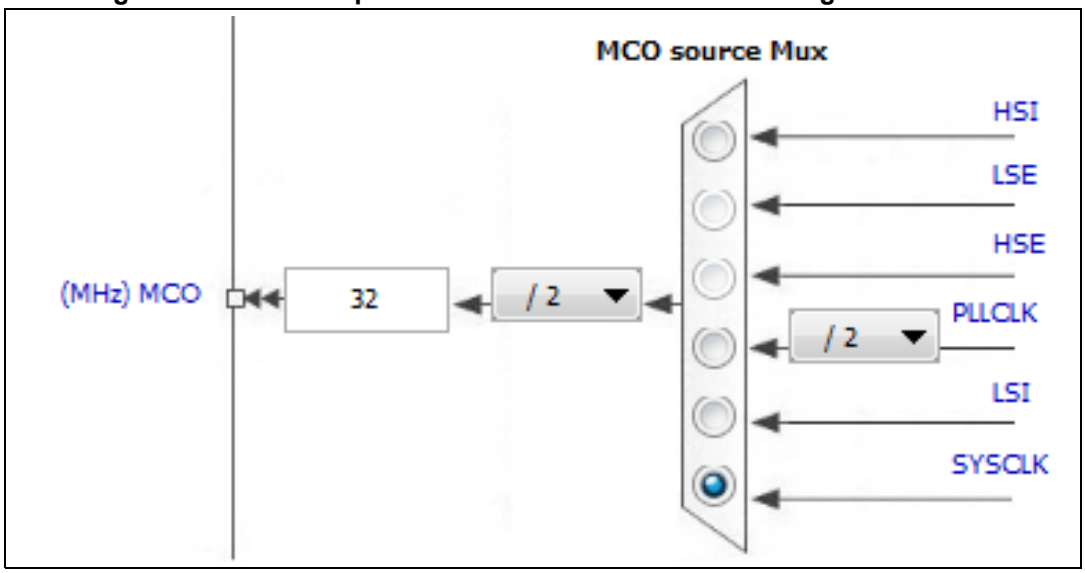
The *Ctrl* + *click* on RCC_MCO pin command sequence under STM32CubeMX highlights in blue the alternate pin. An example is shown in *Figure 67*.

Figure 67. MCO alternate pin highlight exemple with L073



STM32CubeMX Clock Configuration pane selects the signal to route to pin and the divider as presented in *Figure 68*.

Figure 68. MCO Multiplexer in STM32CubeMX Clock Configuration Pane



The divider allows to output a signal frequency compatible with output capabilities.

8.2.2 HAL_RCC_MCOConfig

Independantly of the fact that STM32CubeMX is used or not, MCO configuration is done using the hal_rcc or LL function:

stm32XXxx_hal_rcc.c/ stm32XXxx_hal_rcc.h

```
void HAL_RCC_MCOConfig( uint32_t RCC_MCOx, uint32_t RCC_MCOSource, uint32_t RCC_MCODiv)
```

Examples based on LL drivers are available in STM32Cube libraries (refer to STM32CubeProjectList.html) which configure the GPIO and the related registers depending on source and divider.

They also configure the selected GPIO accordingly:

```
/* Configure the MCO1 pin in alternate function mode */
    GPIO_InitStruct.Pin = MCO1_PIN;
    GPIO_InitStruct.Mode = GPIO_MODE_AF_PP;
    GPIO_InitStruct.Speed = GPIO_SPEED_HIGH;
    GPIO_InitStruct.Pull = GPIO_NOPULL;
    GPIO_InitStruct.Alternate = GPIO_AF0_MCO;
    HAL_GPIO_Init(MCO1_GPIO_PORT, &GPIO_InitStruct);
```

Caution: The setting of GPIO speed (OSPEED) must be carefully set.

OSPEED setting and maximum output frequency value are described in the datasheet of the related MCU in chapter *I/O port characteristics.*

Max Frequency values are given for a typical load of 50 pF or 10 pF.

If the measure is performed with an oscilloscope, the load of the probe circuitry must be taken into account.

If the frequency of the signal under observation exceeds the GPIO capability (e.g. 216 MHz Sysclock on F7 while GPIO maximum frequency is 100 MHz), use a divider to produce a suitable signal.

The default value in RCC HAL function is the highest (which is good).

In case of a STM32CubeMX generated project, be aware that default value applied in generated `MX_GPIO_init()` function (executed after MCO config) is the lowest.

In case the output clock is higher than 1 MHz, it is recommended to change this.

A too low OSPEED setting can be suspected in case no signal or very noisy/flatten signal (small amplitude).

A too high setting can be suspected if a signal with a long and high amplitude dumping oscillation is observed (overshoot / undershoot).

8.2.3 STM32 Series differences

STM32L4 Series also provides an LSCO (Low Speed Clock Output) on PA2 in order to output LSE or LSI, same as MCO, but with benefit to be still available during stop and standby mode.

Refer to section 6.2.15 Clock-out capability of STMicroelectronics reference manual *STM32L4x5 and STM32L4x6 advanced Arm®-based 32-bit MCUs* (RM035) for details.

HAL Function to call is:

```
void HAL_RCCEx_EnableLSCO(uint32_t LSCOSource)
```

in stm32l4xx_hal_rcc_ex.c/.h

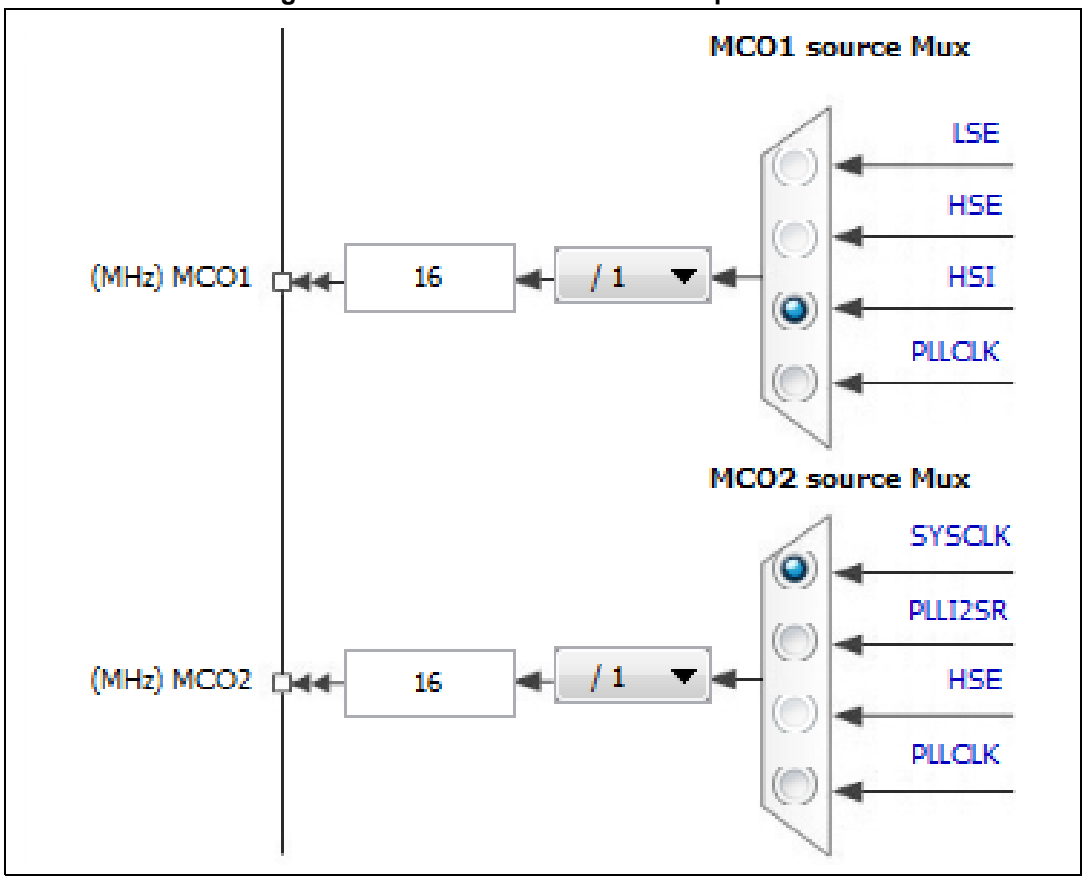
Note: LSCO is conflicting with UART2 TX (PA2). On Nucleo-64 board, the use of the LSCO board use and the use of the ST-LINK VCP are mutually exclusive.

SB63 must be set in order to get the LSCO signal available on Morpho and ARDUINO® connectors.

Refer to the board user manual for details.

STM32F4 and STM32F7 Series devices provide two different MCO outputs given choice of four clocks each as shown in *Figure 69*. Refer also to *Appendix D: Cortex®-M debug capabilities reminder on page 116*.

Figure 69. STM32F4/F7 dual MCO capabilities



9 Dual-Core microcontroller debugging

STM32H7x5/x7 Series are dual core microcontrollers using heterogeneous core architecture: An Arm Cortex-M7 core and an Arm Cortex-M4 core.

Debug process is different for these dual core microcontrollers as we need a simultaneous debug of both cores using a single hardware debug probe.

Note: Refer to AN5286 and AN5361, both available on st.com which explain how to proceed to debug dual core with IAR™ EWARM (AN5286), MDK-Arm (AN5286) and STM32CubeIDE (AN5361).

Dual debug is supported using

– STM32CubeIDE, IAR™ EWARM starting from version 8.30, or MDK-Arm version v5.25 and later

– ST-Link server starting from version v1.1.1-3

10 From debug to release

It is important to have in mind that most of technics presented in this AN and suitable for debugging have to be properly cleaned to prevent problem while releasing the application.

The following action list can be used as a checklist helping to avoid the most common problems:

- Remove software BKPT instructions or take care to let them inside #ifdef DEBUG statements.
- Ensure `printf()` uses available data path on final product.
  Semihosting and SWO cause hardfault otherwise.
- Reestablish Code Optimization level.
- Implement proper Fault Handlers.
- Reset DBGMCU registers to default.

11 Troubleshooting

Table 6 summarizes solutions to overcome some of the most frequent issues faced during debug setting and operation.

Table 6. Troubleshooting

Problem	Solution
Connection with target lost during debug of low-power system	Ensure debug in low-power in DBGMCU register is enabled. Ensure SWD pin not set in analog state. Refer to *Section 4.1: SWD/JTAG pinout* and to *Section 4.3: Low-power case*.
Fail to get printf via SWO	Refer to *Section 7.3: Printf via SWO/SWV*.
An unexpected power consumption is measured for a low-power application.	Check that low-power debug in DBGMCU register is OFF. Beware that this register is reset only with a POR (power-on reset). Refer to *Section 4.3*.
Fail to connect to a board with Normal/System Reset	Try ConnectUnderReset / Hardware Reset connection mode. This resets SWD connection in case it has been disabled by application. Refer to *Section 4.2*.
Fail to connect on board using ConnectUnderReset/Hardware using ST-LINK	Ensure NRST of ST-LINK is properly connected to MCU NRST (e.g. check SB12 for Nucleo).
Fail to see clock signal on MCO output	Ensure that the clock configured to MCO is in the supported range of the GPIO and that the OSPEED setting is correct. Refer to *Section 8.2*.
Impossible to evaluate a value or a variable, or impossible to set a breakpoint at a specific line in code	Compiler optimization is probably enabled. Remove it. Refer to *Chapter 3: Compiling for debug*.

Appendix A Managing DBGMCU registers

This appendix provides a tutorial for the different ways to Read/Write the DBGMCU registers with various tools and IDEs.

A.1 By software

HAL and LL provide functions to set/reset DBGMCU registers.

Refer to STM32Cube\Repository\STM32Cube_FW_[MCU] _[Version]\Drivers\STM32[MCU]xx_HAL_Driver\ STM32[MCU]xx_User_Manual.chm

Figure 70 and *Figure 71* show the positions of the DBGMCU registers iwithin the LL and HAL libraries.

Figure 70. DBMCU Register LL Library Functions

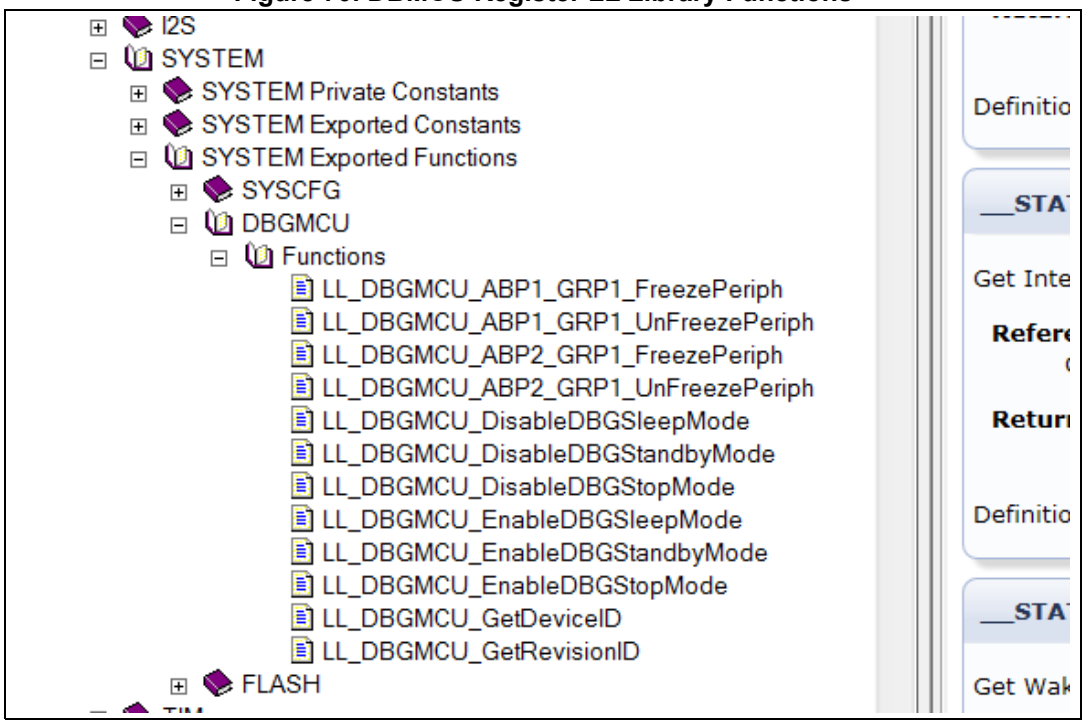
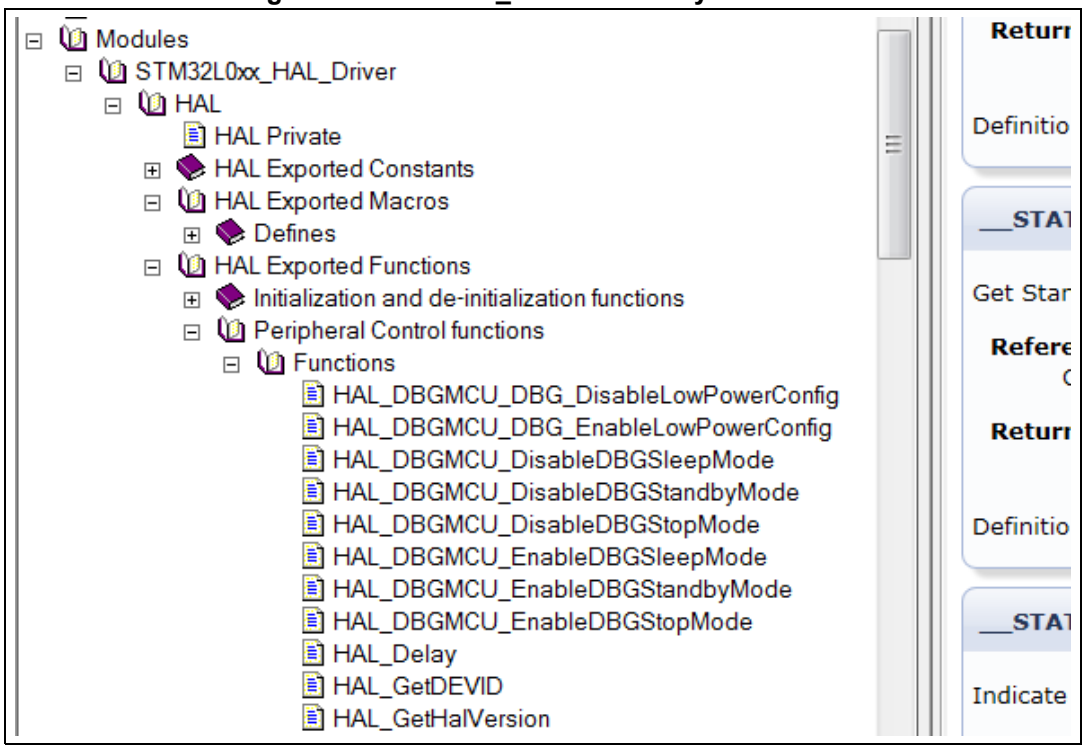


Figure 71. DBGMCU_CR HAL Library Functions



For M0 Cortex based families (L0/F0) DBGMCU module need to be clocked by setting bit 22 of register RCC_APB2ENR (refer to the corresponding reference manual) prior to be written.

```
RCC->APB2ENR |= RCC_APB2ENR_DBGMCUEN;
```

Some HAL macros are also available to Enable/Disable this clock.

```
__HAL_RCC_DBGMCU_CLK_ENABLE();

HAL_DBGMCU_EnableDBGStopMode();

HAL_DBGMCU_EnableDBGStandbyMode();

HAL_DBGMCU_EnableDBGSleepMode();

__HAL_RCC_DBGMCU_CLK_DISABLE();
```

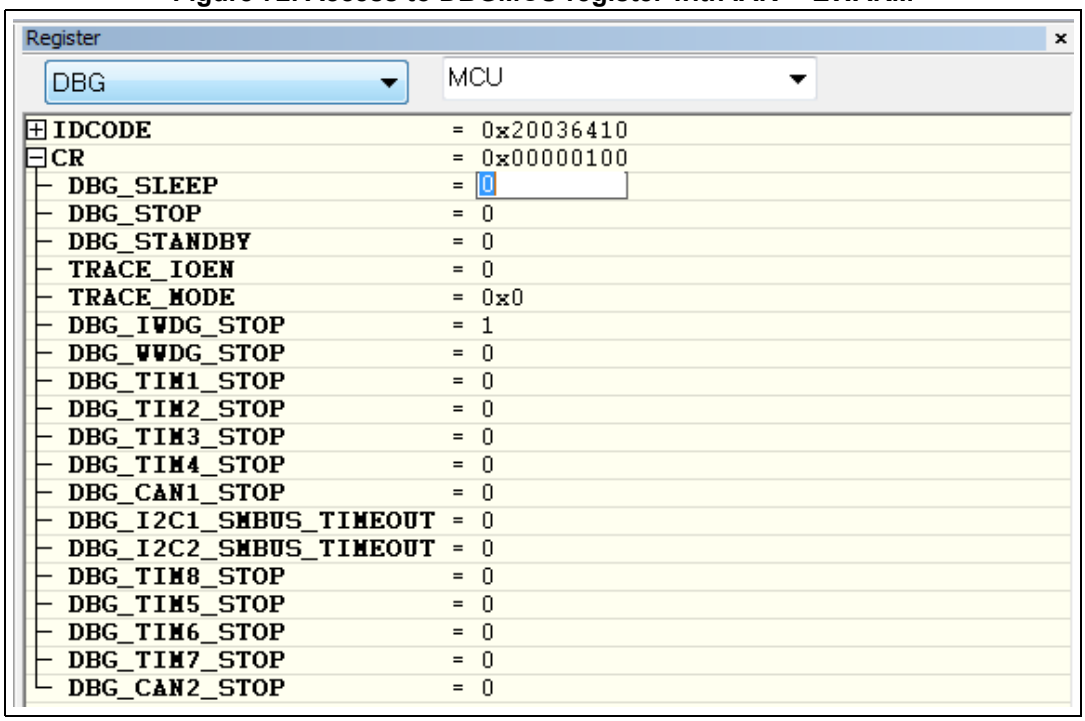
A.2 By debugger

In order to avoid debugging specific lines in the source code, there are several possibilities to set DBGMCU registers through debugger interfaces or scripts.

IAR™ EWARM

Read/Write of DBGMCU registers is possible through the register window as shown in *Figure 72*:

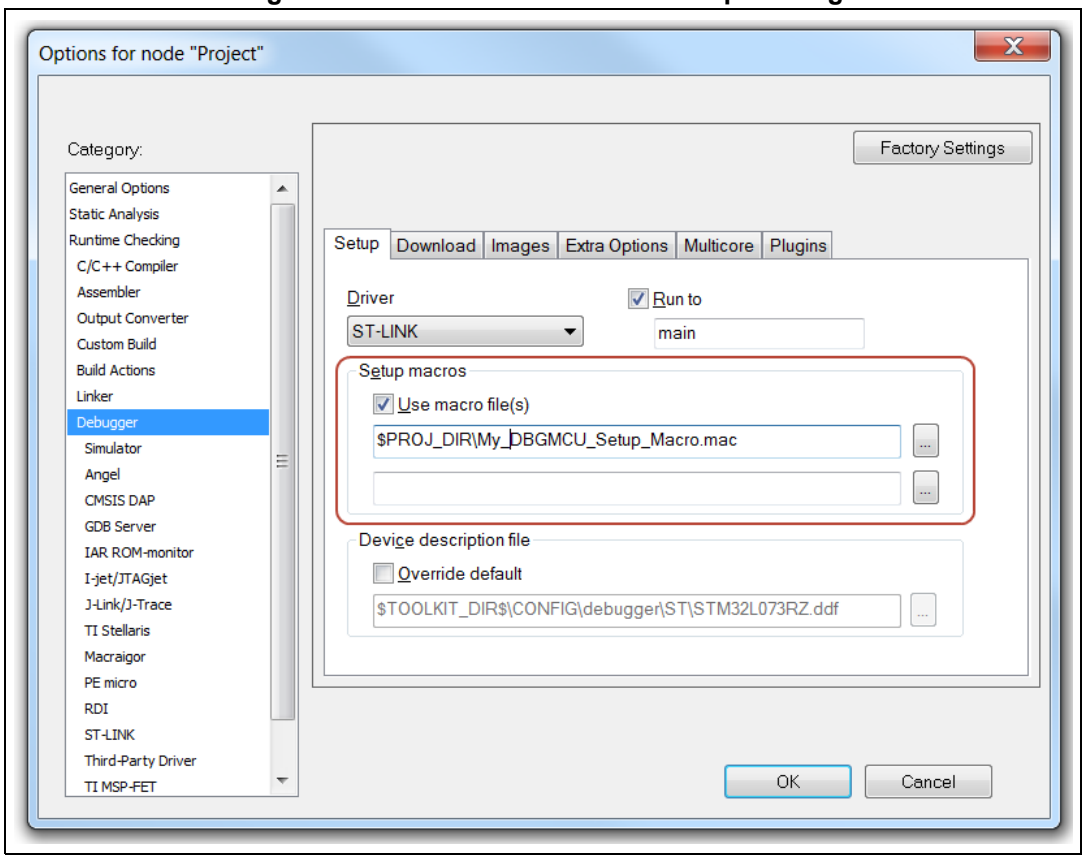
Figure 72. Access to DBGMCU register with IAR™ EWARM



In case a more permanent setup is required EWARM C-SPY® debugger macros enable to define `execUserSetup()`, which is executed at debugger start prior to program execution.

Figure 73 shows the ***Project Option Debugger*** -> ***Setup Pane***.

Figure 73. EWARM C-SPY® Macro script setting



A basic sample code of `execUserSetup()` function used to enable low-power debug on L0 is provided below:

```
execUserSetup(){/* Write a message to the debug log  */
__message "L0 DBGMCU Setup IAR Macro \n";

__writeMemory32 (0x00400000, 0x40021034, "Memory"); // Enable clock DBG
__writeMemory32 (0x00000007, 0x40015804, "Memory"); // Enable low-power
Debug in DBG_CR
__writeMemory32 (0x00000001, 0x40015808, "Memory"); // DBG_APB1_FZ Timer2
Stop Enable

}
```

For further information about feature offer by C-SPY® macros please refer to *C-SPY® Debugging Guide* available in IAR Help Menu and on www.iar.com.

Note: IAR™ EWARM enables Low-Power debug by default if connected with I-jet™ or cmis-dap compliant probes.

Keil® MDK-Arm µVision

At runtime, access to the DBGMCU register is possible through ***View*** -> ***System Viewer*** -> ***DBG***.

Figure 74. Accessing DBGMCU register in Keil® MDK-Arm µVision (1/2)

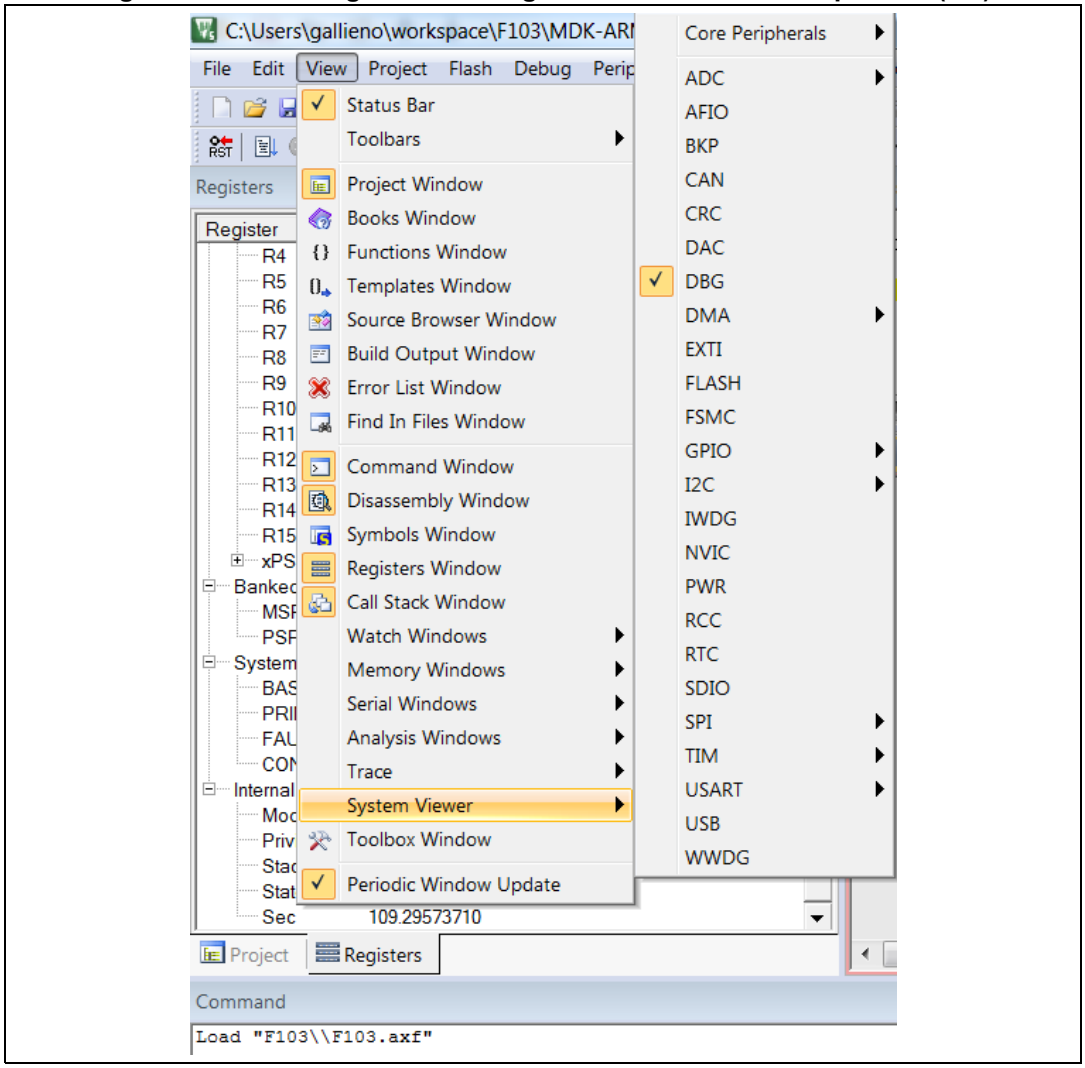
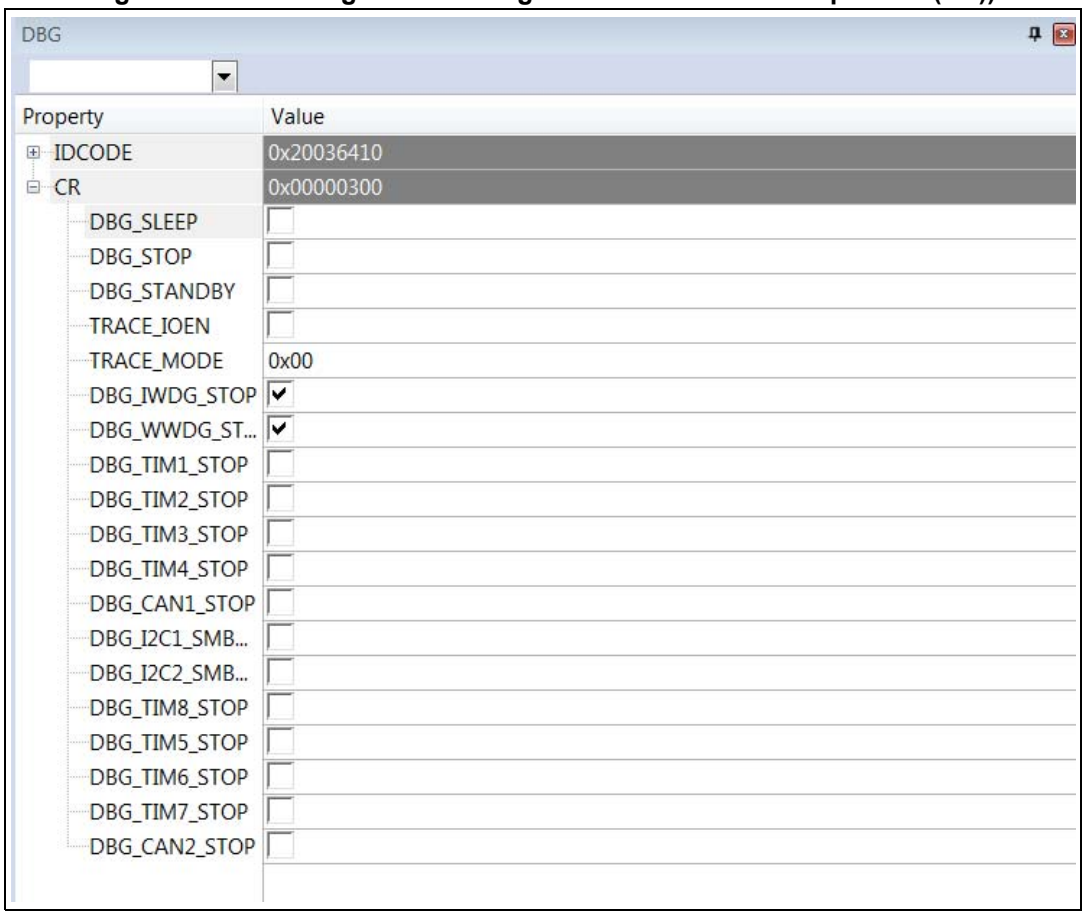


Figure 75. Accessing DBGMCU register in Keil® MDK-Arm µVision (2/2))



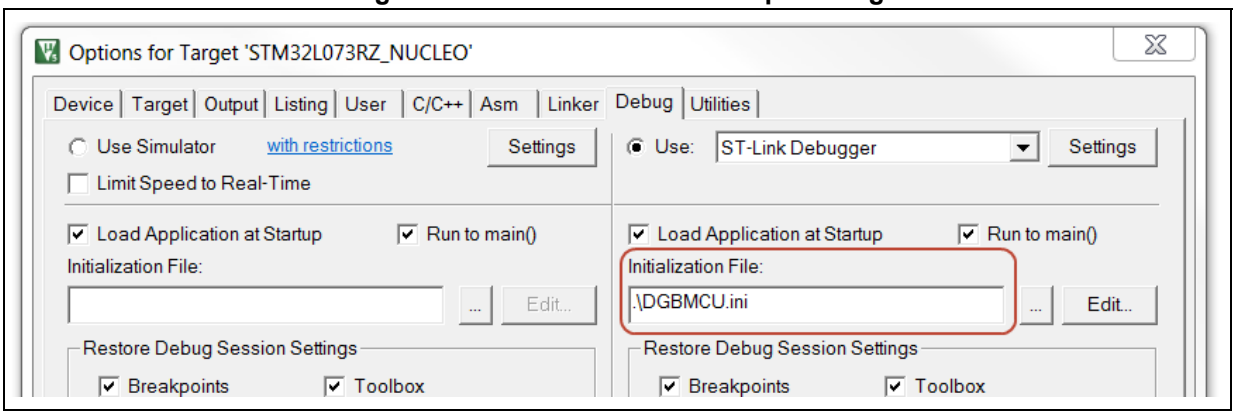
Each bit in the register can be set or reset independently.

For a permanent debug configuration, use Keil® MDK-Arm µVision initialization file capability.

Debugger script files are plain text files that contain debugger commands. These files are not created by the tools. The user must create them to suit his specific needs. Typically, they are used to configure the debugger or to setup or initialize something prior to running the program.

Figure 76 shows initialization script setting in ***Project option*** ->***Debug Pane***.

Figure 76. Keil® Initialization script setting



Sample code for Init file setting DBGMCU registers on M0 based MCU (Clock enabling)

```
FUNC void DBGMCUSetup (void) {

// DBGMCU configuration
_WDWORD(0x40021034, 0x00400000);  // Enable clock DBG
_WDWORD(0x40015804, 0x00000007);  // DBG_CR
_WDWORD(0x40015808, 0x00000001);  // DBG_APB1_FZ

}
DBGMCUSetup();
```

For further information regarding Keil® MDK-Arm µVision initialization script, refer to http://www.keil.com.

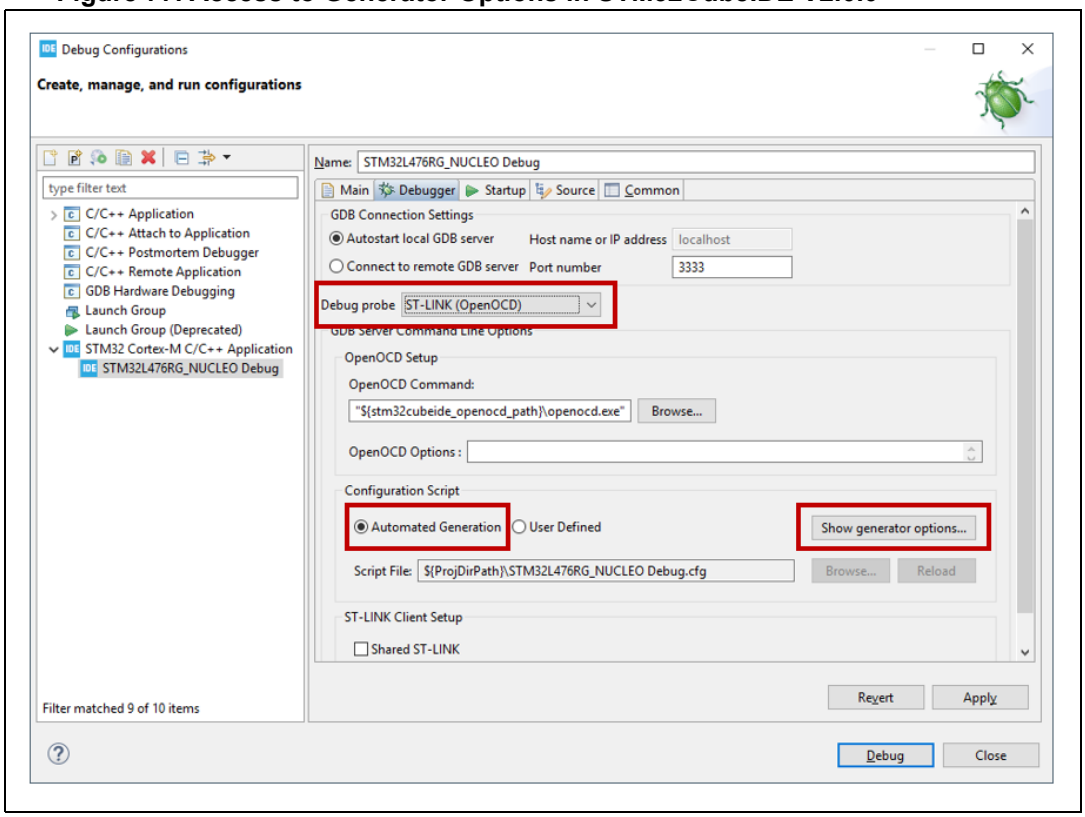
STM32CubeIDE

By default, STM32CubeIDEenable Low-Power debug. This default setting can be changed through Run->Debug Configurations->Debugger Pane.

- Setting up with OpenOCD server

By clicking on the Show generator options as presented in the figure below.

Figure 77. Access to Generator Options in STM32CubeIDE V2.0.0



DBGMCU options are available under Reset Mode in the Mode Setup group as shown in *Figure 78*.

Figure 78. Generator Options debug MCU in STM32CubeIDE

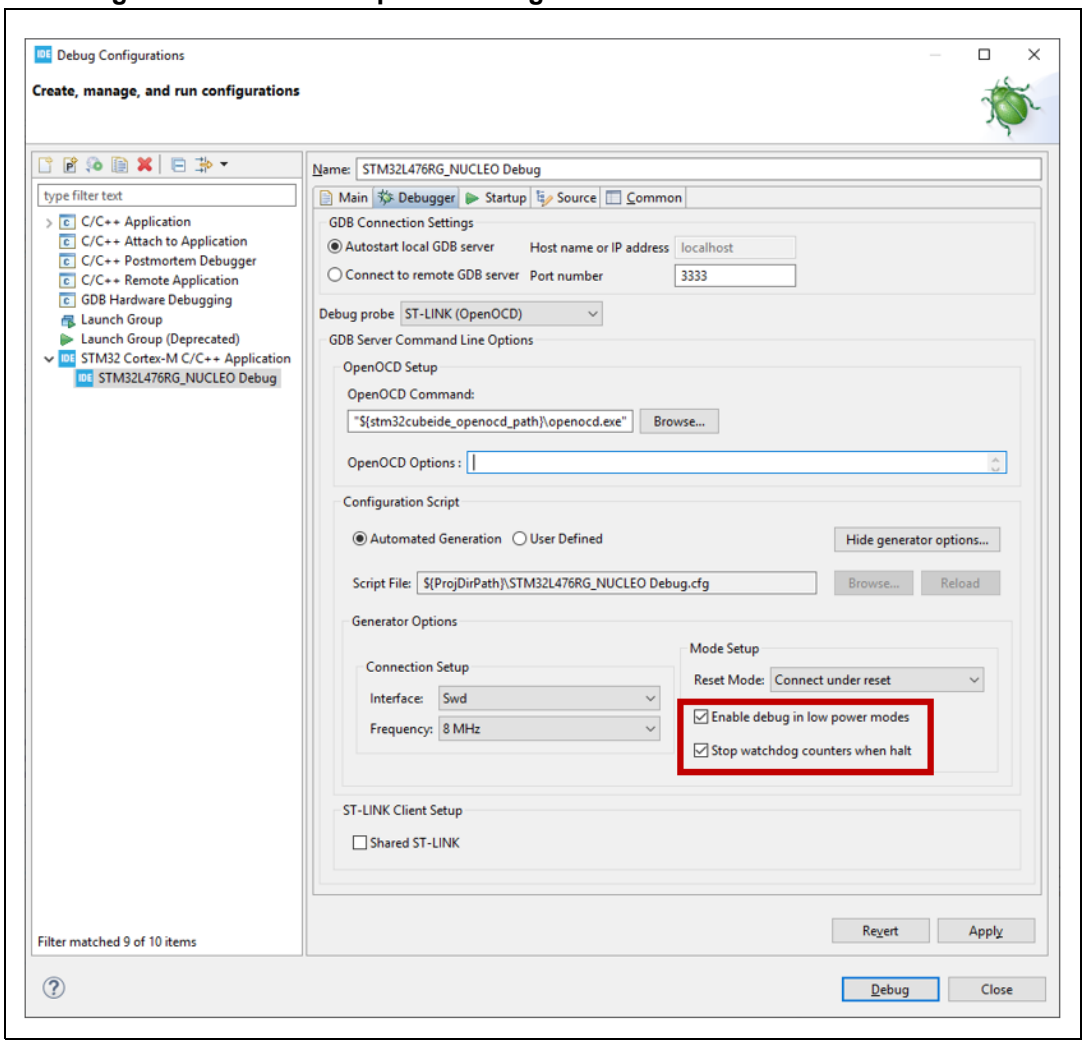
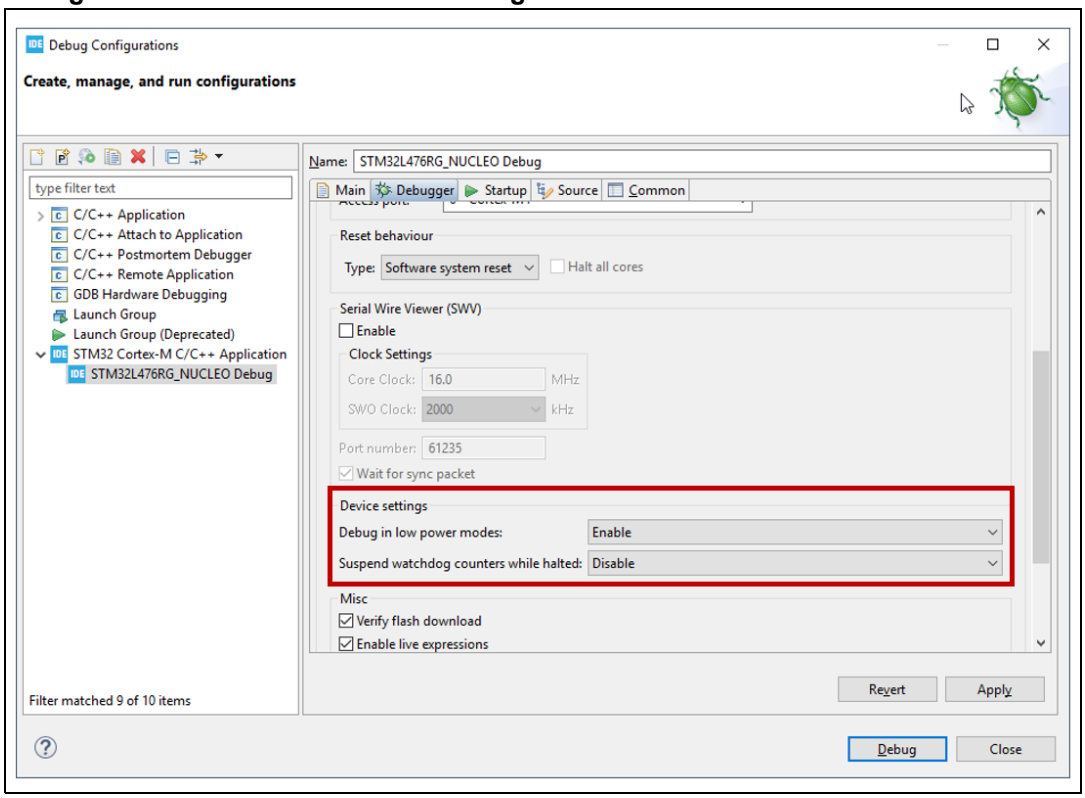
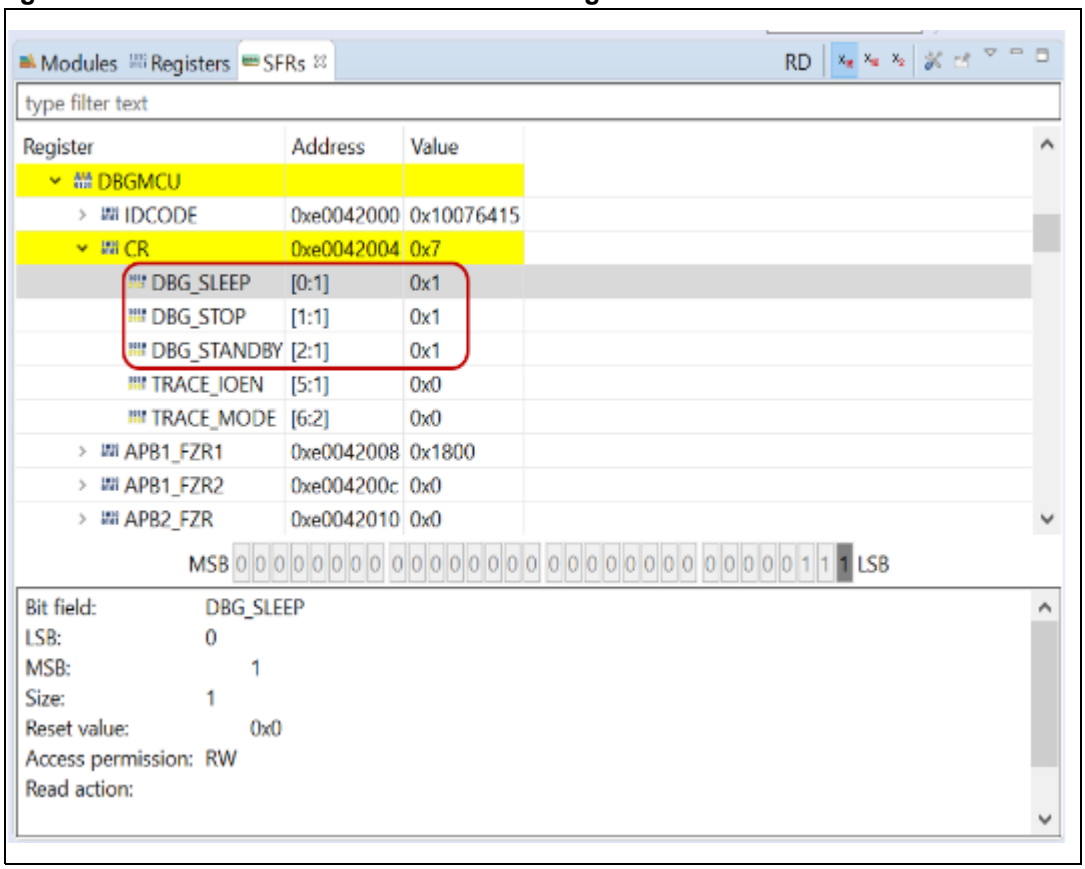


Figure 79. Access to DBGMCU settings with STM32CubeIDE V1.3.0



If needed, the DBGMCU value can be changed at run time through the I/O Registers window as shown in *Figure 80*.

Figure 80. Runtime R/W access to DBGMCU register with SSTM32CubeIDE



Note: All DBGMCU registers values are kept while reset. Pay attention to not let a debug or unwanted state when returning to normal execution. (refer to Chapter 9: Dual-Core microcontroller debugging on page 92).

Appendix B Use Nucleo “cuttable” ST-LINK as stand-alone VCP

As stated in *Section 7.2: Printf via UART on page 69*, it is required to have an adapter between MCU and PC to setup a proper serial connection.

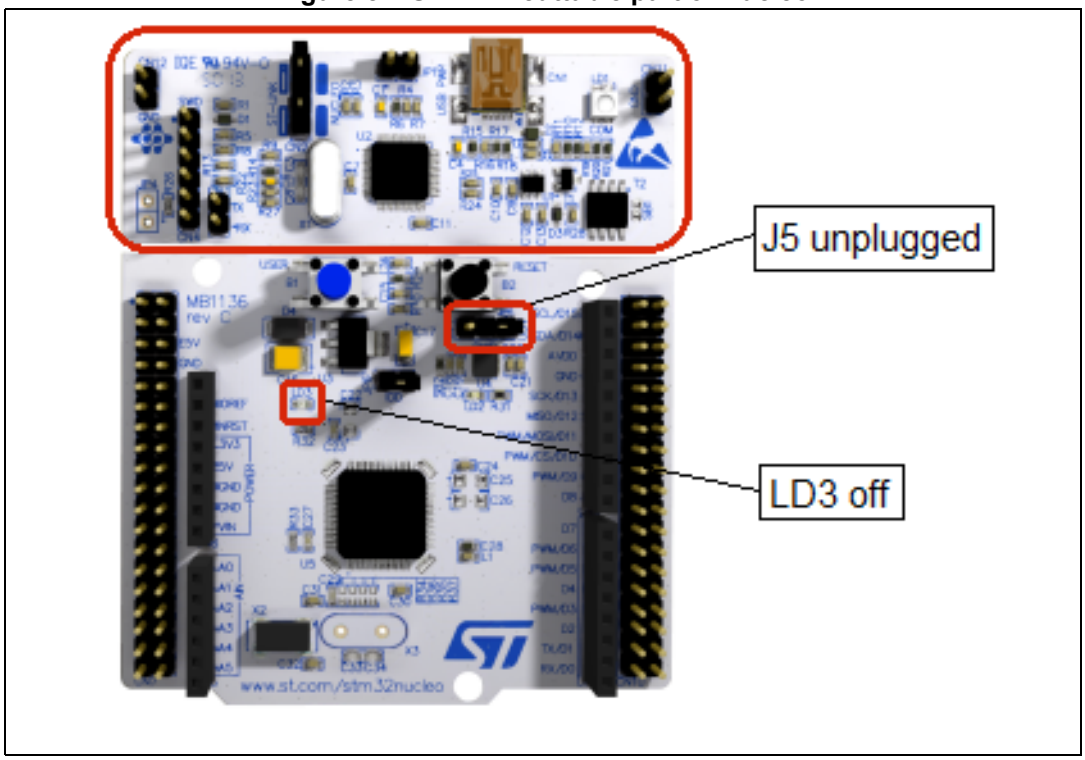
Design constraints may prevent to use the default UART connected to VCP, or may require another serial connection with the PC.

In such a case, it is simpler and cheaper to use another Nucleo board instead of getting the appropriate RS232 level shifter.

The "Cuttable PCB" capabilities of the Nucleo-64 and Nucleo-144 boards represent their capacity to disconnect on-board ST-LINK from STM32 application part.

The simple way to disconnect the ST-LINK part from the MCU application part is to power off the MCU by removing jumper J5. This is indicated by the fact that LED LD3 is off when a USB cable is connected. This configuration is presented in As show in *Figure 81*.

Figure 81. ST-LINK cuttable part of Nucleo



In this case the ST-LINK part can be used as a stand-alone module.

1. As debugger interface to program and debug an external application as documented in the user manual
   - STM32 Nucleo-144 board: section 6.3.4 of *Using ST-LINK/V2-1 to program and debug an external STM32 application* (UM1974)
   - STM32 Nucleo-64 board: *Using ST-LINK/V2-1 to program and debug an external STM32 application* (UM1724)
2. As an alternative and/or additional Virtual COM port

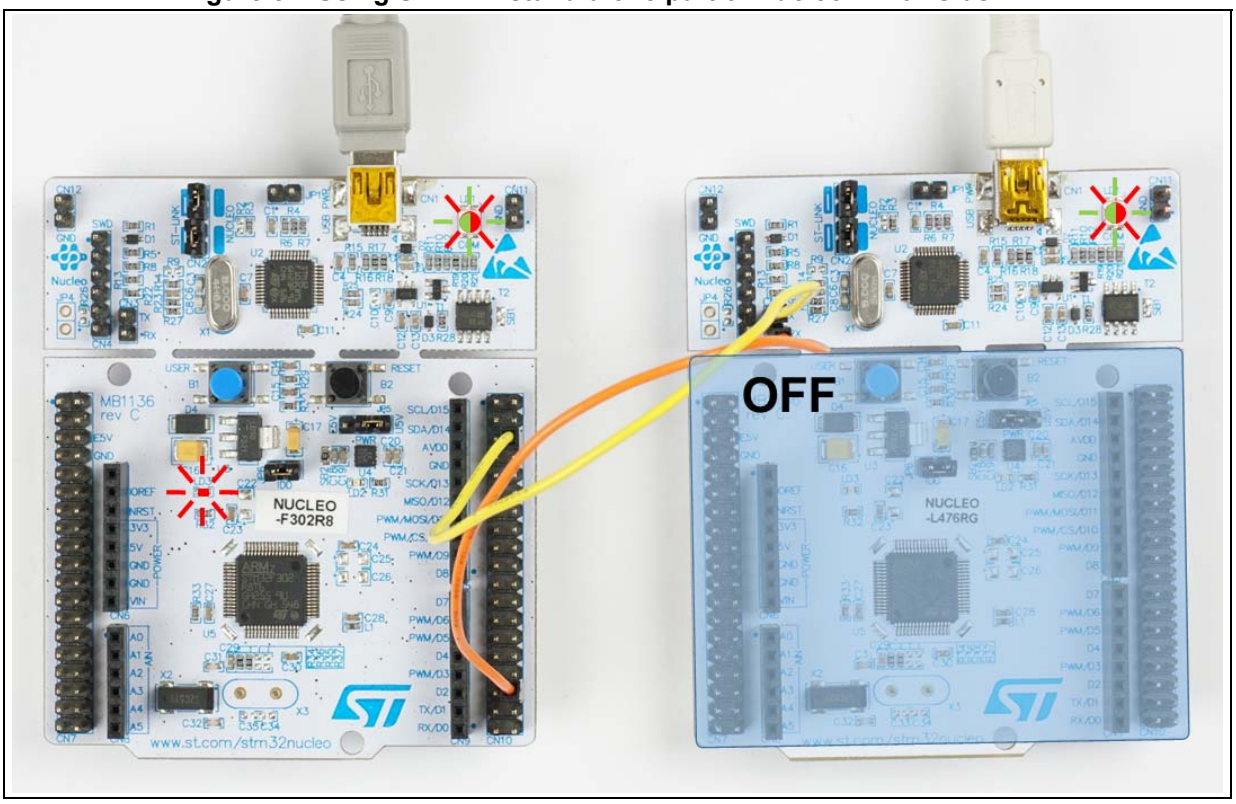
Any available UART of the STM32 application can be connected to the CN3 connector of the ST-LINK part.

Figure 82 illustrates a project using NUCLEO-F302R8 is using ST-LINK part of a NUCLEO-L476RG for connection of UART1 to the host.

UART1 RX PC5 is routed via Morpho Connector CN10 Pin 6 to CN3 TX of NUCLEO-L476RG ST-LINK.

UART1 TX PC4 is routed via Morpho Connector CN10 Pin 34 to CN3 RX of NUCLEO-L476RG ST-LINK.

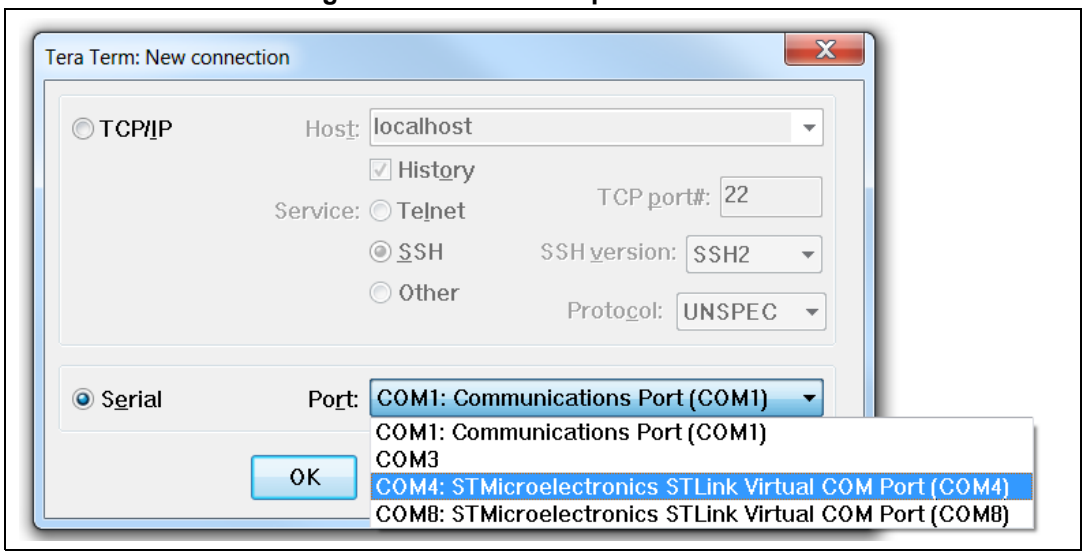
Figure 82. Using ST-LINK stand-alone part of Nucleo-L476RG as VCP



With this setup, on the PC side, two Virtual COM ports are available with potentially two different serial channels:

1. Nucleo-F302R8 UART2 (native default VCP) to COM4
2. Nucleo-F302R8 UART1 (VCP through Nucleo-L476RG) to COM8

Figure 83. Virtual COM port on PC side



Note: This usage implies to have several targets connected to a single host PC.

In order to properly identify the target and the VCP, refer to *Appendix C: Managing various targets on the same PC*.

Appendix C Managing various targets on the same PC

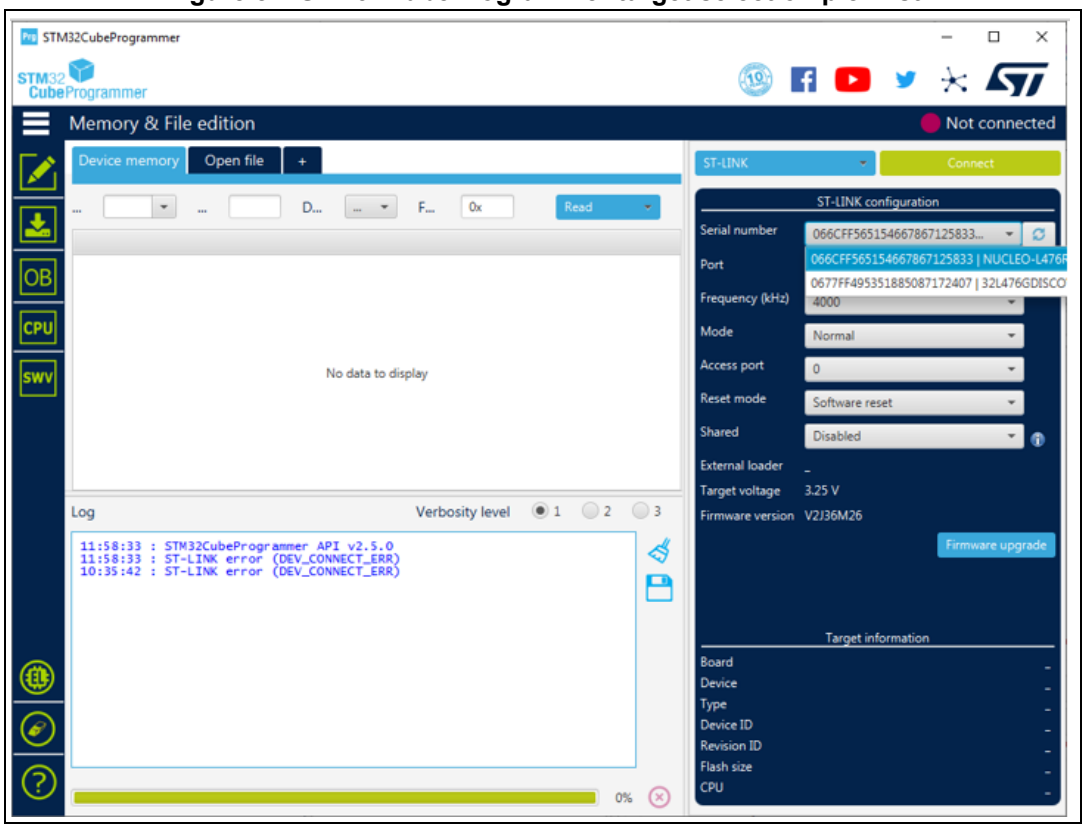
This appendix provides hints to identify and control the connection to a specific target among several ones using ST-LINK probe.

Each ST-LINK connection is identified by a serial number.

In order to correlate a serial number with a board, it is advised to use STM32CubeProgrammer.

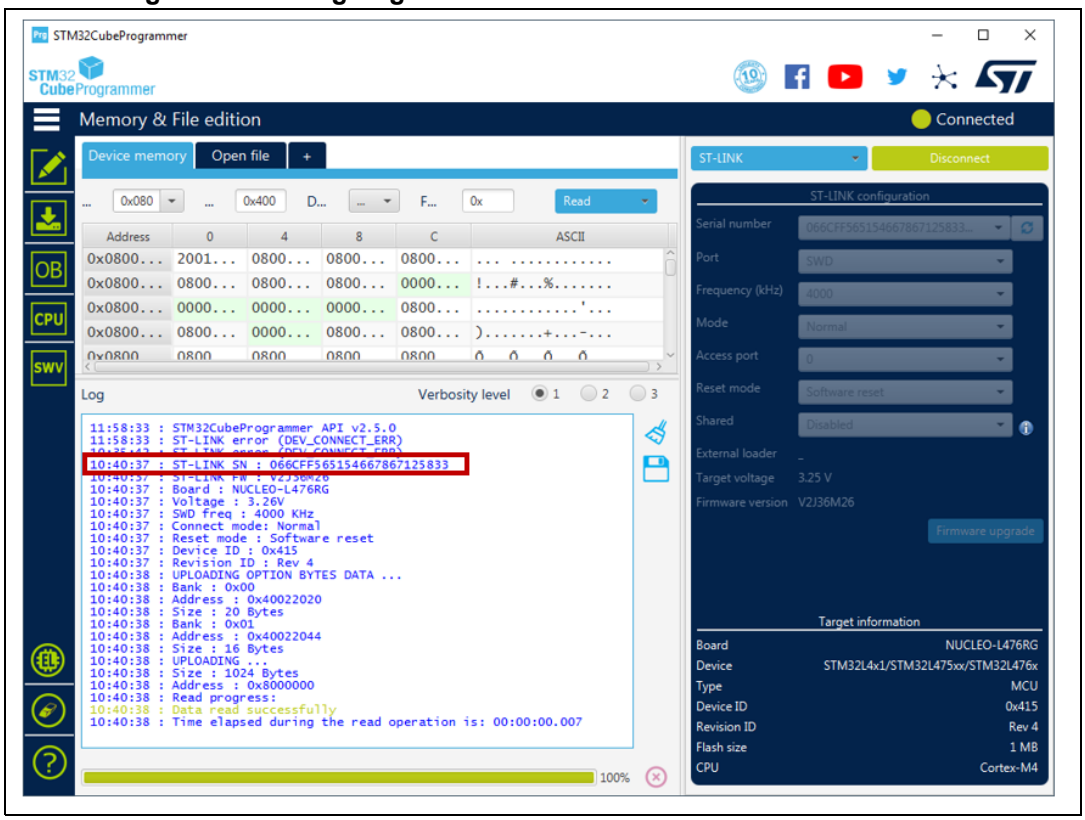
At the top of the screen, the serial number pick list contains all connected ST-LINK probes. By selecting one, access to the target is generated, making blinking of the related ST-LINK LED switch from red to green.

Figure 84. STM32CubeProgrammer target selection pick list



Once the target is identified, it is possible to copy the S/N from the console in the clip-board as shown in *Figure 85*.

Figure 85. Getting target ST-LINK S/N from the console



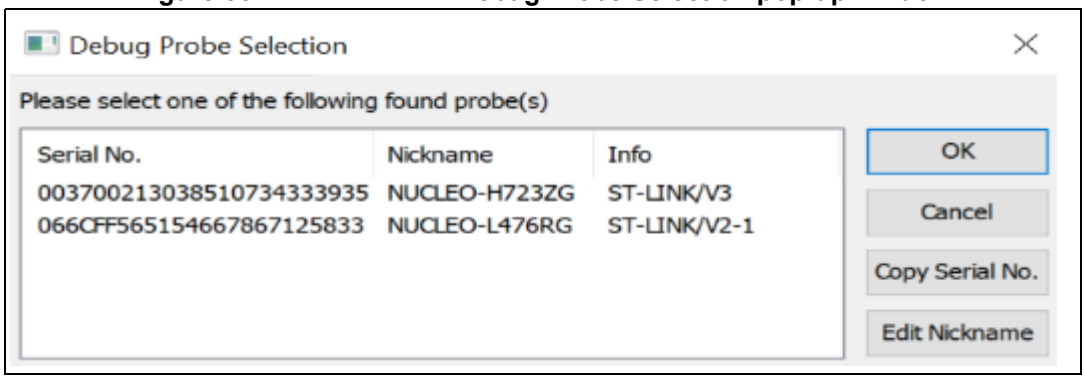
The next sections detail the selection of a specific target with each of the main IDEs considered in this application note.

IAR™ EWARM

The first time a debug session is launched while several targets are connected, a Debug Probe Selection window pops up.

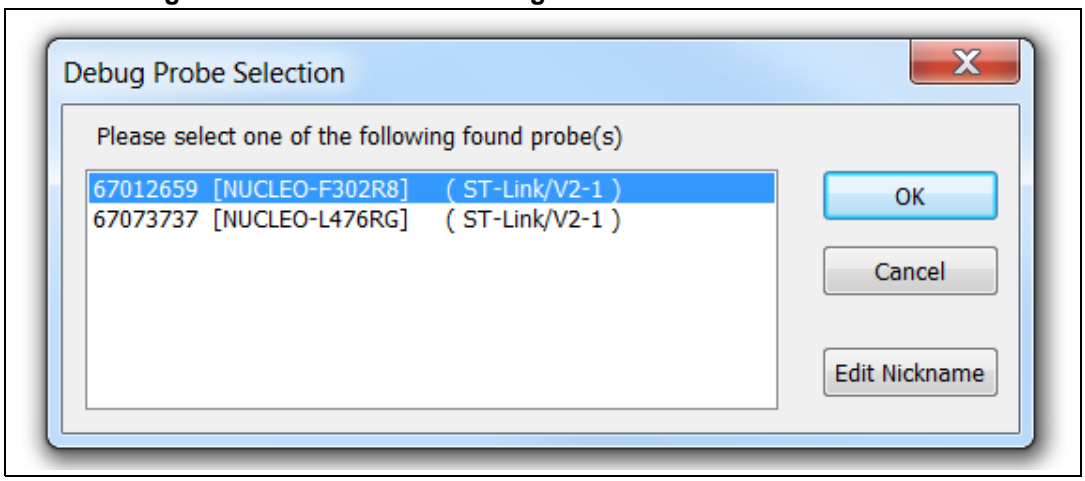
A list of connected targets is displayed, identified by the last four bytes of the ST-LINK S/N as illustrated in *Figure 86*.

Figure 86. IAR™ EWARM Debug Probe Selection pop-up window



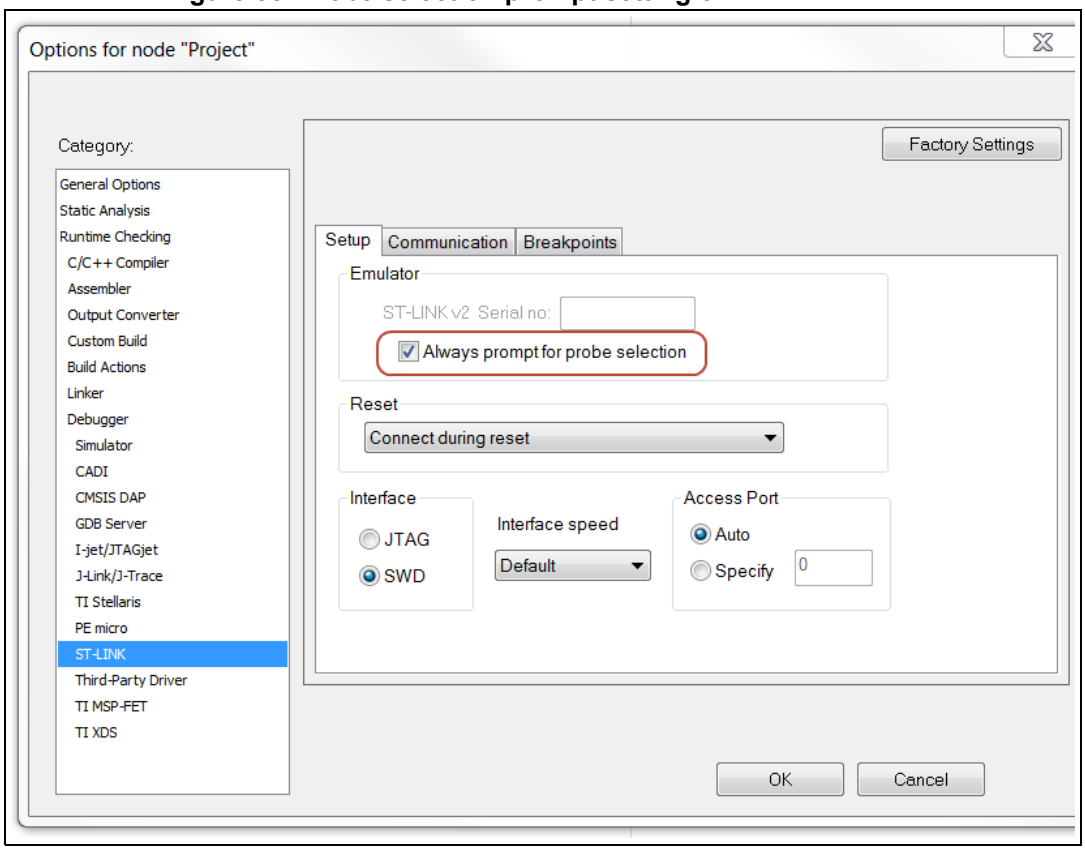
It is recommended to use the Edit Nickname feature to ease board identification in anticipation of further connection as shown in *Figure 87*.

Figure 87. IAR™ EWARM Debug Probe Selection with nickname



Important: The pop-up window is displayed only at first time. The selection made is then applied by default to further connections. Changing this initial selection requires that the "Debug Probe Selection" display is forced by setting the "Always prompt for probe selection" option in ***Option*** -> ***ST-LINK*** -> ***Setup*** as shown in *Figure 88*.

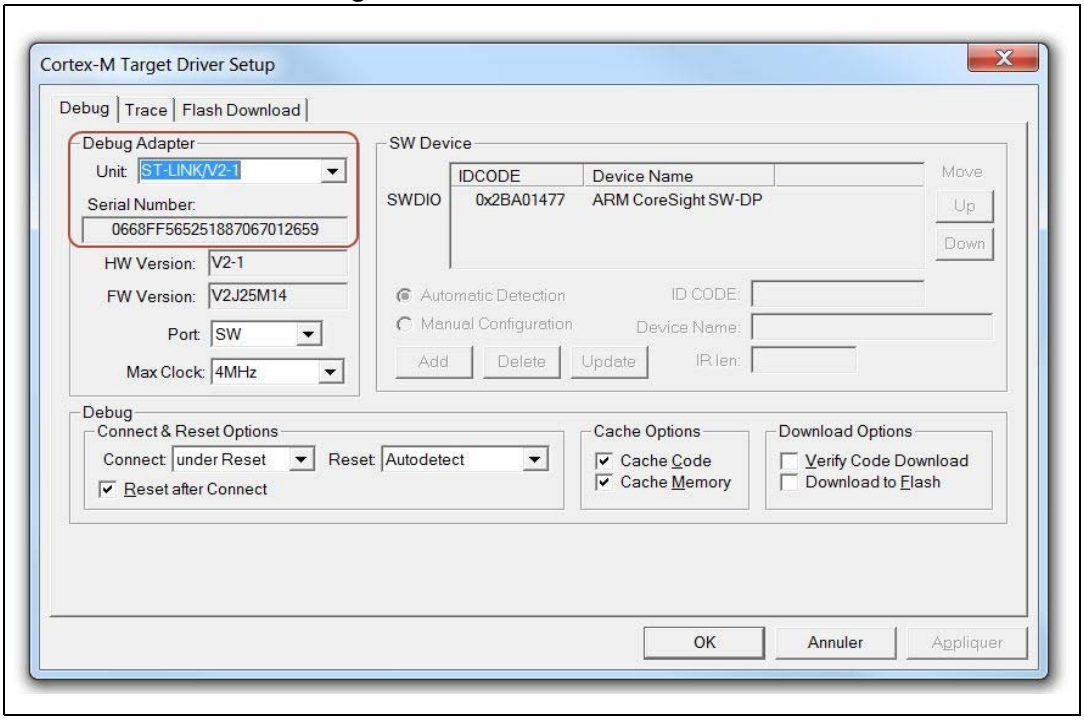
Figure 88. Probe selection prompt setting on IAR™ EWARM



Keil® MDK-Arm µVision

The list of connected targets is visible in ST-LINK debug pane (***Options*** -> ***Debug*** -> ***ST-LINK*** -> ***Settings*** -> ***Debug Pane***) as presented in *Figure 89*.

Figure 89. Keil® ST-LINK selection



In the Debug Adapter section, the pick list allows to select among all connected targets.

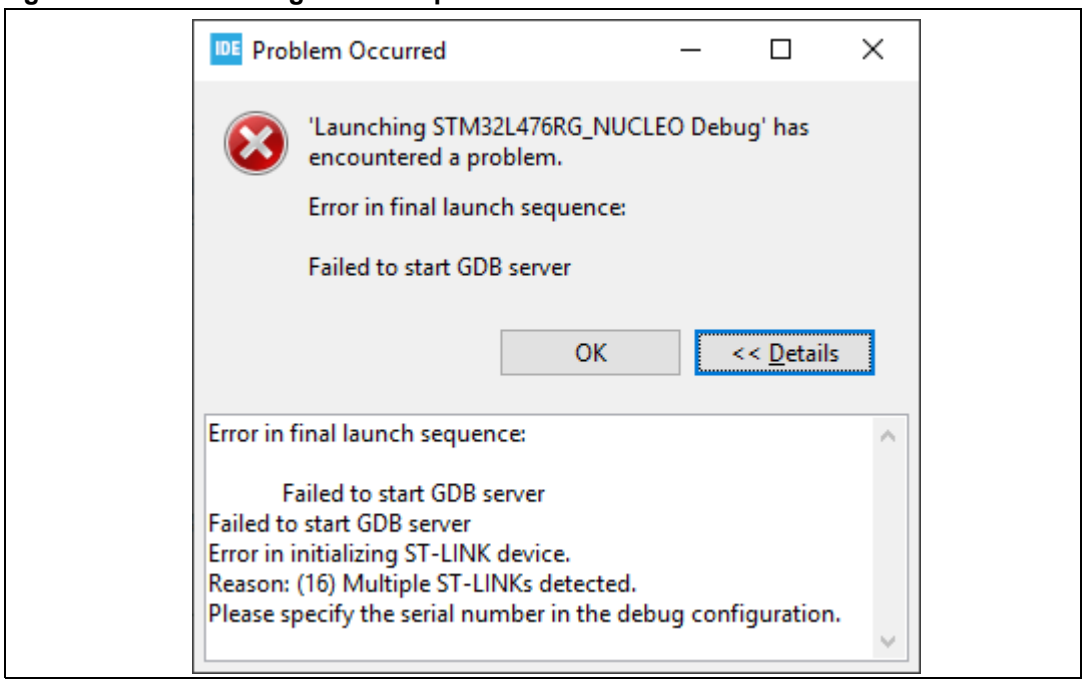
At selection it can be observed a brief activity of the ST-LINK LED of the related board and the Serial Number is displayed.

The selection is stored for the next connections.

STM32CubeIDE

If you try to launch a debug session with two ST-LINK connected, a pop-up message appear as shown in the figure below:

Figure 90. Error message for multiple ST-LINK detected in STM32CubeIDE



- Setting up with OpenOCD

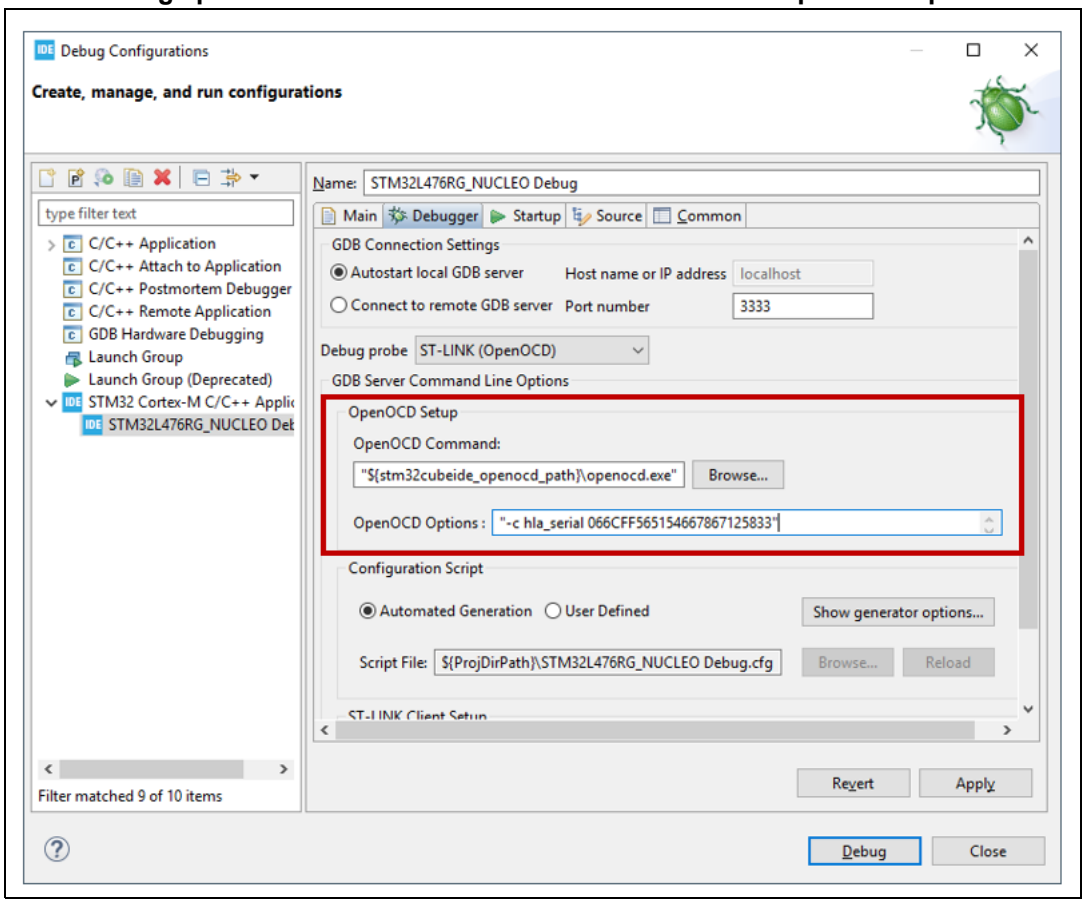
rIt is possible to force the connection to a specific target using the ST-LINK S/N.

In **Run -> Debug Configurations -> Debugger Pane**, add the following OpenOCD option:

```
-c hla_serial [ST-LINK S/N]
```

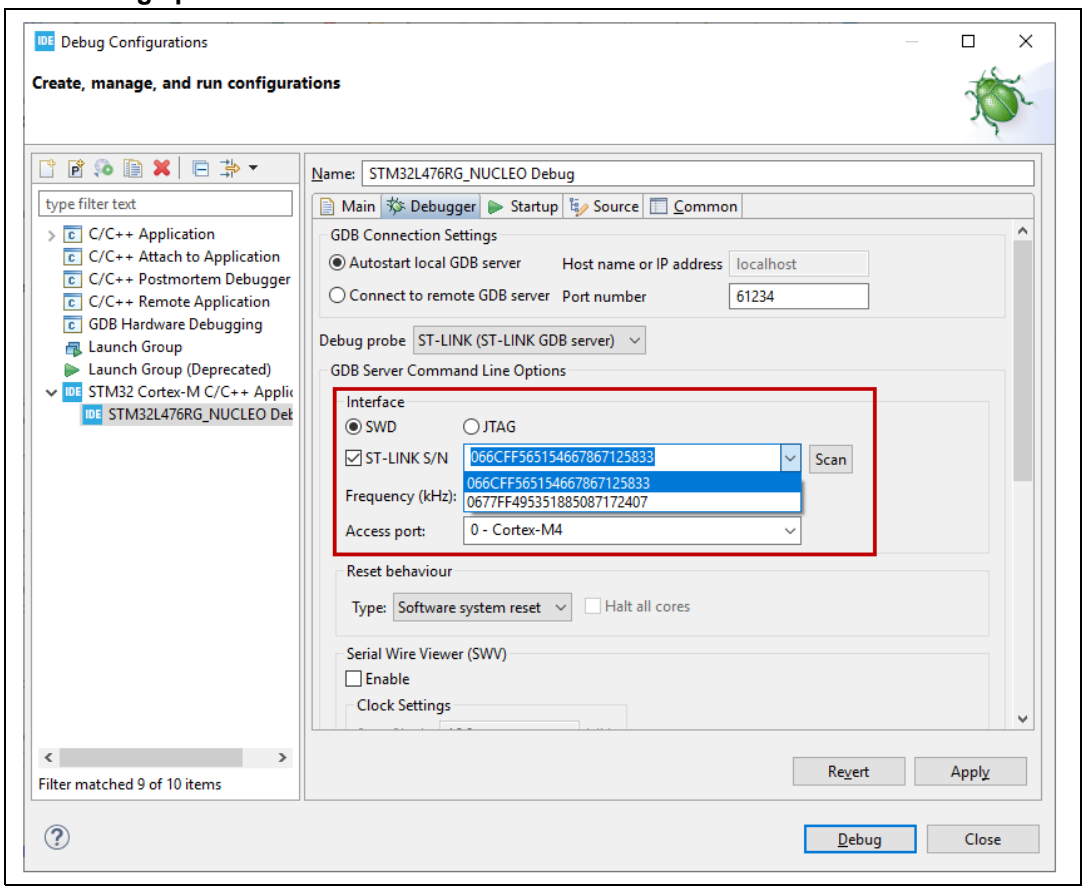
Figure below illustrates the setting of an OpenOCD option for forcing a connection.

Figure 91. Forcing specific ST-LINK S/N with STM32CubeIDE with OpenOCD option



- Setting up with ST-LINK GDB server

Figure 92. Forcing specific ST-LINK S/N with STM32CubeIDE with ST-LINK GDB server



Appendix D Cortex®-M debug capabilities reminder

STM32 families debug capabilities depend on their Cortex®-M type.

Table 7. STM32 Series vs. debug capabilties

STM32 Series	Cortex type	SWD	JTAG	ETM	SWO	Hardware breakpoints	Core Reset	MCO(1)
L0/F0	M0/0+	Yes	No	No	No	4	No	1
F1/L1/F2	M3	Yes	Yes	Yes(2)	Yes	6	Yes	1
F3/F4/L4	M4	Yes	Yes	Yes(2)	Yes	6	Yes	2(2)
F7/H7	M7	Yes	Yes	Yes(2)	Yes	8	Yes	2(2)

1. Microcontroller Clock Output (refer to *Section 8.2: Microcontroller clock output (MCO) on page 87*)
2. Depends on package size. Check availability in the Pin Allocation Table in the related datasheet.

For more details, refer to the related Cortex® Arm® documentation.

D.1 Application notes index

Table 8. STM32 Series vs. debug capabilities

AN references	Subject
AN5361	Dual core microcontrollers debugging
AN5286	Dual core microcontrollers debugging
AN5421	Trustzone
AN5347	Trustzone

Note: Microcontroller Clock Output (refer to Section 8.2: Microcontroller clock output (MCO).

Revision history

Table 9. Document revision history

Date	Revision	Changes
16-Jun-2017	1	Initial release.
29-Jun-2017	2	Added *Table 1: Applicable products*.
26-Jan-2021	3	Updated: – *Section 1.2: Software versions* – *Section 2.1.1: Hardware kits* – *Figure 5: Discovery board example* Added: – *Section 1.1: General information* – *Section 2.4.2: Wiki platform* – *Section 2.4.3: Github* – *Figure 2: Development tools overview* – *Figure 4: STM32 Nucleo-144 structure* – *Figure 15: STM32Cube monitor* – *Figure 14: STM32Cube programmer*

IMPORTANT NOTICE – PLEASE READ CAREFULLY

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

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

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

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

ST and the ST logo are trademarks of ST. All other product or service names are the property of their respective owners.

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

© 2021 STMicroelectronics – All rights reserved