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

List of tables

List of figures

1 Foreword

1.1 Software versions

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

IAR™ EWARM:	V8.10.1
Keil® MDK-ARM µVision:	V5.22
SW4STM32:	V2.0.0
ST-LINK utility:	V4.0.0

1.2 Acronyms

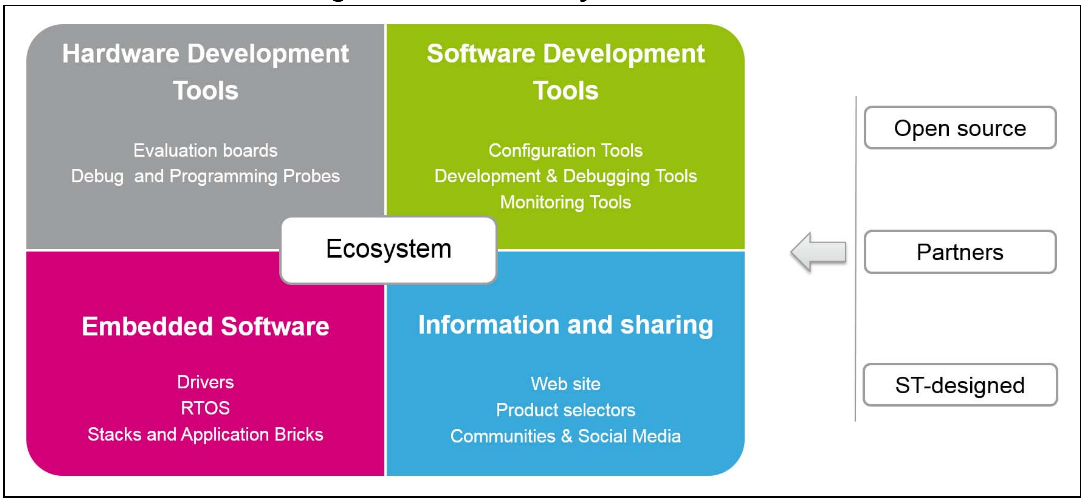
AN:	Application note
CMSIS:	Cortex microcontroller software interface standard
HAL:	Hardware abstraction layer (software library)
IDE:	Integrated development environment
JTAG:	Joint Test Action Group
MCO:	Microcontroller clock output
MCU:	Microcontroller unit
NVIC:	Nested vector interrupt controller
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

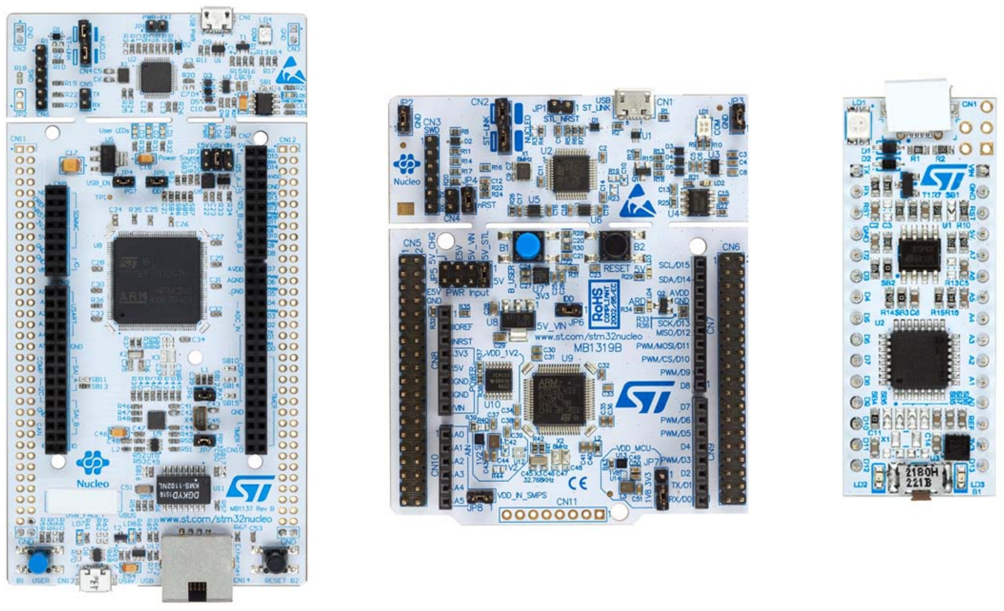
This section lists the hardware kits provided by STMicroelectronics for STM32-based development:

- Nucleo boards
- Discovery kits
- Evaluation boards (EVAL)

Nucleo boards

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

Figure 2. 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-64 boards include Arduino™ Uno rev3 and ST morpho connectors.

Nucleo-32 boards include Arduino™ Nano connectors.

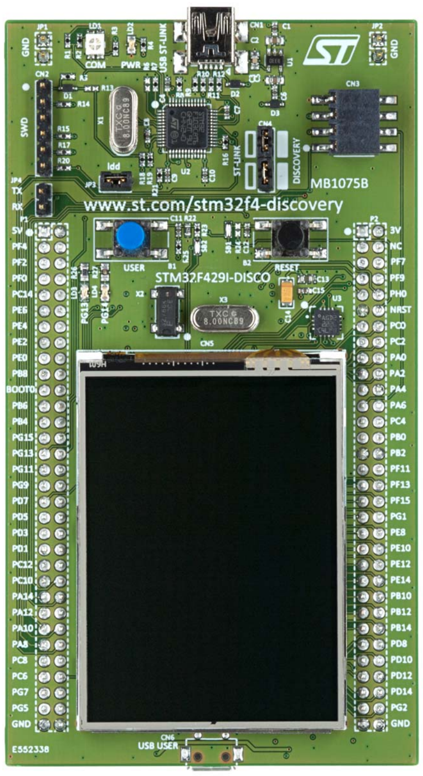
All STM32 Nucleo boards integrate an ST-LINK debugger/programmer, so there is no need for a separate probe.

A complete description of the embedded ST-LINK features is provided in *Section 2.1.2: ST-LINK probe on page 13*. 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 3. Discovery board example



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

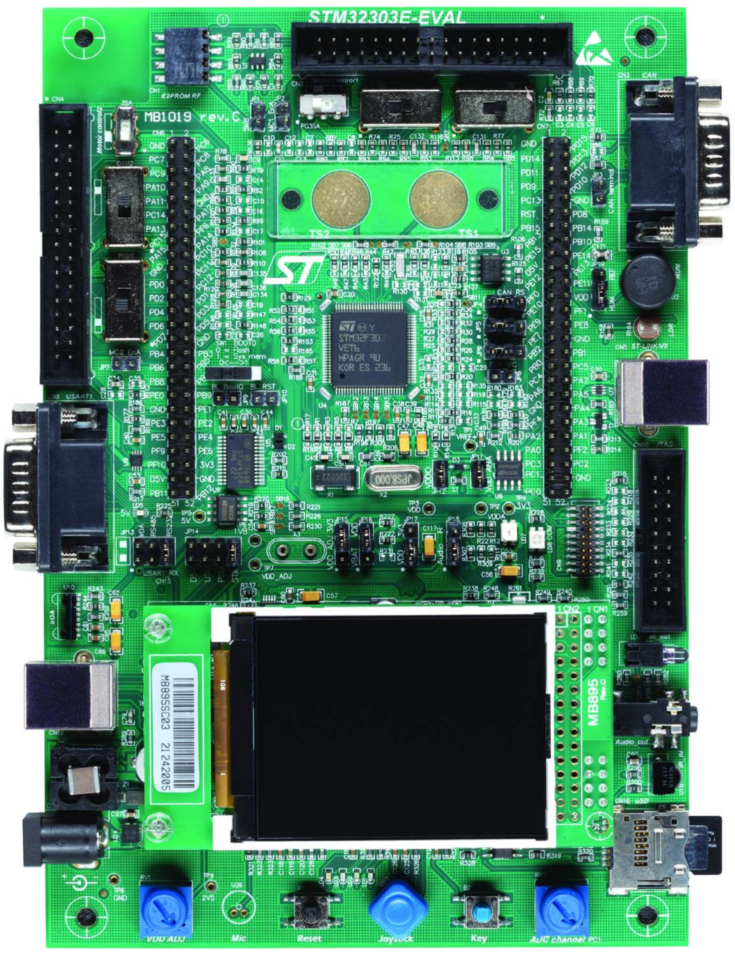
With the integrated debugger/programmer the Discovery kits are ideal for prototyping.

A complete description of the embedded ST-LINK features is provided in *Section 2.1.2: ST-LINK probe on page 13*. Additional information and access to Discovery kits complete documentation sets are available at *www.st.com*.

EVAL boards

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

Figure 4. 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 15*.

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

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)

Figure 5 shows the first ST-LINK version on the left and the ST-LINK/V2 and ST-LINK/V2-ISOL stand-alone probes on the right.

Figure 5. ST-LINK, ST-LINK/V2, and ST-LINK/V2-ISOL stand-alone probes

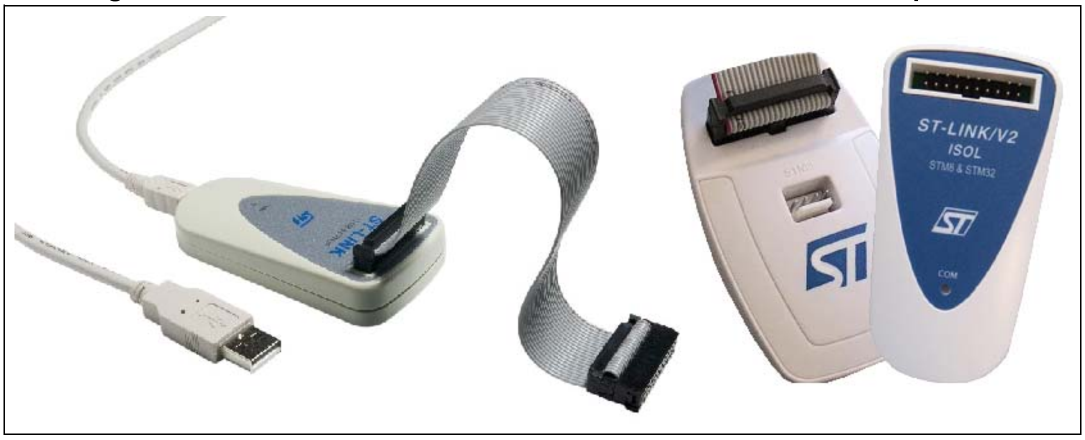
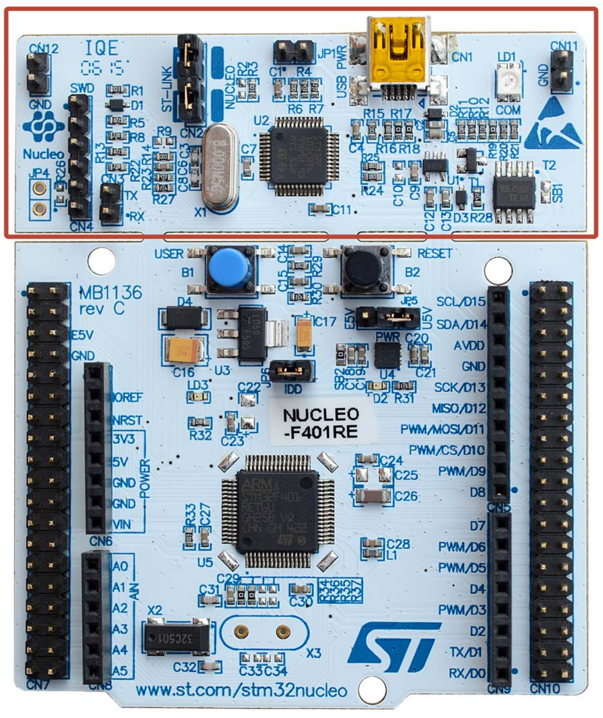


Figure 6 shows an example of an embedded ST-LINK/V2 as part of a Nucleo board.

Figure 6. On-board ST-LINK/V2 on Nucleo



Both stand-alone and embedded versions share the same ST-LINK/V2 basic features:

- 5 V power supplied by a USB connector
- USB 2.0 full-speed-compatible interface
- USB standard 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.

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-LINK004	STM32 ST-LINK utility (refer to *Section 2.2.4: ST-LINK utility on page 19*)
STSW-LINK007	ST-LINK, ST-LINK/V2, ST-LINK/V2-1 firmware upgrade
STSW-LINK009	ST-LINK, ST-LINK/V2, ST-LINK/V2-1 USB driver signed for Windows® 7, Windows® 8, Windows® 10

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

STSW-LINK009 is included in most IDE installation packages (IAR™, Keil®, SW4STM32) 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 ST-LINK utility software (refer to *Section 2.2.4: ST-LINK utility on page 19*).

2.1.3 Alternative debugger probes

J-LINK (Segger), I-Jet™ (IAR™), 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 to use:

- 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*.

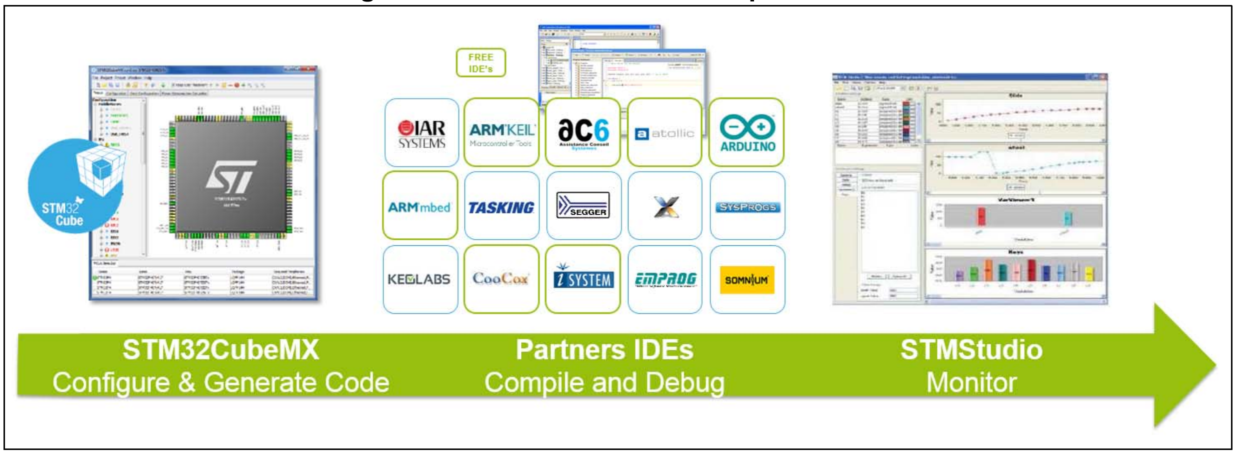
2.2 Software development tools

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 7*.

Figure 7. STM32 software development tools



2.2.1 STM32CubeMX

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.

1. 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)
2. 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).
3. 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
- Microcontroller graphical 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™, 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 Partner IDEs

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

1. IAR™ EWARM
2. Keil® MDK-ARM µVISION
3. SW4STM32

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™ 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

SW4STM32

The System Workbench for STM32 toolchain, called SW4STM32, is a free multi-OS software development environment based on Eclipse, which supports the full range of STM32 microcontrollers and associated boards.

The SW4STM32 toolchain may be obtained from the website www.openstm32.org, which includes forums, blogs, and trainings for technical support. Once registered to this site, users get installation instructions at the Documentation > System Workbench page to proceed with the download of the free toolchain.

The SW4STM32 toolchain and its collaborative website have been built in collaboration with AC6, a service company providing training and consultancy on embedded systems.

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: www.ac6.fr.

Key Features

- Comprehensive support for STM32 microcontrollers, STM32 Nucleo boards, Discovery kits and EVAL boards, as well as STM32 firmware (Standard Peripheral library or STM32Cube HAL)
- GCC C/C++ compiler
- GDB-based debugger
- Eclipse IDE with team-work management
- ST-LINK support
- No code size limit
- Multiple OS support: Windows® (32 and 64 bits), Linux (64 bits) and MacOS™

In case of installation or update through the ***Eclipse Help*** -> ***Install New software...*** menu, the following dependences for SW4STM32 V2.0.0 are required:

- Eclipse Mars2 or Neon
- CDT 8.8.1
- Java 1.8

All above dependences are properly managed in case of All-in-one installer package or through ***Help*** -> ***Check For Updates*** (recommended).

2.2.3 STMStudio

STMicroelectronics STMStudio helps debug and diagnose STM32 applications at run time by reading and displaying their variables in real-time.

Running on a PC, STM Studio interfaces with STM32 MCUs via the standard ST-LINK probe.

STMStudio is a non-intrusive tool, preserving the real-time behavior of applications.

STMStudio perfectly complements traditional debugging tools to fine tune applications. It is well suited for debugging applications which cannot be stopped, such as motor control applications.

Different graphic views are available to match the needs of debugging and diagnosis or to demonstrate application behavior.

Key Features

- Runs on PCs with Microsoft® Windows® OS (XP, Vista, 7, 8, or 10)
- Connects to any STM32 via ST-LINK (JTAG or SWD protocols)
- Reads on-the-fly (non intrusive) variables from RAM while application is running
- Parses DWARF debugging information in the ELF application executable file
- Two types of viewer:
  - Variable viewer: real-time waveforms, oscilloscope-like graphs
  - TouchPoint viewer: association of two variables, one on the X axis, one on the Y axis
- Possibility to log data into a file, and replay later (exhaustive record display, not real-time)

More information about the way to use STMStudio are available in STMicroelectronics user manual *Getting started with STMStudio* (UM1025).

2.2.4 ST-LINK utility

STM32 ST-LINK utility (STSW-LINK004) is a full-featured software interface for programming STM32 microcontrollers.

It provides an easy-to-use and efficient environment for reading, writing and verifying a memory device.

The tool offers a wide range of features to program STM32 internal memories (Flash, RAM, OTP and others), external memories, to verify the programming content (checksum, verify during and after programming, compare with file) and to automate STM32 programming.

STM32 ST-LINK utility is delivered with a graphical user interface (GUI) and with a command line interface (CLI).

Key Features

- Free software
- Supports Motorola S19, Intel HEX and binary formats
- Load, Edit and Save executable and data files generated by the Assembler/Linker or C compilers
- Erase, Program, View and Verify device Flash memory contents
- Program, Erase and Verify external memories with examples of external flash loaders, for users to develop loaders for specific external memories
- Automate STM32 programming (Erase, Verify, Programming, Configuring option bytes, calculate checksum)
- Programming One Time Programmable memory
- Supports Programming and Configuring Option bytes
- Offers a command line interface
- Compare file with target memory
- Supports memory and core status view in Live-update mode
- ST-LINK/V2 firmware upgrade

In debug context ST-LINK is useful:

- To check and update ST-LINK/V2 firmware in case of connection issue
- Recover connection to a board in case of stuck in permanent Low-power or secure state.
- Usage of SWV for printf debugging (Refer to *Chapter 7: Printf debugging on page 58*)

Refer to STMicroelectronics user manual *STM32 ST-LINK utility software description* (UM0892).

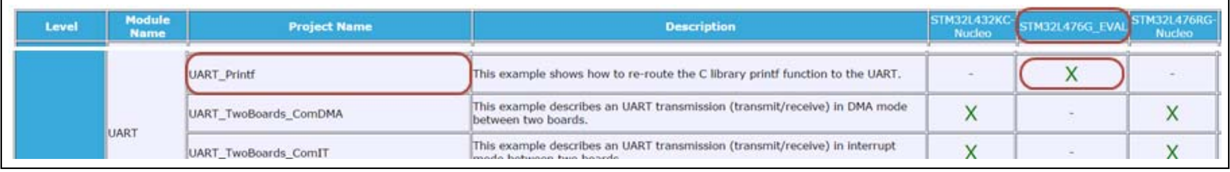
2.3 Embedded software

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 8*).

Figure 8. STM32CubeProjectList screenshot

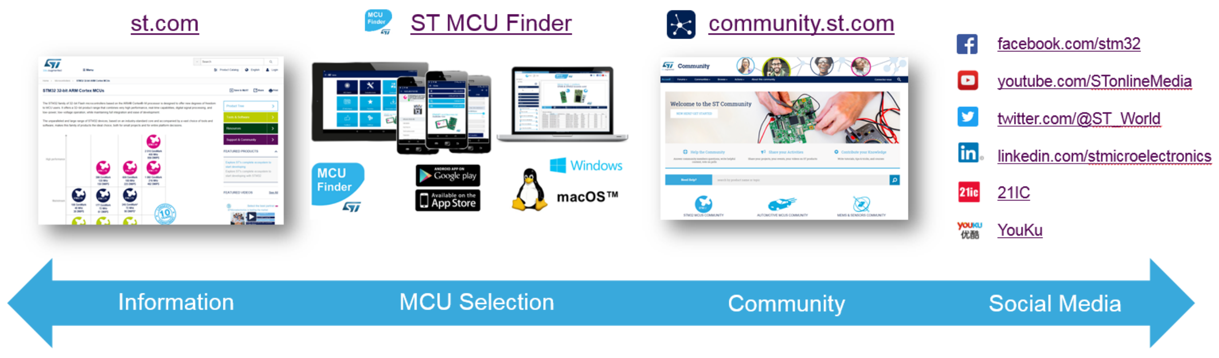


Level	Module Name	Project Name	Description	STM32L432KC-Nucleo	STM32L476G_EVAL	STM32L476RG-Nucleo
	UART	UART_Printf	This example shows how to re-route the C library printf function to the UART.	-	X	-
		UART_TwoBoards_ComDMA	This example describes an UART transmission (transmit/receive) in DMA mode between two boards.	X	-	X
		UART_TwoBoards_ComIT	This example describes an UART transmission (transmit/receive) in interrupt	X	-	X

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 9. 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

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 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.

2.4.3 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:

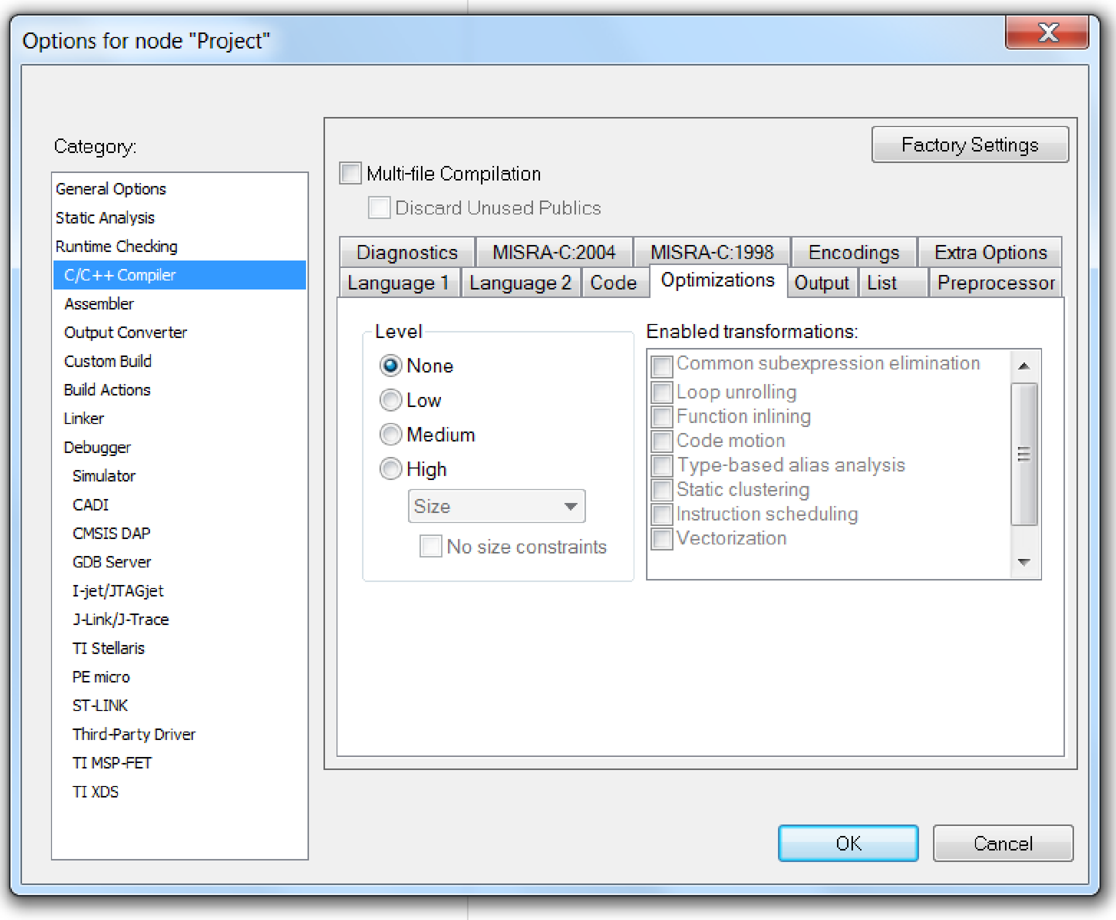
- Problem to set or reach a breakpoint. Some lines are not accessible.
- Impossibility to evaluate a variable (watch feature).
- Inconsistency while stepping (what I get, is not what I see).

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 10. IAR™ EWARM Optimization option



Multi-file Compilation

Discard Unused Publics

Level

[✓ None]

Low

Medium

High

Size

No size constraints

Enabled transformations:

Common subexpression elimination

Loop unrolling

Function inlining

Code motion

Type-based alias analysis

Static clustering

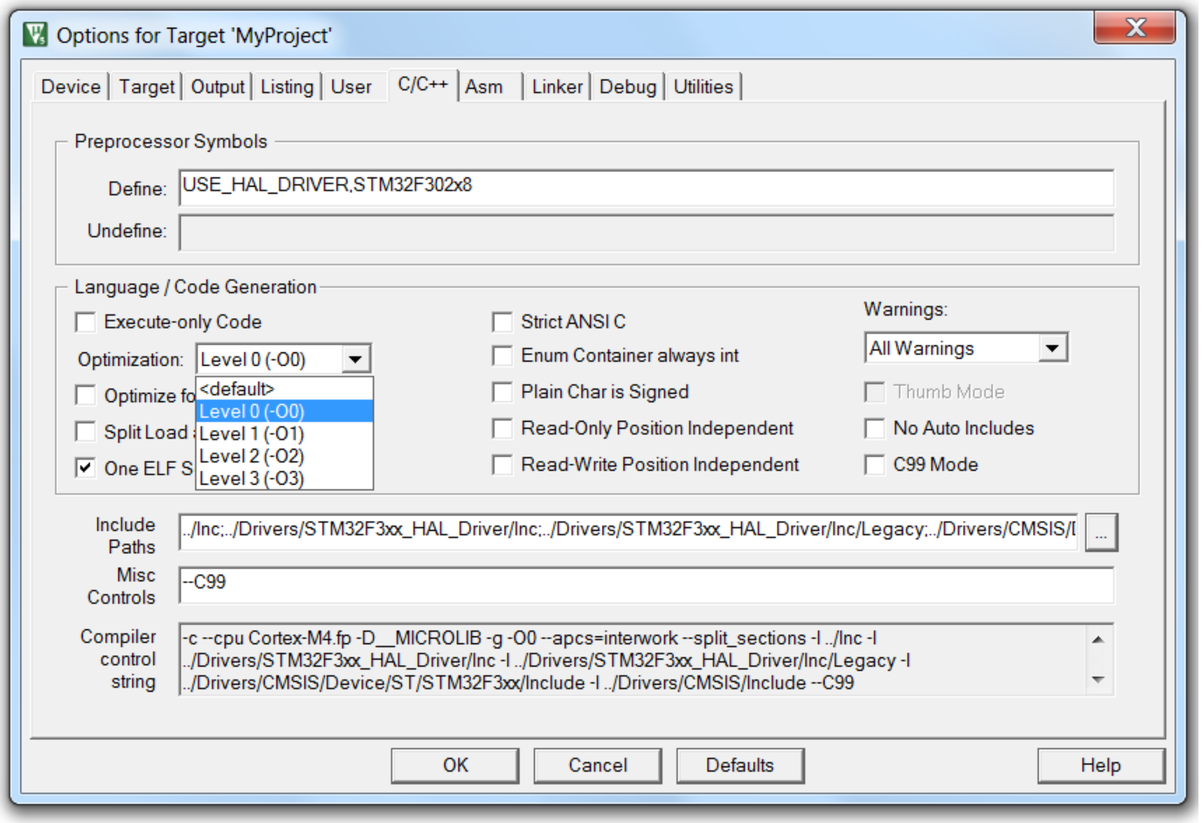
Instruction scheduling

Vectorization

3.1.2 Keil® MDK-ARM µVision

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

Figure 11. Keil® µVision Code Optimization option



Options for Target 'MyProject'

Device | Target | Output | Listing | User | C/C++ | Asm | Linker | Debug | Utilities

Preprocessor Symbols

Define: USE_HAL_DRIVER,STM32F302x8

Undefine:

Language / Code Generation

- [ ] Execute-only Code
- Optimization: Level 0 (-O0)
  - <default>
  - Level 0 (-O0)
  - Level 1 (-O1)
  - Level 2 (-O2)
  - Level 3 (-O3)
- [ ] Optimize fo
- [ ] Split Load
- [✓] One ELF S
- [ ] Strict ANSI C
- [ ] Enum Container always int
- [ ] Plain Char is Signed
- [ ] Read-Only Position Independent
- [ ] Read-Write Position Independent
- Warnings: All Warnings
- [ ] Thumb Mode
- [ ] No Auto Includes
- [ ] C99 Mode

Include Paths: ../Inc;../Drivers/STM32F3xx_HAL_Driver/Inc;../Drivers/STM32F3xx_HAL_Driver/Inc/Legacy;../Drivers/CMSIS/I

Misc Controls: --C99

Compiler control string:

```
-c --cpu Cortex-M4.fp -D__MICROLIB -g -O0 --apcs=interwork --split_sections -I ../Inc -I
../Drivers/STM32F3xx_HAL_Driver/Inc -I ../Drivers/STM32F3xx_HAL_Driver/Inc/Legacy -I
../Drivers/CMSIS/Device/ST/STM32F3xx/Include -I ../Drivers/CMSIS/Include --C99
```

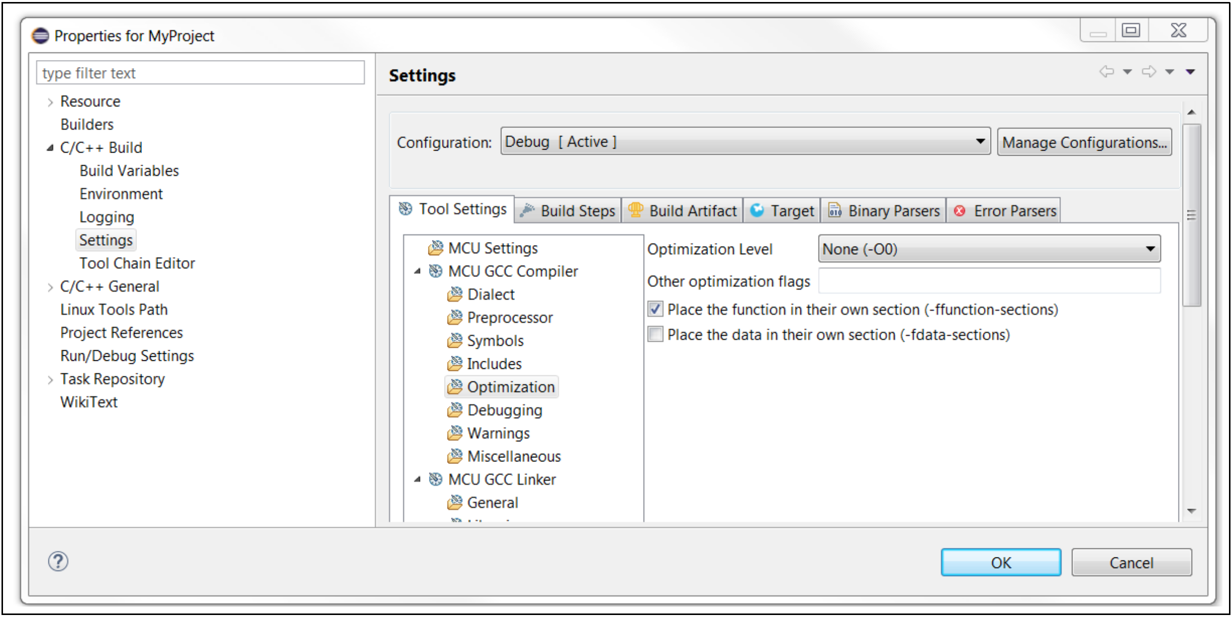
Keil® documentation suggests that Level1 (-O1) can be a suitable alternative for debug.

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

3.1.3 SW4STM32

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

Figure 12. SW4STM32 Optimization Level setting



Configuration: Debug [Active]

Optimization Level: None (-O0)

Other optimization flags:

[✓] Place the function in their own section (-ffunction-sections)

[ ] Place the data in their own section (-fdata-sections)

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.

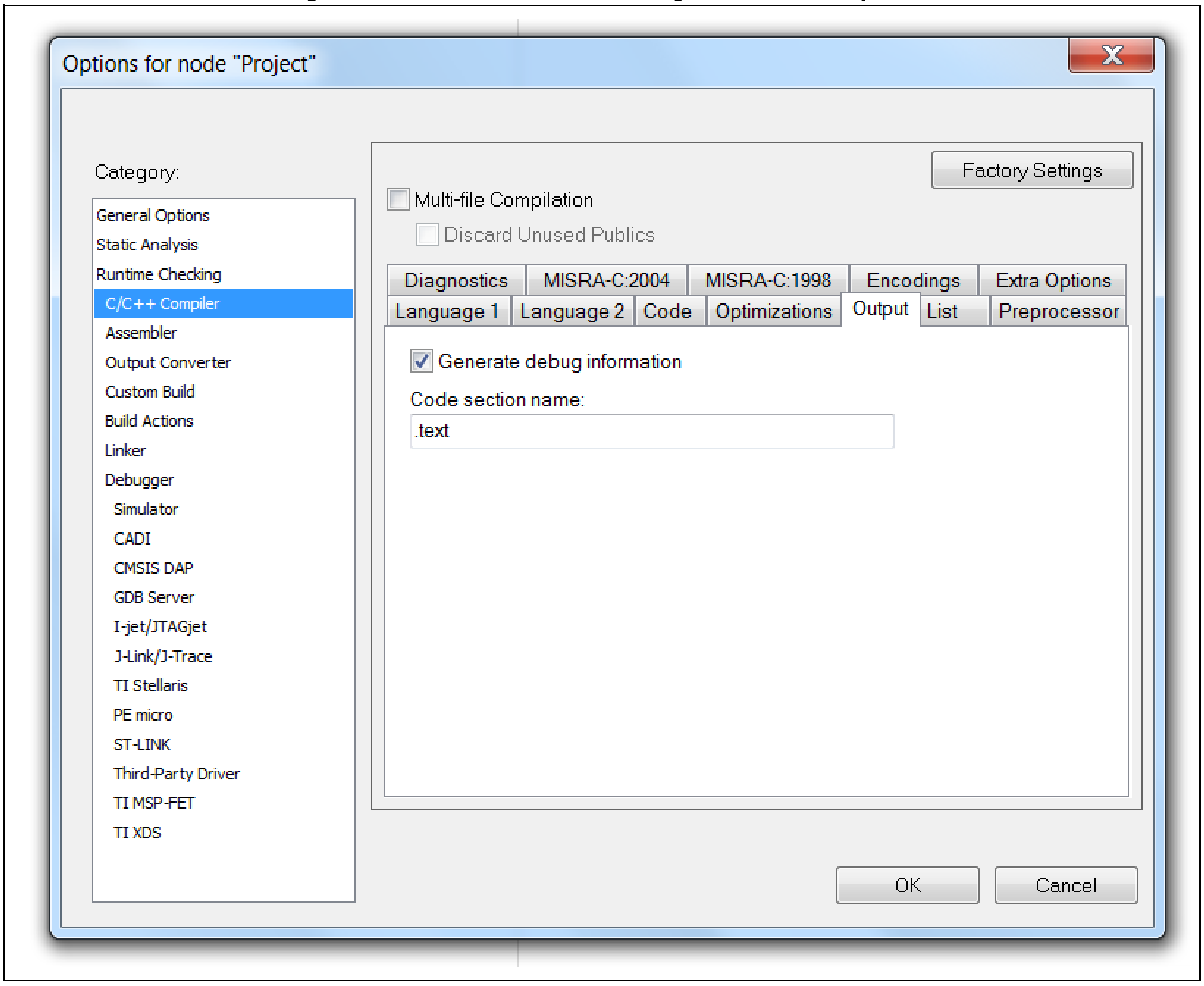
This chapter presents the location of the Debugging Information related option in IAR™, Keil®, and SW4STM32.

3.2.1 IAR™ EWARM

"Generate debug information" option tick box is accessible in ***Options -> C/C++ Compiler -> Output Pane***

It is set by default.

Figure 13. IAR™ Generate debug Information option



[ ] Multi-file Compilation

[ ] Discard Unused Publics

[✓] Generate debug information

Code section name:

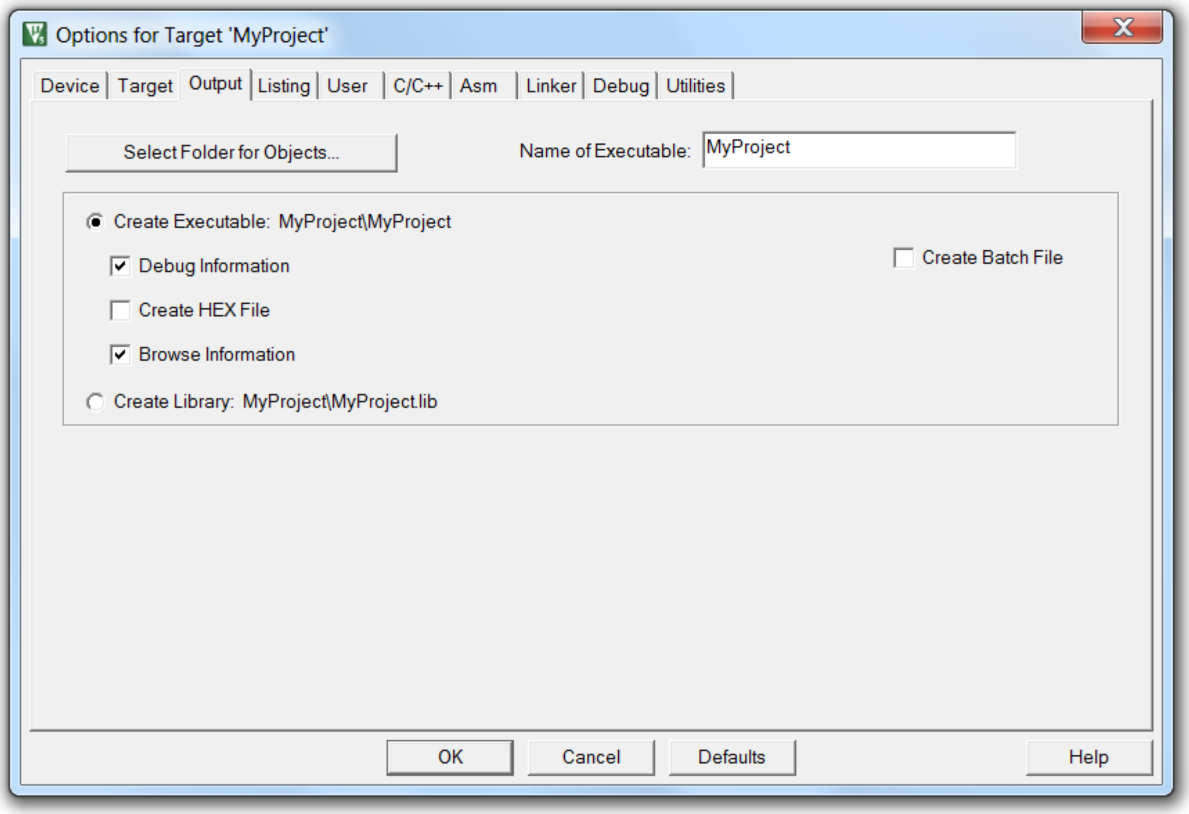
.text

3.2.2 Keil®-MDK-ARM µVision

Debug Information Tick box is accessible in ***Options -> Output Pane.***

It is set by default.

Figure 14. Keil® Debug Information option



Name of Executable: MyProject

(●) Create Executable: MyProject\MyProject

[✓] Debug Information

[ ] Create HEX File

[✓] Browse Information

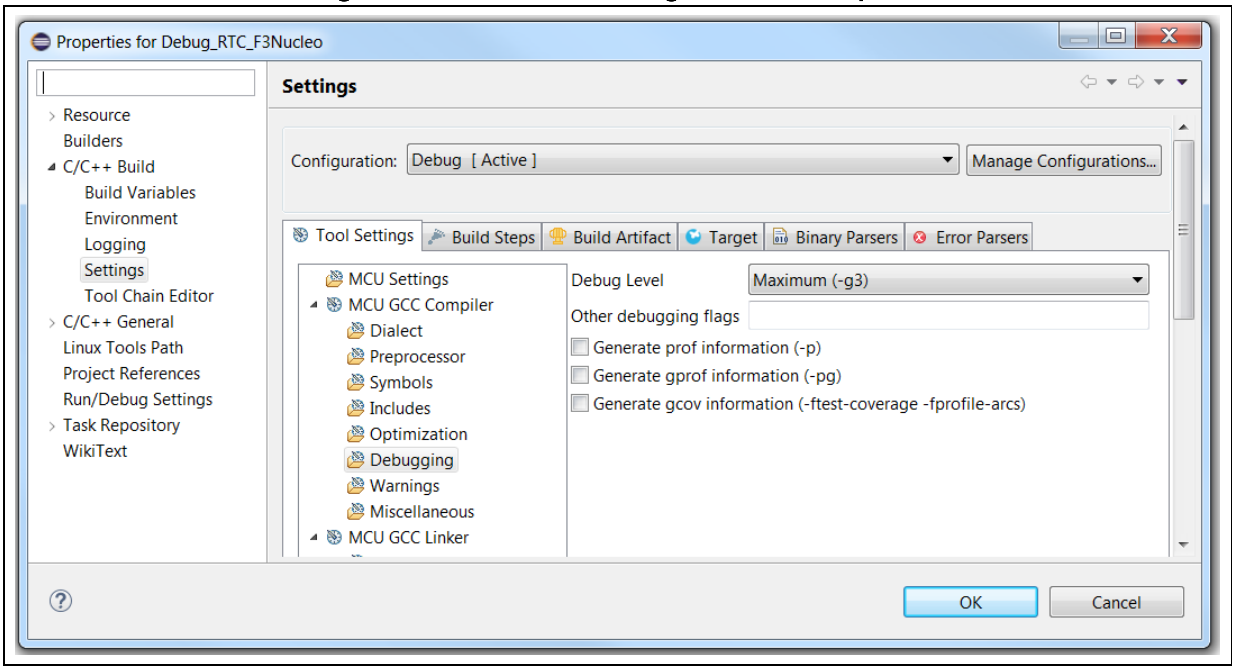
( ) Create Library: MyProject\MyProject.lib

[ ] Create Batch File

3.2.3 SW4STM32

Option to manage Debugging Information are in ***Properties -> C/C++ Build -> Settings -> Tool Settings -> Debugging.***

Figure 15. SW4STM32 Debug information option



Configuration: Debug [Active]
Debug Level: Maximum (-g3)
Other debugging flags:
☐ Generate prof information (-p)
☐ Generate gprof information (-pg)
☐ Generate gcov information (-ftest-coverage -fprofile-arcs)

Debug Level can be set among four levels:

- None (-g0): Level 0 produces no debug information at all; -g0 negates -g.
- 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/-g2): 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 16*
- Reserved but inactive shown by the pins colored in orange in *Figure 17*
- Active shown by the pins colored in green in *Figure 18*

Figure 16. SWD pins PA13 and PA14 in Reset state under STM32CubeMX

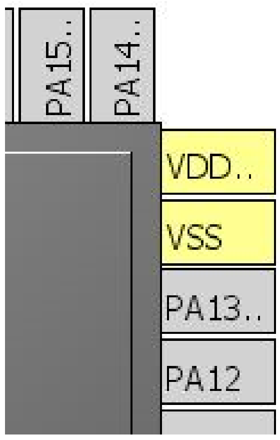


Figure 17. SWD pins PA13 and PA14 in Reserved but inactive state under STM32CubeMX

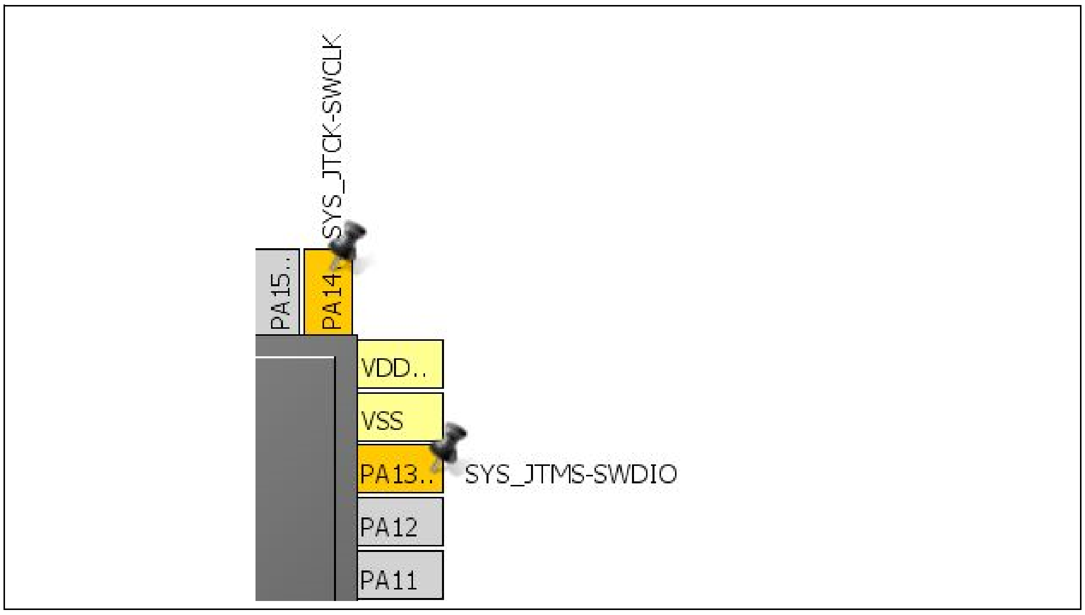
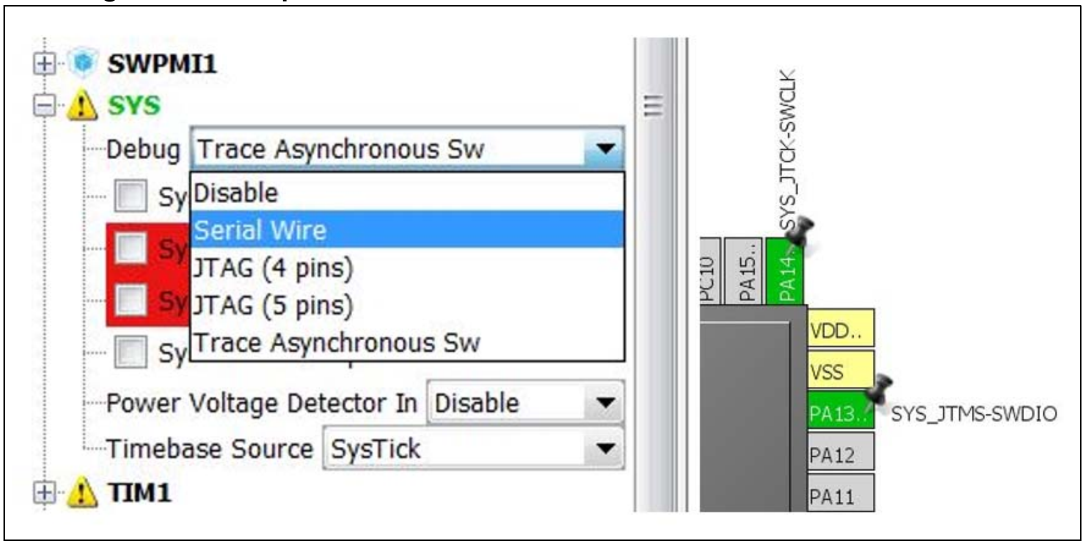


Figure 18. SWD pins PA13 and PA14 in Active State under STM32CubeMX



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 61*). 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/V2 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.

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:

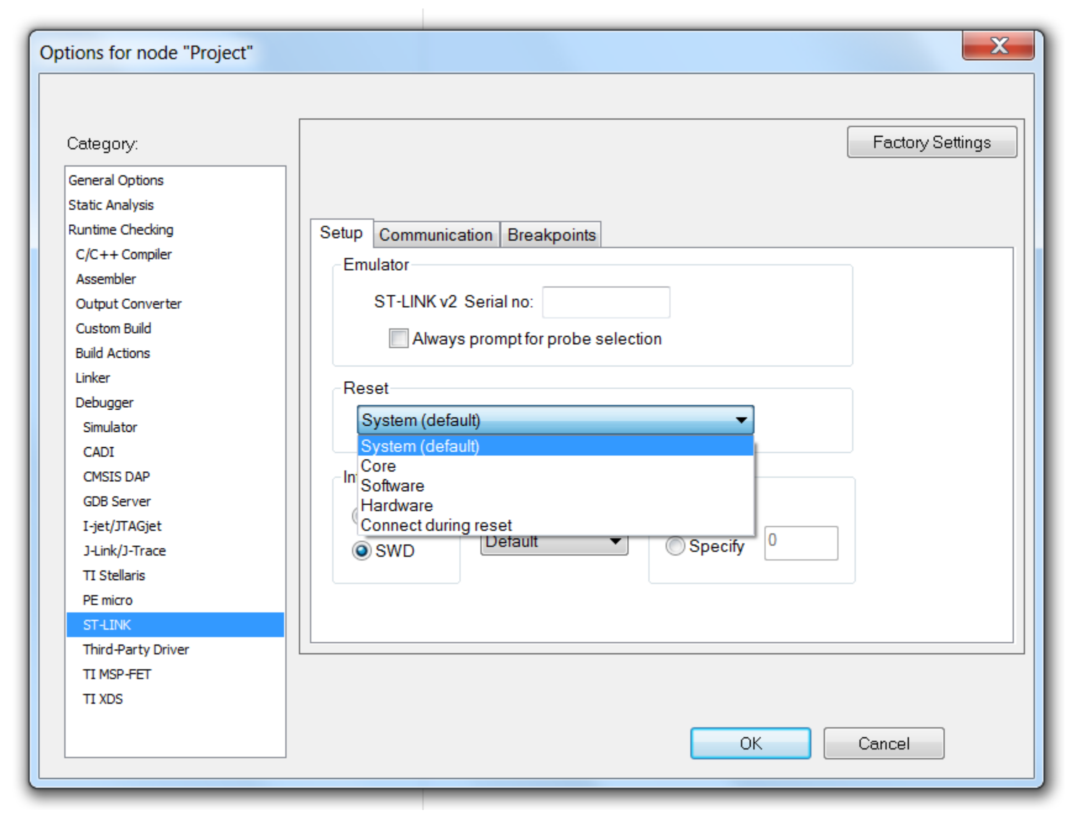
- Normal: Debugger takes control through JTAG/SWD port and starts execution after a software reset.
  This is working only if JTAG/SWD is available:
  - GPIO correctly configured and clocked
  - FCLK or HCLK enabled
  - Main Power domain or Low-Power debug active
- ConnectUnderReset: Debugger takes control 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 19*.

Figure 19. Reset Mode in IAR8.10: screenshot



ST-LINK v2 Serial no:

Always prompt for probe selection

Reset: System (default)

SWD

Default

Specify 0

- 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.

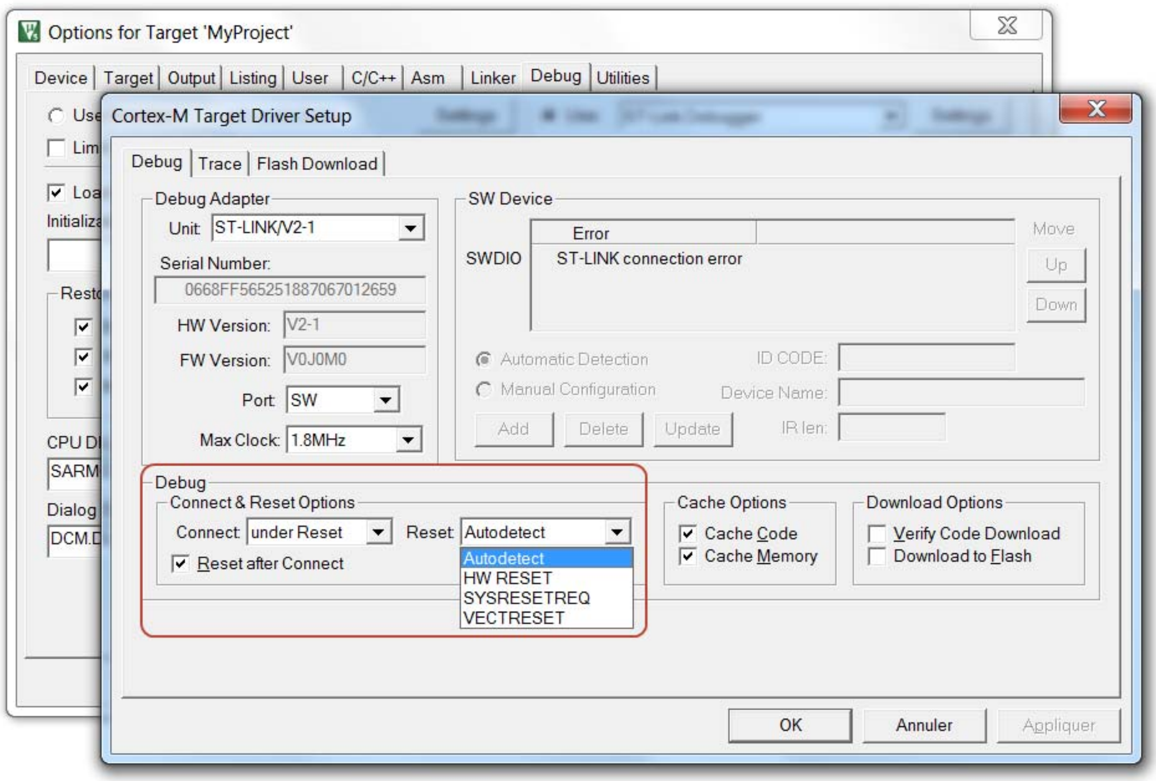
Hotplug connection is accessible with "Attach to running Target" function in project menu.

4.2.3 Keil® MDK-ARM µVISION

Can be set through

Options -> Debug -> Settings -> Debug

Figure 20. Connect and Reset option Keil®



Debug Adapter
Unit: ST-LINK/V2-1
Serial Number: 0668FF565251887067012659
HW Version: V2-1
FW Version: V0J0M0
Port: SW
Max Clock: 1.8MHz

SW Device
SWDIO Error: ST-LINK connection error
Automatic Detection ID CODE:
Manual Configuration Device Name:
IR len:

Debug
Connect & Reset Options
Connect: under Reset Reset: Autodetect
Autodetect / HW RESET / SYSRESETREQ / VECTRESET
✓ Reset after Connect

Cache Options
✓ Cache Code
✓ Cache Memory

Download Options
☐ Verify Code Download
☐ Download to Flash

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.

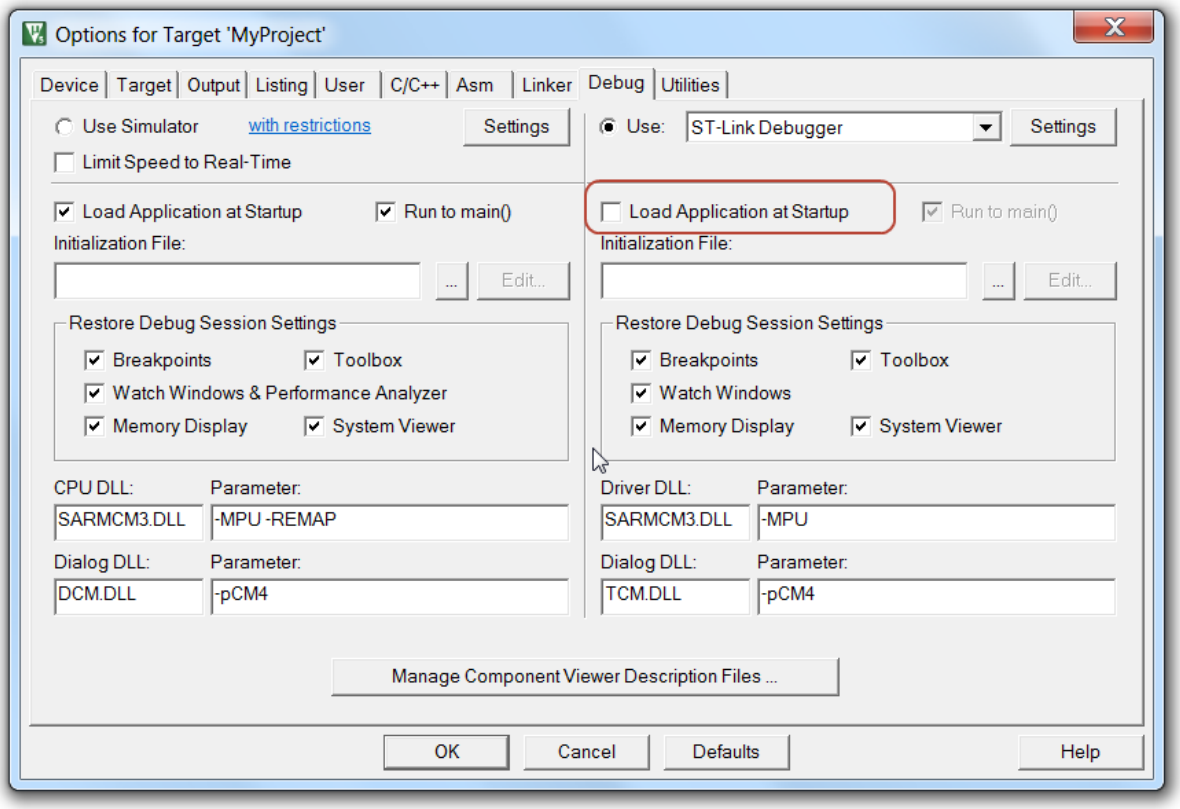
Refer to http://www.keil.com/

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 21. Keil® hotplug step1



Options for Target 'MyProject'

Device | Target | Output | Listing | User | C/C++ | Asm | Linker | Debug | Utilities

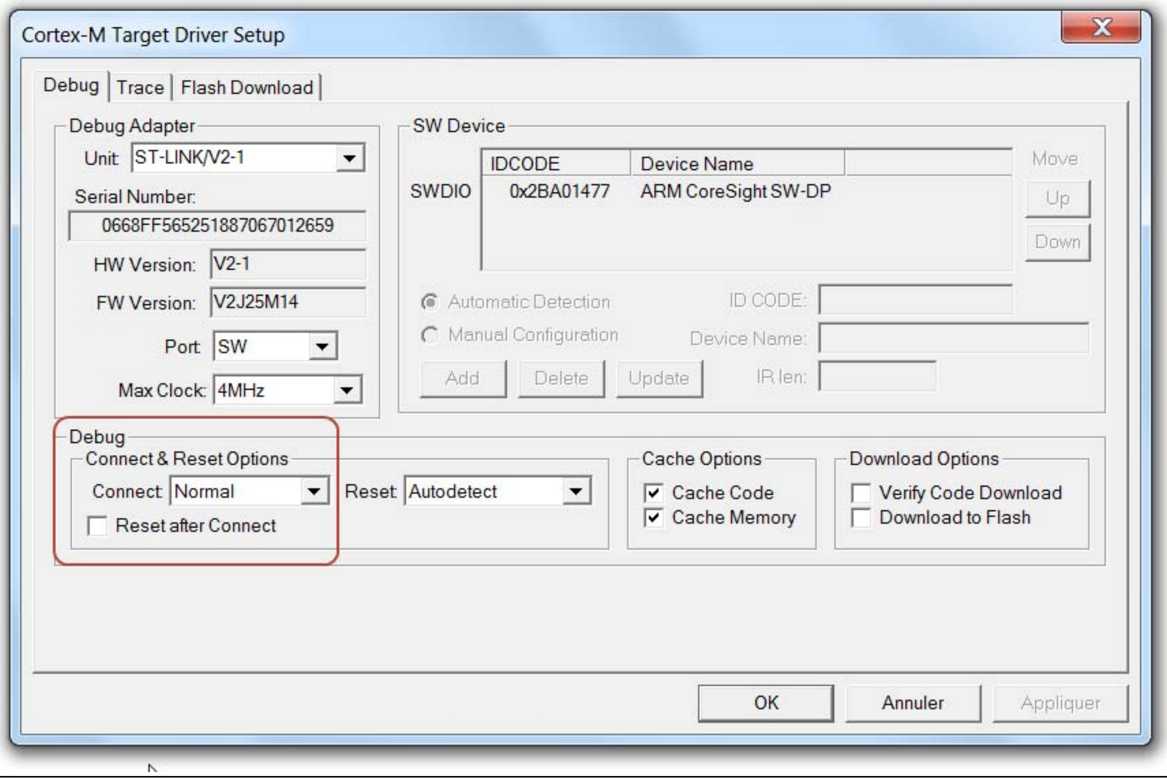
( ) Use Simulator with restrictions Settings
[ ] Limit Speed to Real-Time
[✓] Load Application at Startup [✓] Run to main()
Initialization File:
Restore Debug Session Settings
[✓] Breakpoints [✓] Toolbox
[✓] Watch Windows & Performance Analyzer
[✓] Memory Display [✓] System Viewer
CPU DLL: SARMCM3.DLL Parameter: -MPU -REMAP
Dialog DLL: DCM.DLL Parameter: -pCM4

(●) Use: ST-Link Debugger Settings
[ ] Load Application at Startup [✓] Run to main()
Initialization File:
Restore Debug Session Settings
[✓] Breakpoints [✓] Toolbox
[✓] Watch Windows
[✓] Memory Display [✓] System Viewer
Driver DLL: SARMCM3.DLL Parameter: -MPU
Dialog DLL: TCM.DLL Parameter: -pCM4

Manage Component Viewer Description Files ...

OK Cancel Defaults Help

Figure 22. Keil® hotplug step2


Cortex-M Target Driver Setup

Debug | Trace | Flash Download

Debug Adapter

Unit: ST-LINK/V2-1

Serial Number: 0668FF565251887067012659

HW Version: V2-1

FW Version: V2J25M14

Port: SW

Max Clock: 4MHz

SW Device

	IDCODE	Device Name
SWDIO	0x2BA01477	ARM CoreSight SW-DP

[✓ Automatic Detection] Manual Configuration

ID CODE:

Device Name:

IR len:

Debug

Connect & Reset Options

Connect: Normal

Reset: Autodetect

☐ Reset after Connect

Cache Options

☑ Cache Code

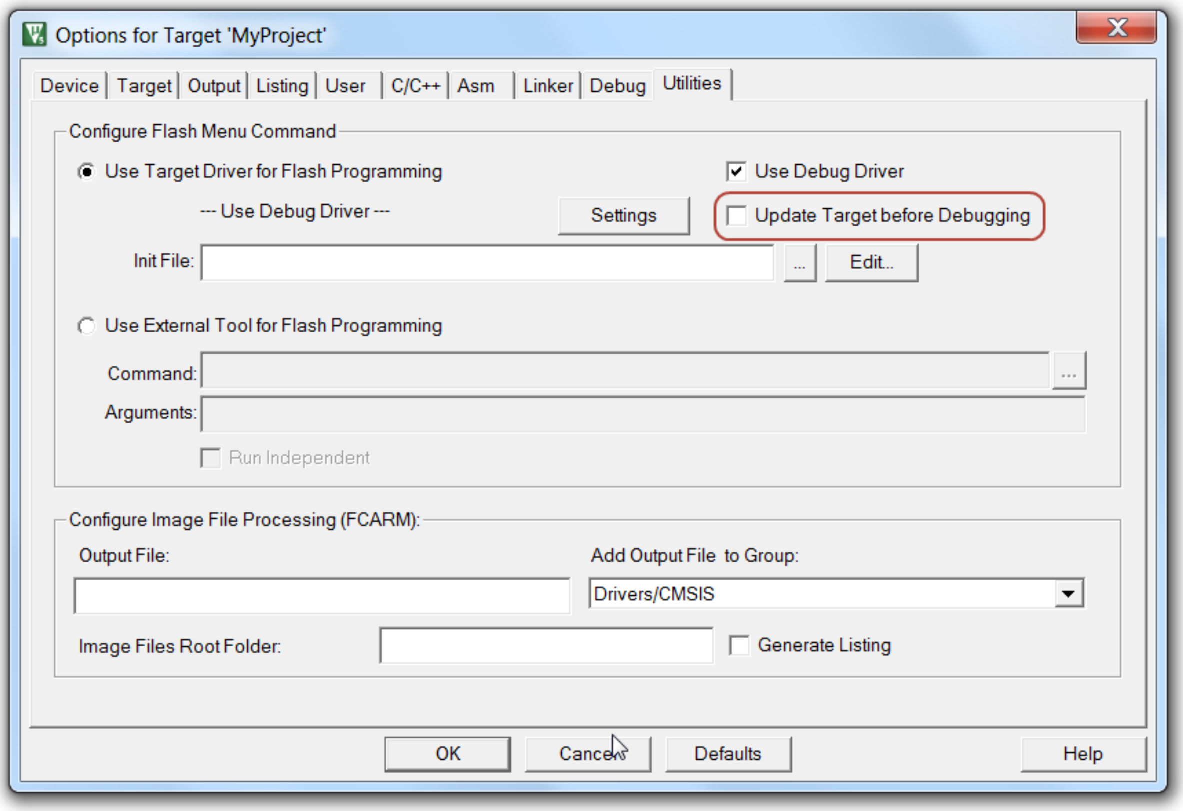
☑ Cache Memory

Download Options

☐ Verify Code Download

☐ Download to Flash

Figure 23. Keil® hotplug step3


Options for Target 'MyProject'

Device | Target | Output | Listing | User | C/C++ | Asm | Linker | Debug | Utilities

Configure Flash Menu Command

(●) Use Target Driver for Flash Programming — [✓] Use Debug Driver

--- Use Debug Driver --- Settings — [ ] Update Target before Debugging

Init File:

( ) Use External Tool for Flash Programming

Command:

Arguments:

[ ] Run Independent

Configure Image File Processing (FCARM):

Output File:

Add Output File to Group: Drivers/CMSIS

Image Files Root Folder:

[ ] Generate Listing

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.

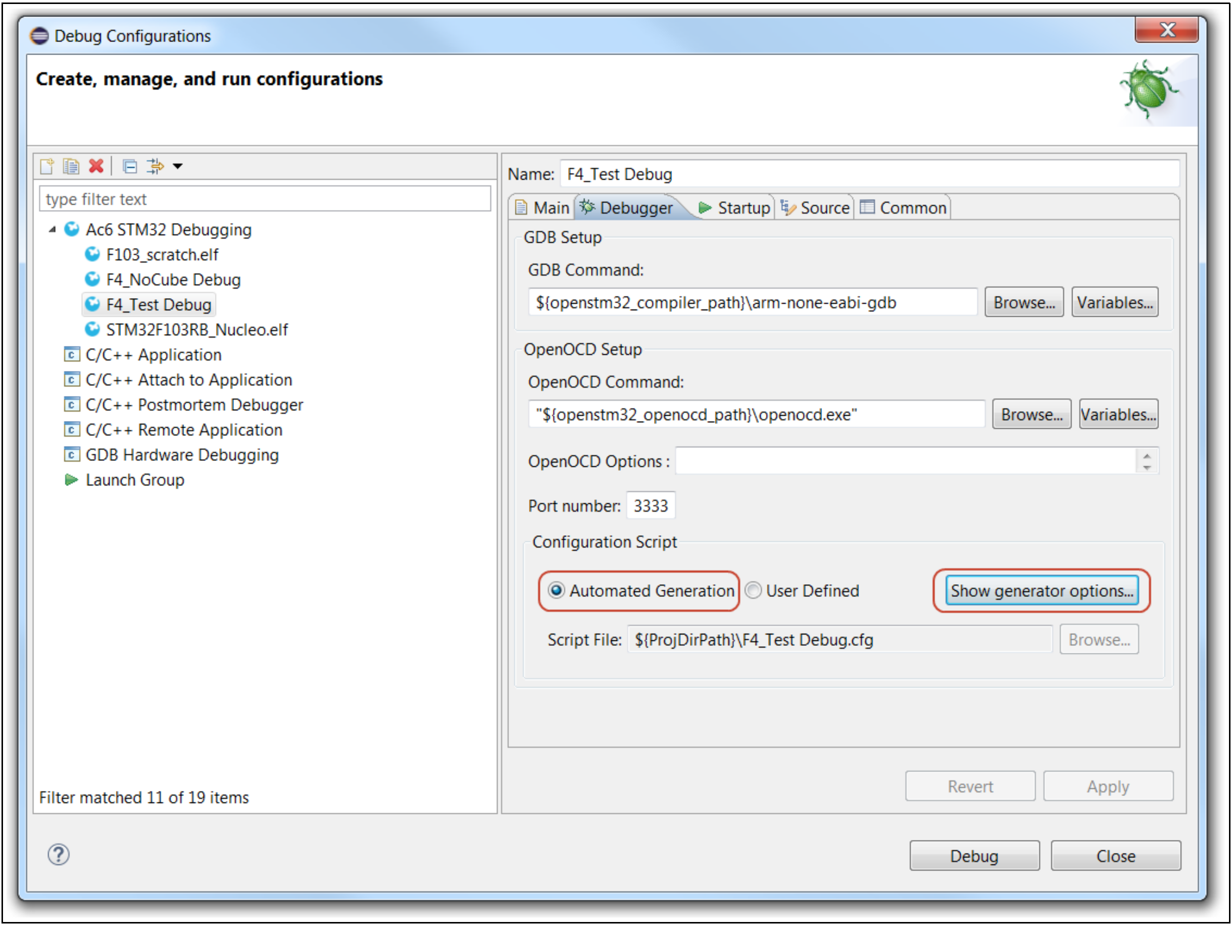
LOAD can be automated using an Initialization File under Options For Target - Debug.

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

4.2.4 SW4STM32

Since version V2.0.0 of SW4STM32, reset and connection modes can be changed through the Generator Options GUI in ***Debug Configuration*** -> ***Debugger Pane*** by clicking on the Show generator options as presented in *Figure 24* and *Figure 25*.

Figure 24. Access to Generator Options in SW4STM32 V2.0.0



Name: F4_Test Debug

GDB Setup

GDB Command: ${openstm32_compiler_path}\arm-none-eabi-gdb

OpenOCD Setup

OpenOCD Command: "${openstm32_openocd_path}\openocd.exe"

OpenOCD Options :

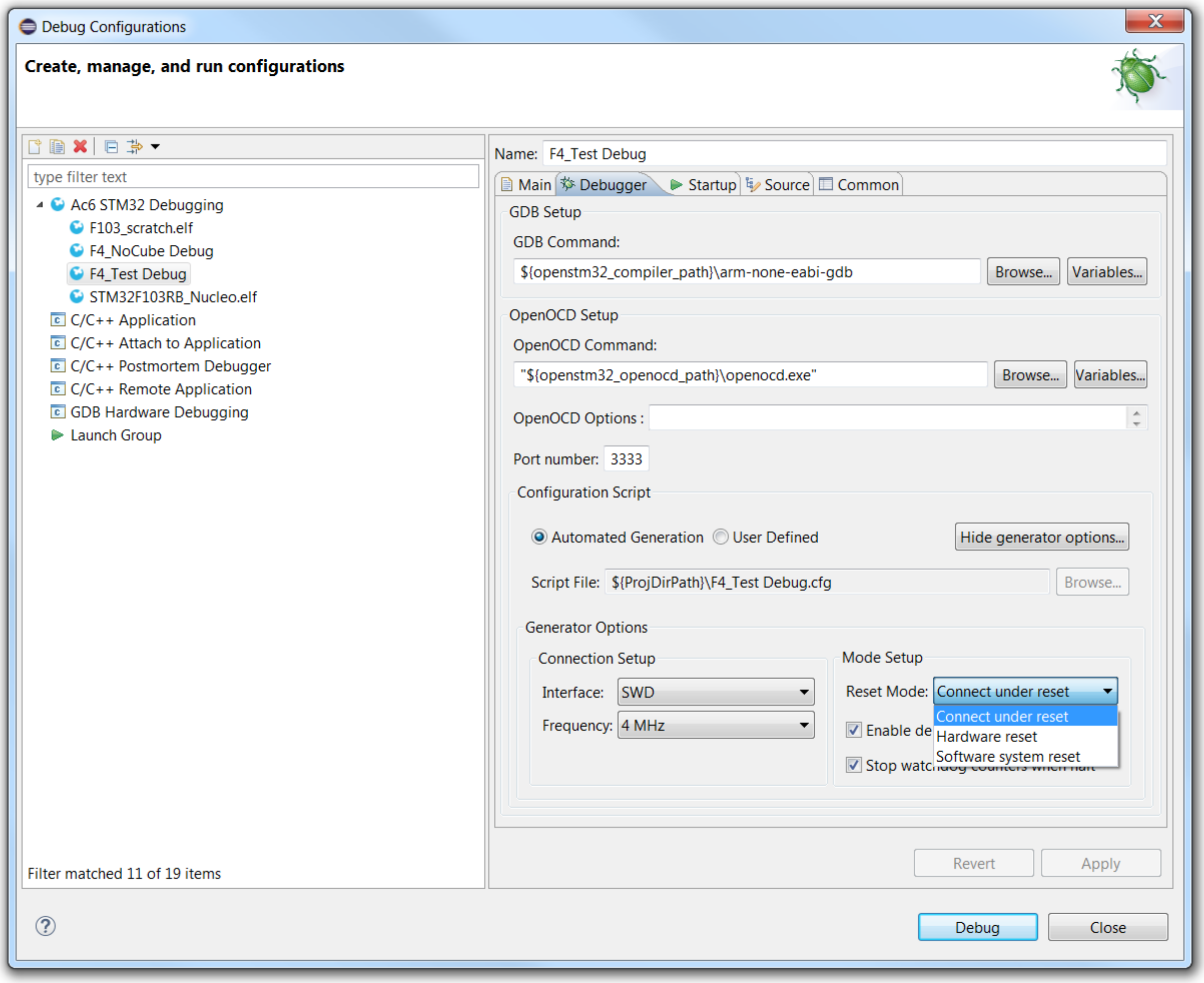
Port number: 3333

Configuration Script

[✓ Automated Generation] User Defined — Show generator options...

Script File: ${ProjDirPath}\F4_Test Debug.cfg

Figure 25. Select Generator Options Reset Mode in SW4STM32 V2.0.0



Debug Configurations

Create, manage, and run configurations

type filter text

- Ac6 STM32 Debugging
  - F103_scratch.elf
  - F4_NoCube Debug
  - F4_Test Debug
  - STM32F103RB_Nucleo.elf
- C/C++ Application
- C/C++ Attach to Application
- C/C++ Postmortem Debugger
- C/C++ Remote Application
- GDB Hardware Debugging
- Launch Group

Filter matched 11 of 19 items

Name: F4_Test Debug

Main | Debugger | Startup | Source | Common

GDB Setup

GDB Command: ${openstm32_compiler_path}\arm-none-eabi-gdb

OpenOCD Setup

OpenOCD Command: "${openstm32_openocd_path}\openocd.exe"

OpenOCD Options :

Port number: 3333

Configuration Script

[✓ Automated Generation] User Defined — Hide generator options...

Script File: ${ProjDirPath}\F4_Test Debug.cfg

Generator Options

Connection Setup

Interface: SWD

Frequency: 4 MHz

Mode Setup

Reset Mode: Connect under reset

- Connect under reset
- Hardware reset
- Software system reset

☑ Enable de

☑ Stop watc

Revert Apply

Debug Close

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 Hardware reset: 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.

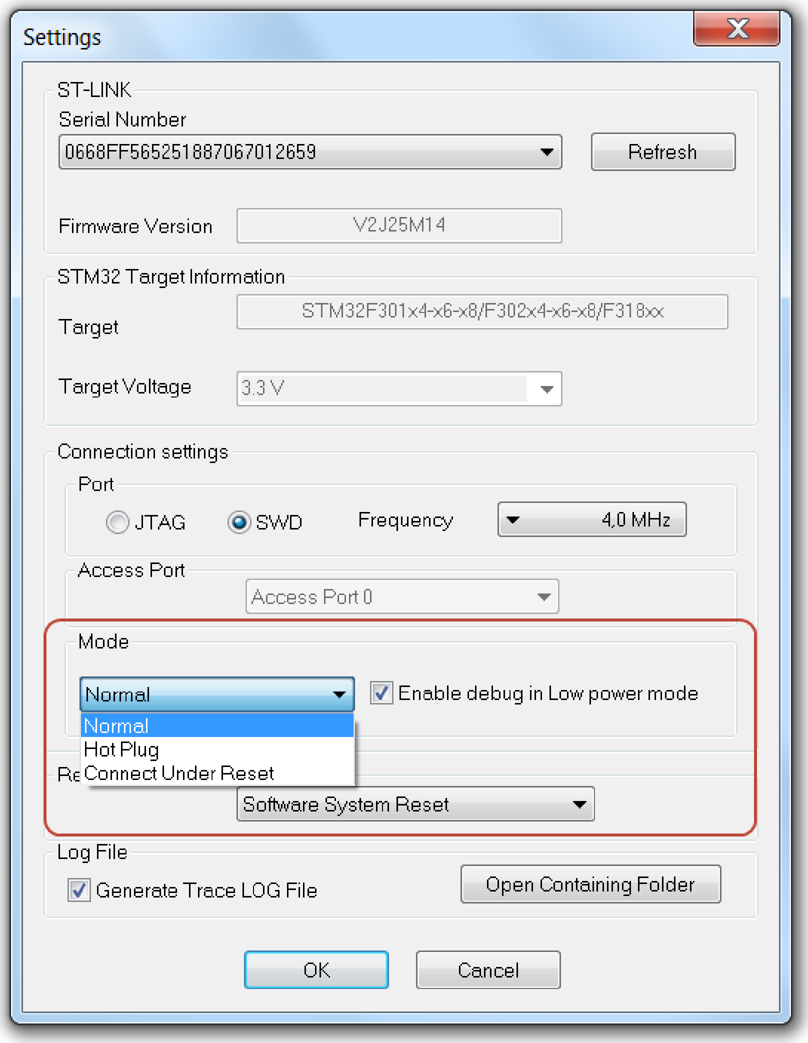
In case of problem to connect to the board with SW4STM32, make sure that NRST from ST-LINK is properly connected to STM32 NRST.

Hotplug mode is not proposed by SW4STM32. ST-LINK utility can be used instead.

4.2.5 ST-LINK utility

Reset and Connection modes can be selected in the Settings Pane according to their description in *Section 4.2.1 on page 31*.

Figure 26. Connection and reset mode in ST-Link utility



Settings

ST-LINK
Serial Number: 0668FF565251887067012659
Refresh
Firmware Version: V2J25M14

STM32 Target Information
Target: STM32F301x4-x6-x8/F302x4-x6-x8/F318xx
Target Voltage: 3.3 V

Connection settings
Port: JTAG [✓ SWD]
Frequency: 4.0 MHz
Access Port: Access Port 0

Mode: Normal
[✓] Enable debug in Low power mode
Normal
Hot Plug
Connect Under Reset
Re
Software System Reset

Log File
[✓] Generate Trace LOG File
Open Containing Folder

OK
Cancel

Note: In Keil® MDK-ARM µVISION, IAR™ EWARM and ST-LINK utility, 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 78 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: From debug to release on page 76*).

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 78* to find suitable ways to handle debug options depending on IDEs and needs.

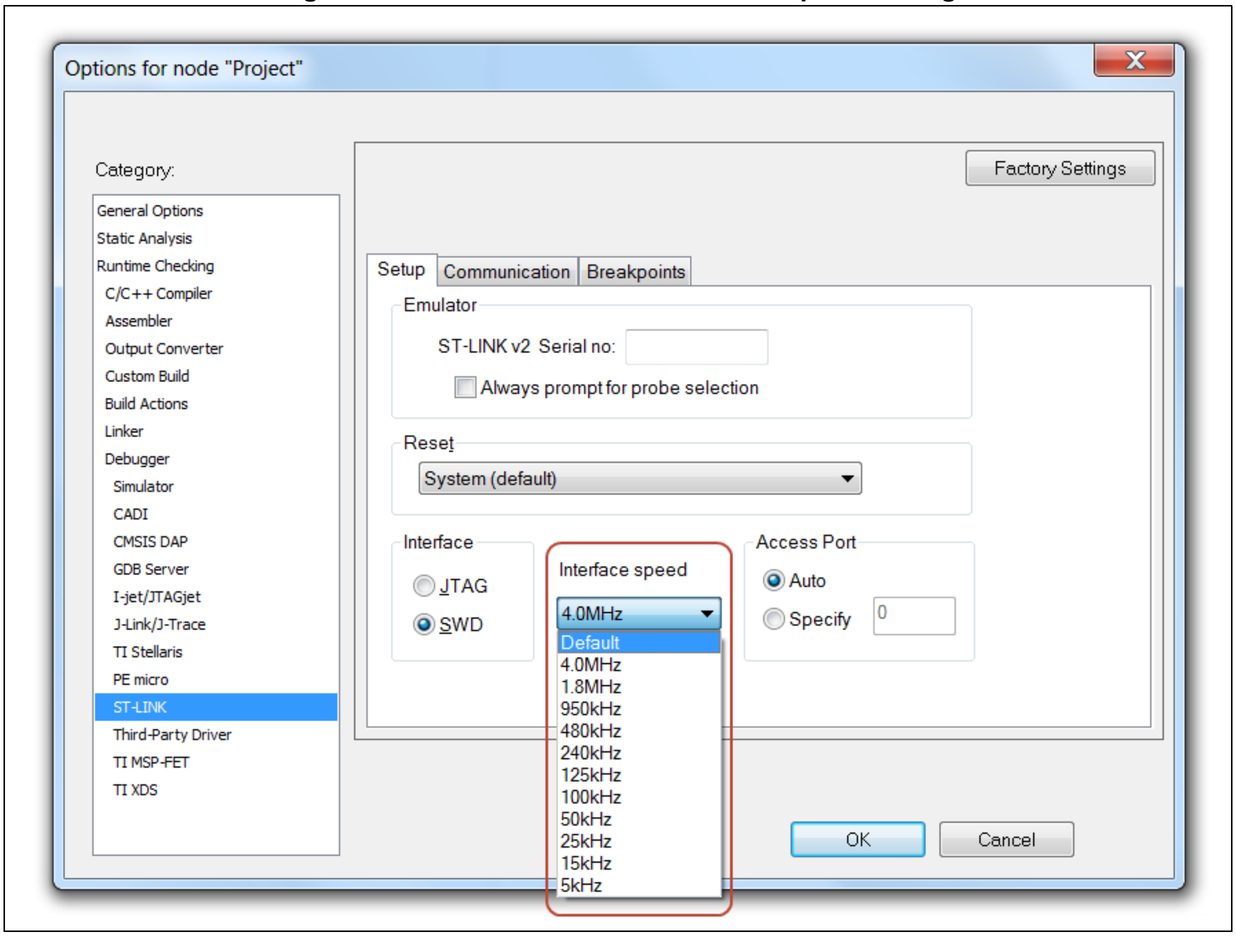
5.2 Debug performance

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 ST-LINK utility, IAR™ EWARM, or Keil® MDK-ARM µVISION 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

Figure 27. IAR™ EWARM ST-LINK SWD Speed setting



Options for node "Project"

Category:

- General Options
- Static Analysis
- Runtime Checking
- C/C++ Compiler
- Assembler
- Output Converter
- Custom Build
- Build Actions
- Linker
- Debugger
- Simulator
- CADI
- CMSIS DAP
- GDB Server
- I-jet/JTAGjet
- J-Link/J-Trace
- TI Stellaris
- PE micro
- ST-LINK
- Third-Party Driver
- TI MSP-FET
- TI XDS

Factory Settings

Setup | Communication | Breakpoints

Emulator

ST-LINK v2 Serial no:

☐ Always prompt for probe selection

Reset: System (default)

Interface: JTAG [✓ SWD]

Interface speed: 4.0MHz

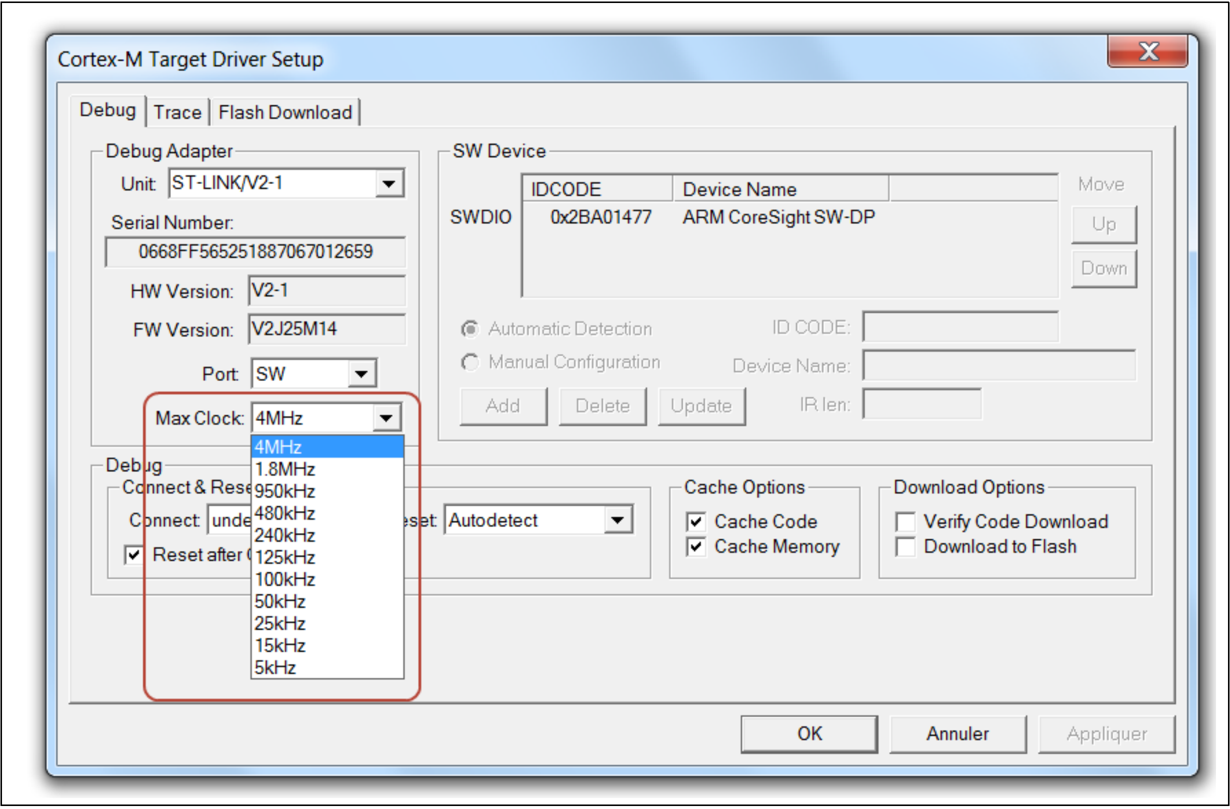
- Default
- 4.0MHz
- 1.8MHz
- 950kHz
- 480kHz
- 240kHz
- 125kHz
- 100kHz
- 50kHz
- 25kHz
- 15kHz
- 5kHz

Access Port: [✓ Auto] Specify 0

OK Cancel

5.2.2 Keil® MDK-ARM µVISION

Figure 28. Keil® SWD Speed Setting



Cortex-M Target Driver Setup

Debug | Trace | Flash Download

Debug Adapter
Unit: ST-LINK/V2-1
Serial Number: 0668FF565251887067012659
HW Version: V2-1
FW Version: V2J25M14
Port: SW
Max Clock: 4MHz

4MHz
1.8MHz
950kHz
480kHz
240kHz
125kHz
100kHz
50kHz
25kHz
15kHz
5kHz

SW Device

	IDCODE	Device Name
SWDIO	0x2BA01477	ARM CoreSight SW-DP

Automatic Detection
Manual Configuration
ID CODE:
Device Name:
IR len:

Debug
Connect & Rese
Connect: unde
eset: Autodetect
✓ Reset after

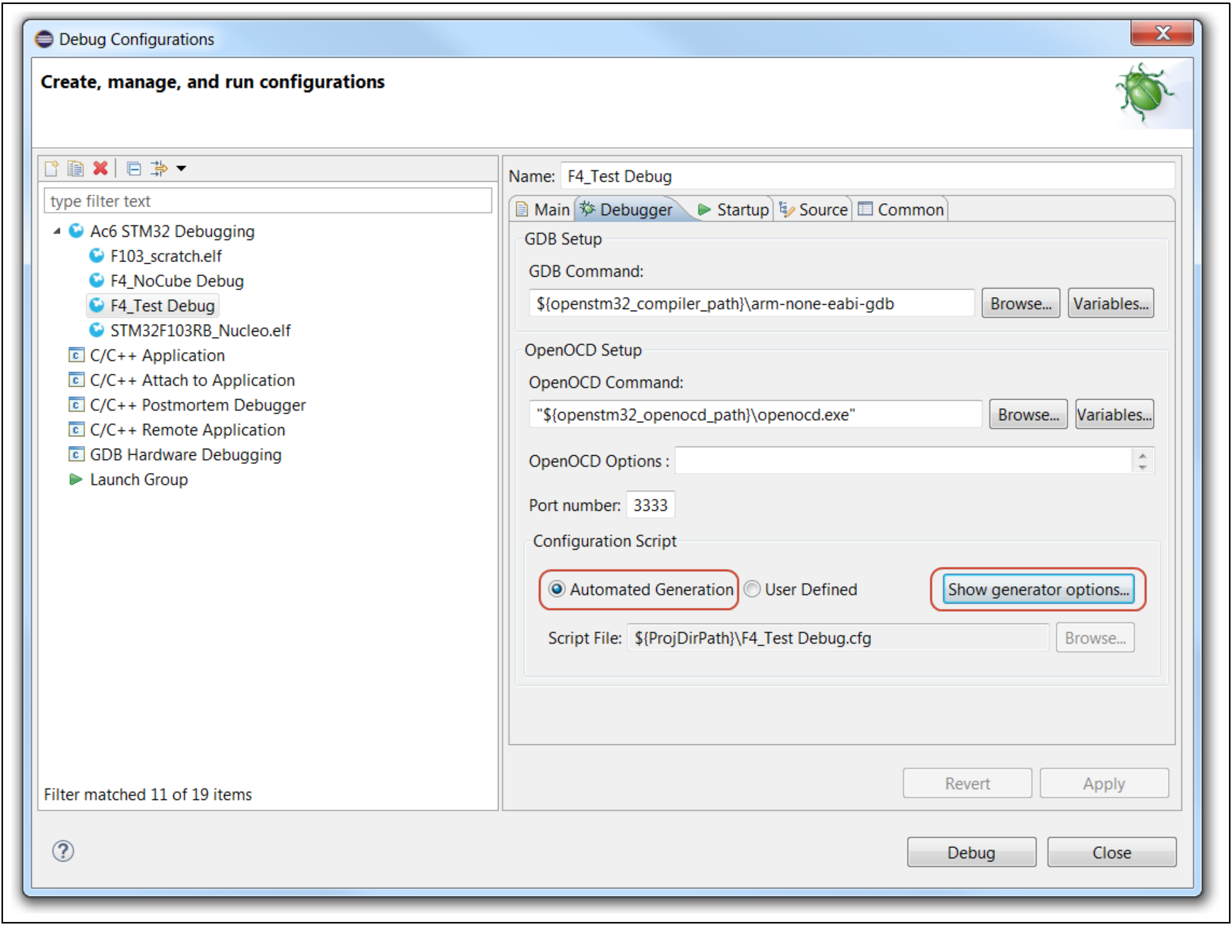
Cache Options
✓ Cache Code
✓ Cache Memory

Download Options
Verify Code Download
Download to Flash

5.2.3 SW4STM32

Since version V2.0.0 of SW4STM32, the ST-LINK speed setting can be changed through the OpenOCD Generator Options GUI in ***Debug Configuration*** -> ***Debugger Pane*** by clicking on the Show generator options as presented in *Figure 29*.

Figure 29. Access to Generator Options in SW4STM32 V2.0.0



Debug Configurations

Create, manage, and run configurations

type filter text

- Ac6 STM32 Debugging
  - F103_scratch.elf
  - F4_NoCube Debug
  - F4_Test Debug
  - STM32F103RB_Nucleo.elf
- C/C++ Application
- C/C++ Attach to Application
- C/C++ Postmortem Debugger
- C/C++ Remote Application
- GDB Hardware Debugging
- Launch Group

Filter matched 11 of 19 items

Name: F4_Test Debug

Main | Debugger | Startup | Source | Common

GDB Setup

GDB Command: ${openstm32_compiler_path}\arm-none-eabi-gdb

OpenOCD Setup

OpenOCD Command: "${openstm32_openocd_path}\openocd.exe"

OpenOCD Options :

Port number: 3333

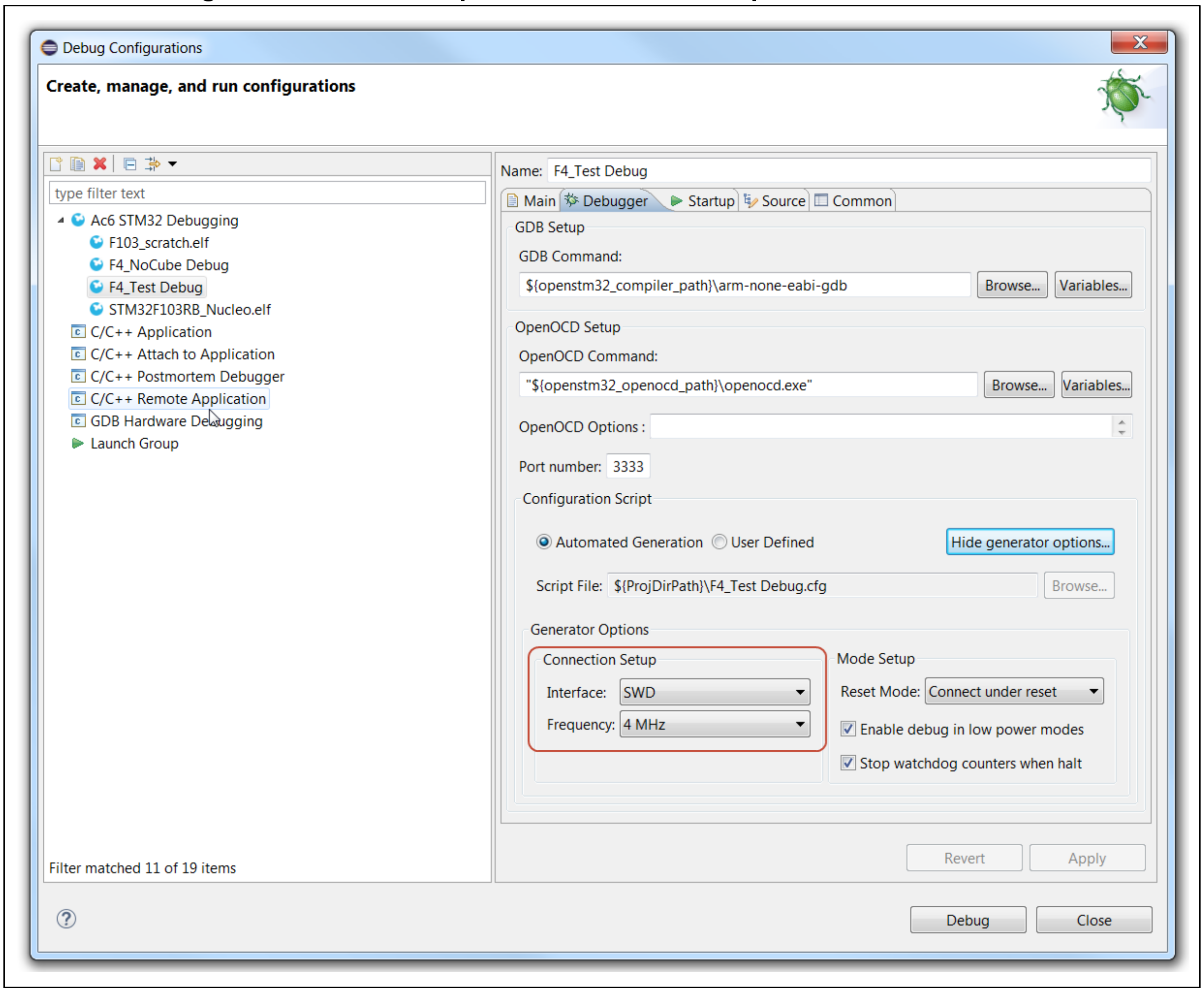
Configuration Script

[✓ Automated Generation] User Defined — Show generator options...

Script File: ${ProjDirPath}\F4_Test Debug.cfg

The ST-LINK connection mode and speed are available in the Connection Setup Group as shown in *Figure 30*.

Figure 30. Generator Options Connection Setup in SW4STM32 V2.0.0


Debug Configurations

Create, manage, and run configurations

type filter text

- Ac6 STM32 Debugging
  - F103_scratch.elf
  - F4_NoCube Debug
  - F4_Test Debug
  - STM32F103RB_Nucleo.elf
- C/C++ Application
- C/C++ Attach to Application
- C/C++ Postmortem Debugger
- C/C++ Remote Application
- GDB Hardware Debugging
- Launch Group

Filter matched 11 of 19 items

Name: F4_Test Debug

Main | Debugger | Startup | Source | Common

GDB Setup

GDB Command: ${openstm32_compiler_path}\arm-none-eabi-gdb

OpenOCD Setup

OpenOCD Command: "${openstm32_openocd_path}\openocd.exe"

OpenOCD Options :

Port number: 3333

Configuration Script

[✓ Automated Generation] User Defined

Script File: ${ProjDirPath}\F4_Test Debug.cfg

Generator Options

Connection Setup

Interface: SWD

Frequency: 4 MHz

Mode Setup

Reset Mode: Connect under reset

✓ Enable debug in low power modes

✓ Stop watchdog counters when halt

The Connection Setup group allows to specify the debug probe communication channel and clock speed.

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 97 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
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 connexion is killed. There is no way back. In this case, nobody - even STMicroelectronics - can perform any analysis of defective parts.

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, SW4STM32).

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

In SW4STM32 all weak default handlers point to the same DefaultHandler which can be confusing.

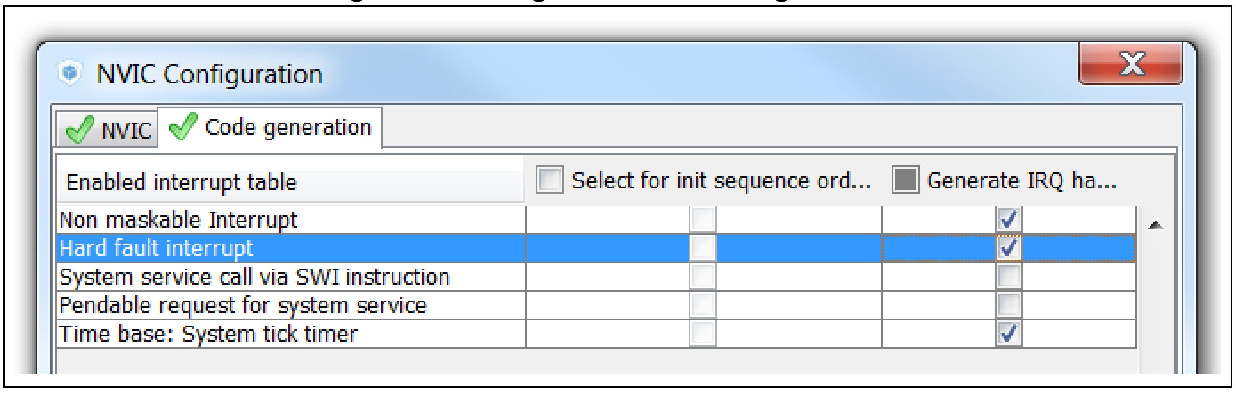
A more efficient approach is to trap the exception by instrumenting Handlers.

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 31*.

Figure 31. Asking for Handler code generation



Enabled interrupt table	Select for init sequence ord...	Generate IRQ ha...
Non maskable Interrupt	☐	✓
Hard fault interrupt	☐	✓
System service call via SWI instruction	☐	☐
Pendable request for system service	☐	☐
Time base: System tick timer	☐	✓

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: From debug to release on page 76*). it is advised to set it under `#ifdef` statement.

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

- IAR™ and SW4STM32

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

- Keil®

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

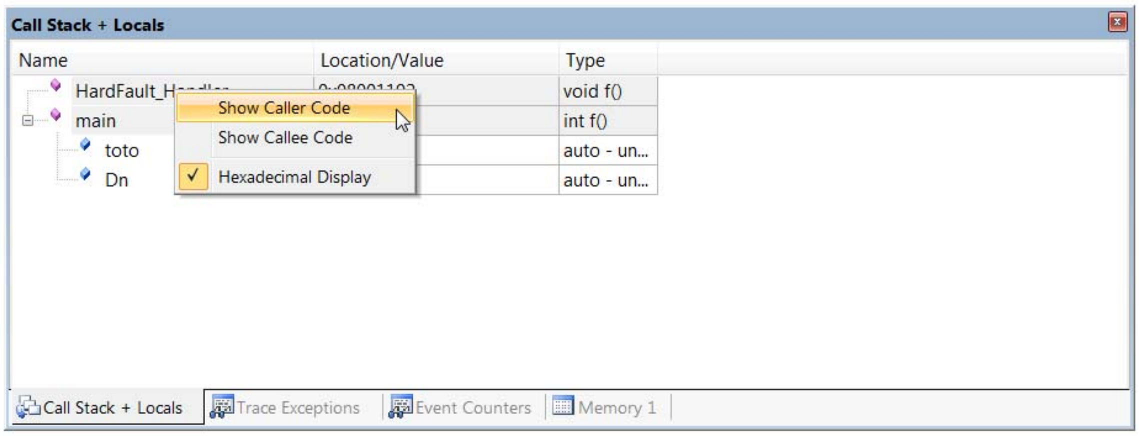
For each IDE, it is also possible to use the abstraction function defined in the CMSIS library and provided in STM32Cube software pack.

```
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 32* leads to the faulty line.

Figure 32. 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*.

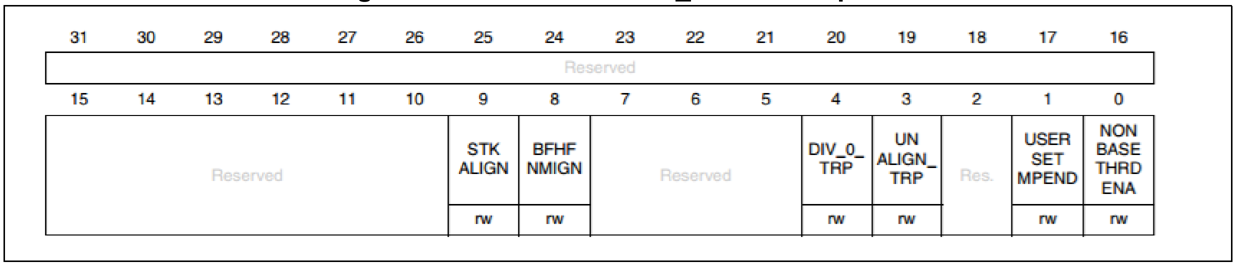
6.3.2 Cortex®-M3/4/7 case

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 33*.

Figure 33. Cortex®-M3 SCB_CCR Description



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

Setting bit 5 of SCB_CCR register

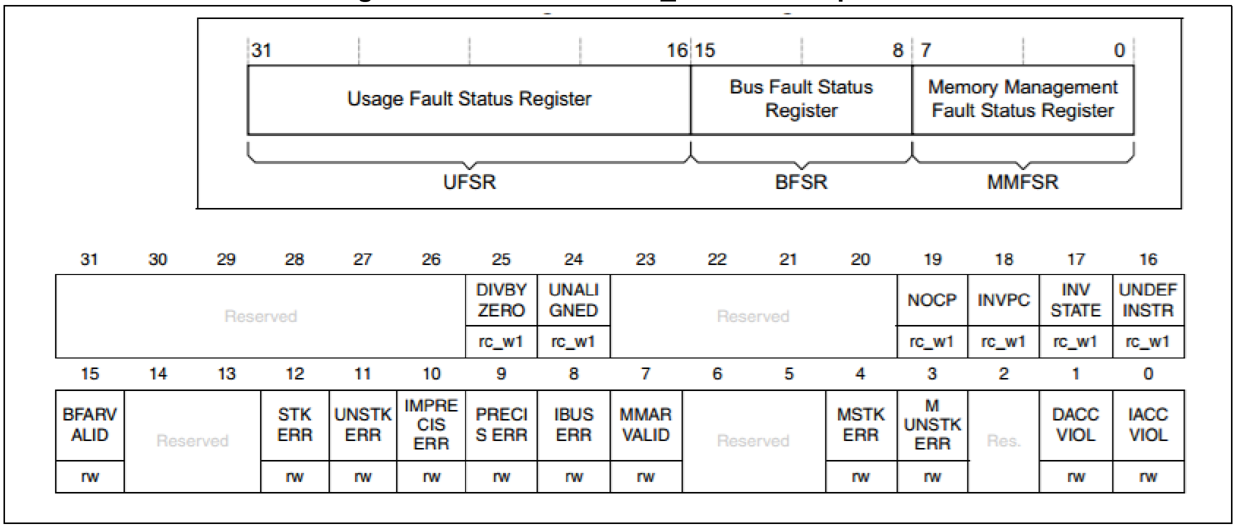
```
SCB->CCR |= 0x10; // enable div-by-0 trap
```

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 34*.

Figure 34. Cortex-M3 SCB_CFSR Description

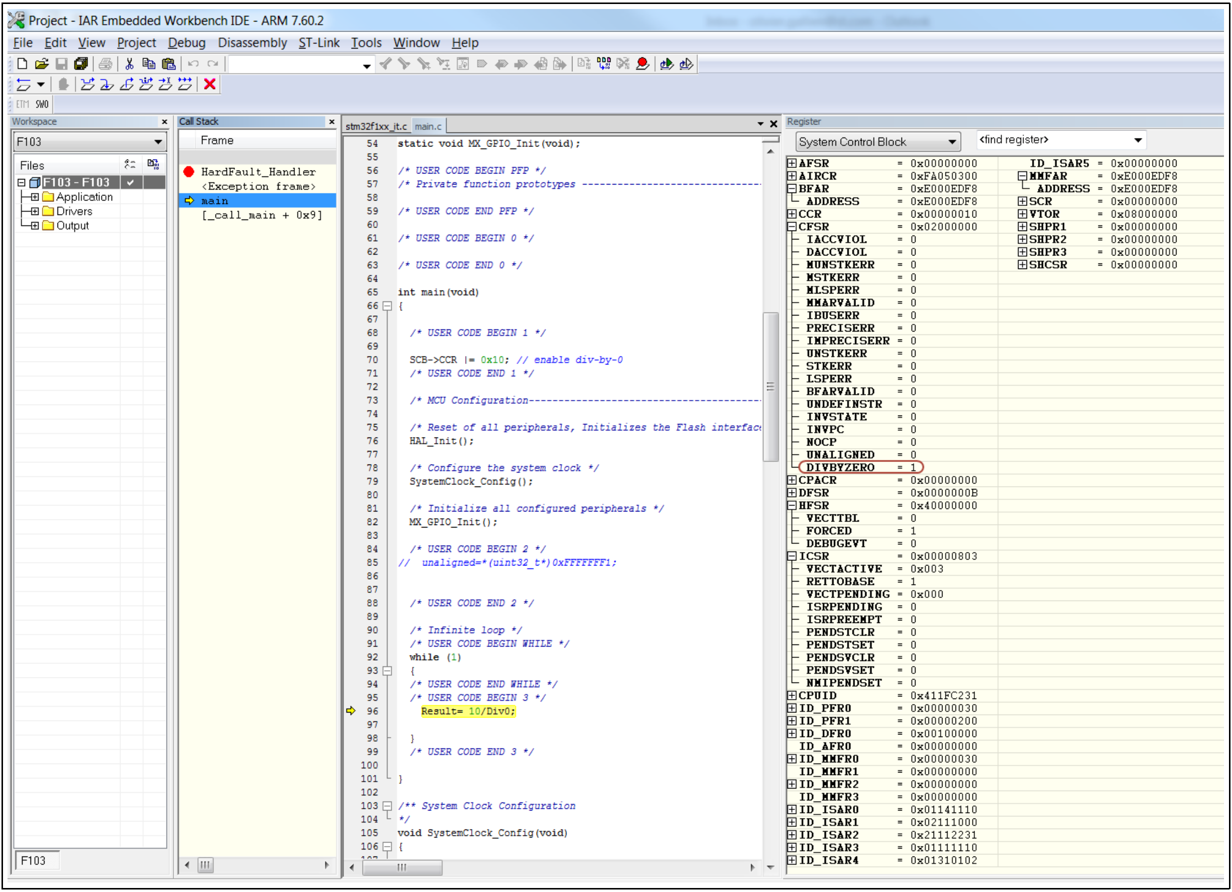


The following sections describe the management of SCB registers as a function of the selected IDE.

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 35*.

Figure 35. IAR™ exception handling



```
HardFault_Handler
<Exception frame>
main
[_call_main + 0x9]
```

```c
static void MX_GPIO_Init(void);

/* USER CODE BEGIN PFP */
/* Private function prototypes -----------------------------

/* USER CODE END PFP */

/* USER CODE BEGIN 0 */

/* USER CODE END 0 */

int main(void)
{

  /* USER CODE BEGIN 1 */

  SCB->CCR |= 0x10; // enable div-by-0
  /* USER CODE END 1 */

  /* MCU Configuration------------------------------------

  /* Reset of all peripherals, Initializes the Flash interfac
  HAL_Init();

  /* Configure the system clock */
  SystemClock_Config();

  /* Initialize all configured peripherals */
  MX_GPIO_Init();

  /* USER CODE BEGIN 2 */
//  unaligned=*(uint32_t*)0xFFFFFFF1;


  /* USER CODE END 2 */

  /* Infinite loop */
  /* USER CODE BEGIN WHILE */
  while (1)
  {
  /* USER CODE END WHILE */
  /* USER CODE BEGIN 3 */
    Result= 10/Div0;

  }
  /* USER CODE END 3 */

}

/** System Clock Configuration
*/
void SystemClock_Config(void)
{
```

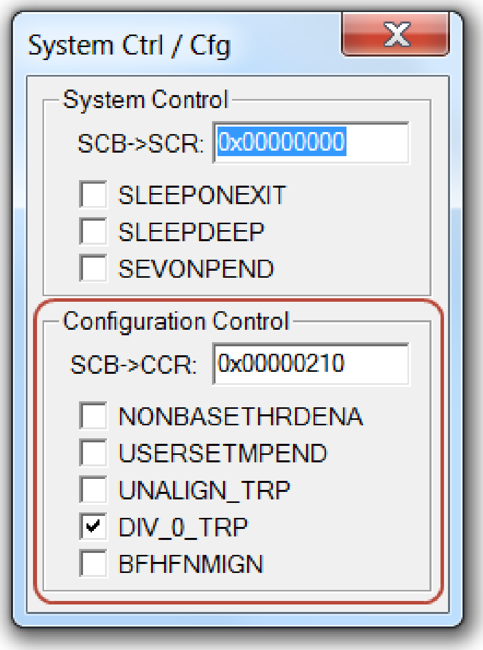
```
⊞AFSR            = 0x00000000        ID_ISAR5  = 0x00000000
⊞AIRCR           = 0xFA050300       ⊟MMFAR     = 0xE000EDF8
⊟BFAR            = 0xE000EDF8        └ ADDRESS = 0xE000EDF8
 └ ADDRESS       = 0xE000EDF8       ⊞SCR       = 0x00000000
⊞CCR             = 0x00000010       ⊞VTOR      = 0x08000000
⊟CFSR            = 0x02000000       ⊞SHPR1     = 0x00000000
 ├ IACCVIOL      = 0                ⊞SHPR2     = 0x00000000
 ├ DACCVIOL      = 0                ⊞SHPR3     = 0x00000000
 ├ MUNSTKERR     = 0                ⊞SHCSR     = 0x00000000
 ├ MSTKERR       = 0
 ├ MLSPERR       = 0
 ├ MMARVALID     = 0
 ├ IBUSERR       = 0
 ├ PRECISERR     = 0
 ├ IMPRECISERR   = 0
 ├ UNSTKERR      = 0
 ├ STKERR        = 0
 ├ LSPERR        = 0
 ├ BFARVALID     = 0
 ├ UNDEFINSTR    = 0
 ├ INVSTATE      = 0
 ├ INVPC         = 0
 ├ NOCP          = 0
 ├ UNALIGNED     = 0
 └ DIVBYZERO     = 1
⊞CPACR           = 0x00000000
⊞DFSR            = 0x0000000B
⊟HFSR            = 0x40000000
 ├ VECTTBL       = 0
 ├ FORCED        = 1
 └ DEBUGEVT      = 0
⊟ICSR            = 0x00000803
 ├ VECTACTIVE    = 0x003
 ├ RETTOBASE     = 1
 ├ VECTPENDING   = 0x000
 ├ ISRPENDING    = 0
 ├ ISRPREEMPT    = 0
 ├ PENDSTCLR     = 0
 ├ PENDSTSET     = 0
 ├ PENDSVCLR     = 0
 ├ PENDSVSET     = 0
 └ NMIPENDSET    = 0
⊞CPUID           = 0x411FC231
⊞ID_PFR0         = 0x00000030
⊞ID_PFR1         = 0x00000200
⊞ID_DFR0         = 0x00100000
 ID_AFR0         = 0x00000000
⊞ID_MMFR0        = 0x00000030
 ID_MMFR1        = 0x00000000
⊞ID_MMFR2        = 0x00000000
 ID_MMFR3        = 0x00000000
⊞ID_ISAR0        = 0x01141110
⊞ID_ISAR1        = 0x02111000
⊞ID_ISAR2        = 0x21112231
⊞ID_ISAR3        = 0x01111110
⊞ID_ISAR4        = 0x01310102
```

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 36* for details.

Figure 36. Keil® System Control and Configure



System Ctrl / Cfg

System Control

SCB->SCR: 0x00000000

☐ SLEEPONEXIT
☐ SLEEPDEEP
☐ SEVONPEND

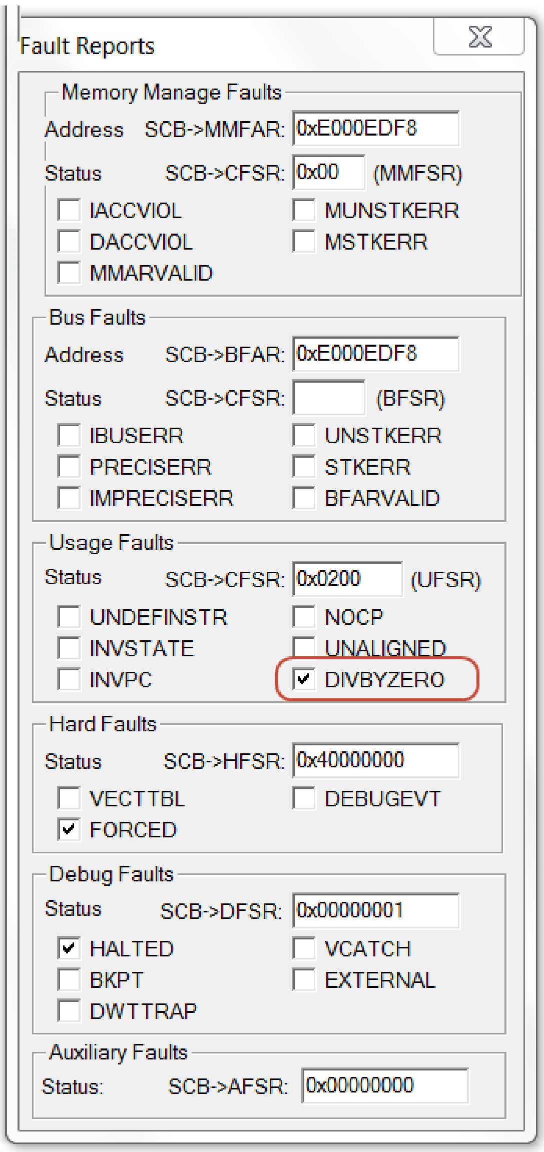
Configuration Control

SCB->CCR: 0x00000210

☐ NONBASETHRDENA
☐ USERSETMPEND
☐ UNALIGN_TRP
☑ DIV_0_TRP
☐ BFHFNMIGN

The fault type can be investigated using ***Peripherals -> Core Peripherals*** -> ***Fault Reports*** as shown in *Figure 37*.

Figure 37. Keil® Fault Reports



Fault Reports

Memory Manage Faults

Address SCB->MMFAR: 0xE000EDF8

Status SCB->CFSR: 0x00 (MMFSR)

- [ ] IACCVIOL
- [ ] MUNSTKERR
- [ ] DACCVIOL
- [ ] MSTKERR
- [ ] MMARVALID

Bus Faults

Address SCB->BFAR: 0xE000EDF8

Status SCB->CFSR: (BFSR)

- [ ] IBUSERR
- [ ] UNSTKERR
- [ ] PRECISERR
- [ ] STKERR
- [ ] IMPRECISERR
- [ ] BFARVALID

Usage Faults

Status SCB->CFSR: 0x0200 (UFSR)

- [ ] UNDEFINSTR
- [ ] NOCP
- [ ] INVSTATE
- [ ] UNALIGNED
- [ ] INVPC
- [✓] DIVBYZERO

Hard Faults

Status SCB->HFSR: 0x40000000

- [ ] VECTTBL
- [ ] DEBUGEVT
- [✓] FORCED

Debug Faults

Status SCB->DFSR: 0x00000001

- [✓] HALTED
- [ ] VCATCH
- [ ] BKPT
- [ ] EXTERNAL
- [ ] DWTTRAP

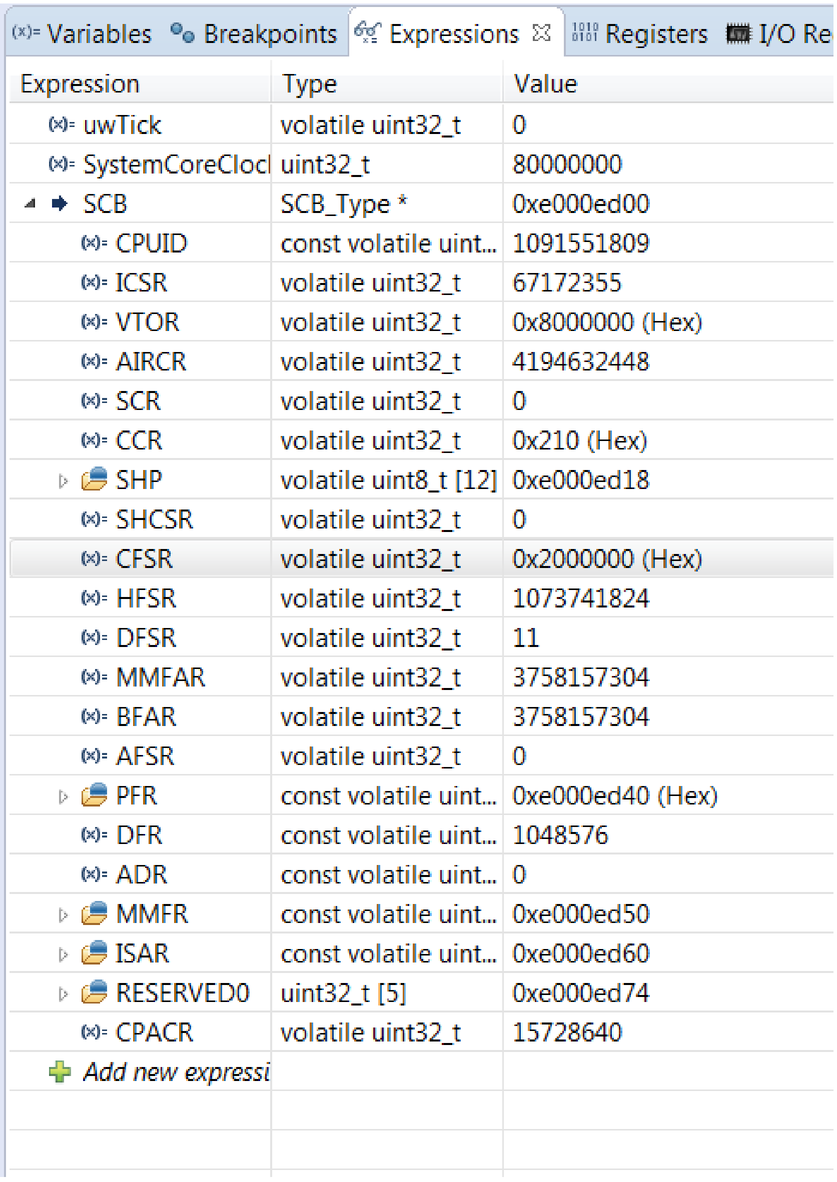
Auxiliary Faults

Status: SCB->AFSR: 0x00000000

SW4STM32

At runtime, while debug is in break state, the SCB register can be accessed in read mode or in write mode through: ***Expression Pane*** -> ***Add new expression*** as shown in *Figure 38*.

Figure 38. SW4STM32 SCB register access



Variables Breakpoints Expressions Registers I/O Re

Expression	Type	Value
uwTick	volatile uint32_t	0
SystemCoreClocl	uint32_t	80000000
SCB	SCB_Type *	0xe000ed00
CPUID	const volatile uint...	1091551809
ICSR	volatile uint32_t	67172355
VTOR	volatile uint32_t	0x8000000 (Hex)
AIRCR	volatile uint32_t	4194632448
SCR	volatile uint32_t	0
CCR	volatile uint32_t	0x210 (Hex)
SHP	volatile uint8_t [12]	0xe000ed18
SHCSR	volatile uint32_t	0
CFSR	volatile uint32_t	0x2000000 (Hex)
HFSR	volatile uint32_t	1073741824
DFSR	volatile uint32_t	11
MMFAR	volatile uint32_t	3758157304
BFAR	volatile uint32_t	3758157304
AFSR	volatile uint32_t	0
PFR	const volatile uint...	0xe000ed40 (Hex)
DFR	const volatile uint...	1048576
ADR	const volatile uint...	0
MMFR	const volatile uint...	0xe000ed50
ISAR	const volatile uint...	0xe000ed60
RESERVED0	uint32_t [5]	0xe000ed74
CPACR	volatile uint32_t	15728640
Add new expressi		

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:

- *STM32F10xxx/20xxx/21xxx/L1xxxx Cortex®-M3 programming manual* (PM0056)
- *STM32F3, STM32F4 and STM32L4 Series Cortex®-M4 programming manual* (PM0214)
- S*TM32F7 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.

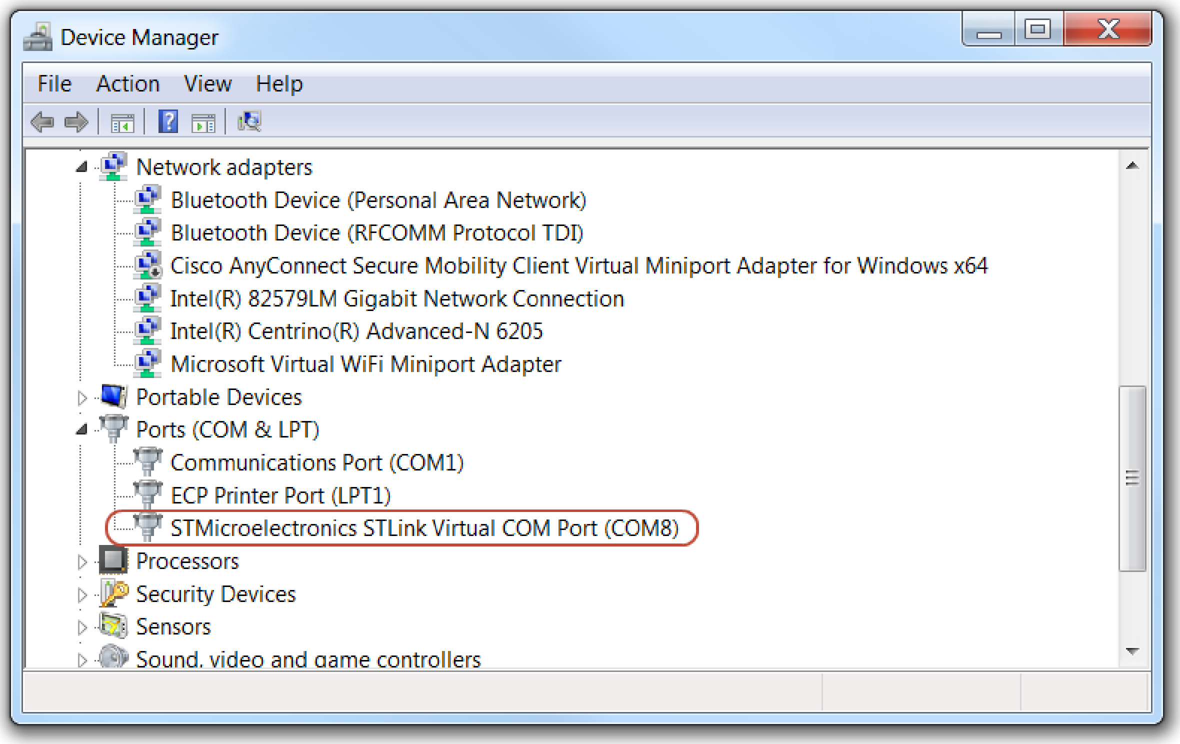
7.1 STM32 Virtual-COM port driver

STM32 Virtual-COM Port Driver (VCP) is a feature supported by ST-LINKV2-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 39*.

Figure 39. Virtual-COM port on Windows® PC



Network adapters
Bluetooth Device (Personal Area Network)
Bluetooth Device (RFCOMM Protocol TDI)
Cisco AnyConnect Secure Mobility Client Virtual Miniport Adapter for Windows x64
Intel(R) 82579LM Gigabit Network Connection
Intel(R) Centrino(R) Advanced-N 6205
Microsoft Virtual WiFi Miniport Adapter
Portable Devices
Ports (COM & LPT)
Communications Port (COM1)
ECP Printer Port (LPT1)
STMicroelectronics STLink Virtual COM Port (COM8)
Processors
Security Devices
Sensors
Sound, video and game controllers

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 "cutted" ST-LINK as stand-alone VCP on page 89* explains how to use ST-LINK Nucleo stand-alone part as VCP.

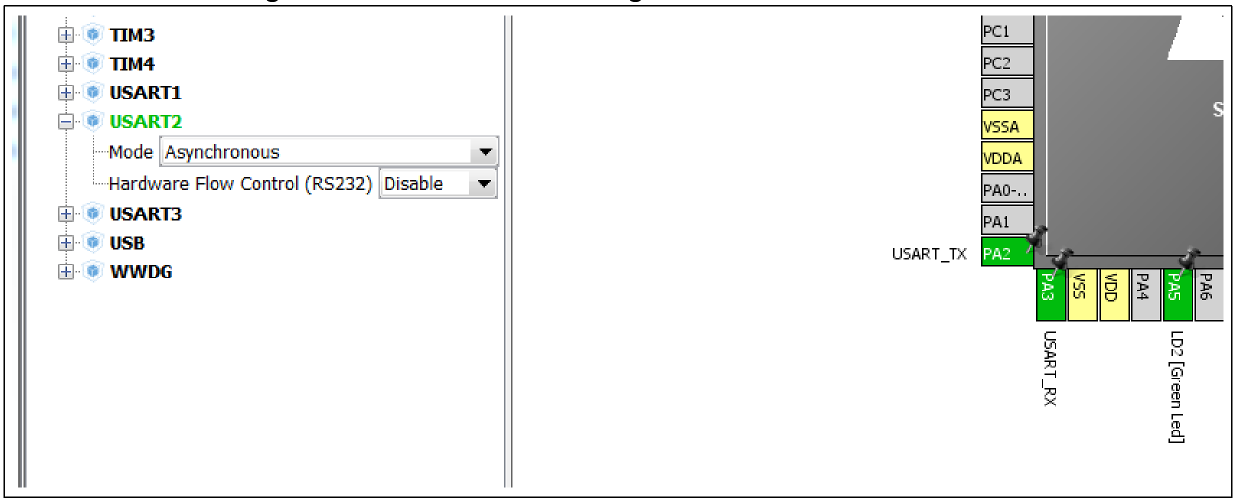
The straight-forward way to set a Serial Com port with PC host is to use the USART connected to VCP.

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

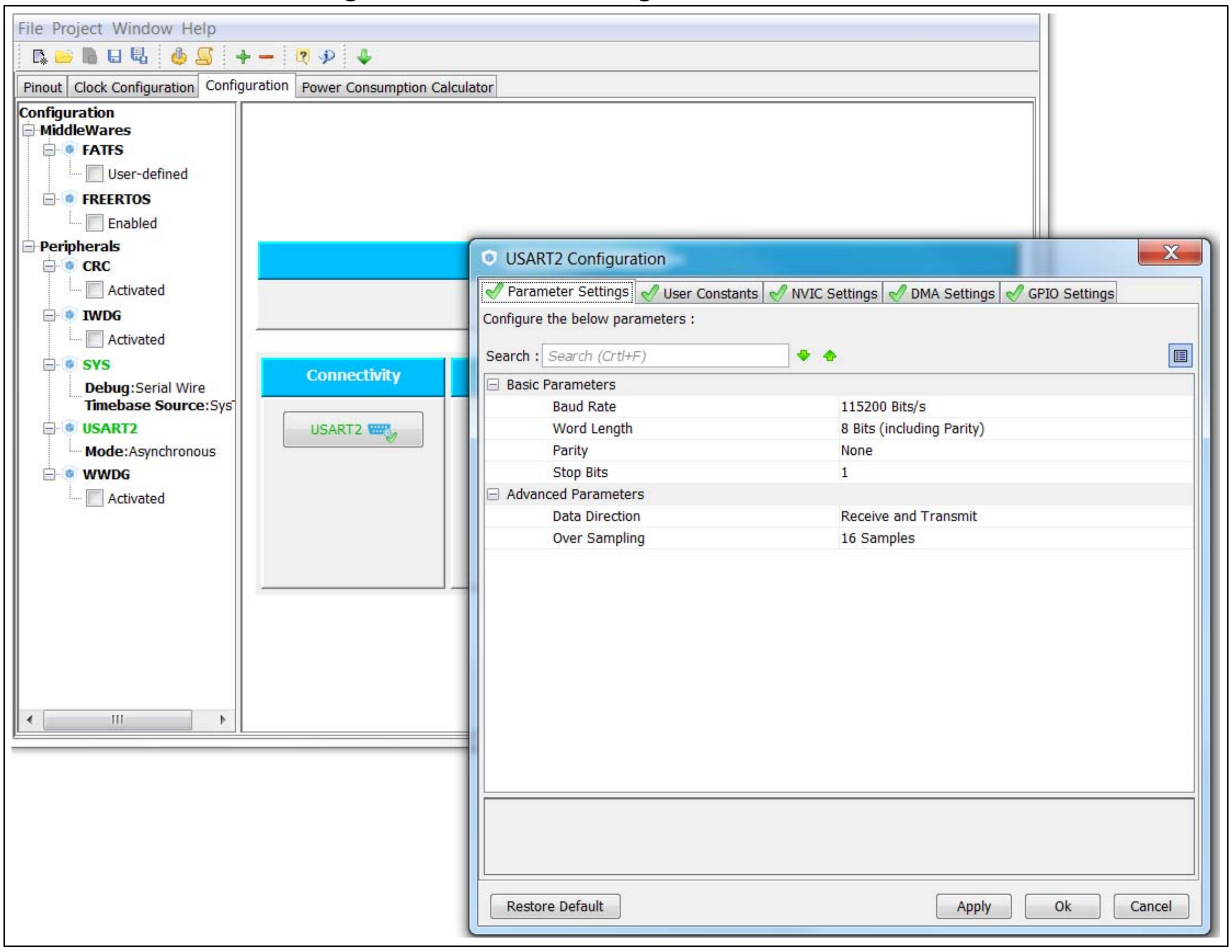
In STM32CubeMX, for Nucleo board, the VCP USART pins (PA2/PA3) are reserved by default, but required to be enabled by selecting "asynchronous" in USART mode selection box as shown in *Figure 40*.

Figure 40. USART Pinout configuration with STM32CubeMX


Mode: Asynchronous
Hardware Flow Control (RS232): Disable

Then, set the UART communication settings in ***Configuration*** -> ***USART2 Configuration*** -> ***Parameter Settings*** as shown in *Figure 41*.

Figure 41. USART2 setting with STM32CubeMX



Configuration
- MiddleWares
  - FATFS
    - User-defined
  - FREERTOS
    - Enabled
- Peripherals
  - CRC
    - Activated
  - IWDG
    - Activated
  - SYS
    - **Debug:**Serial Wire
    - **Timebase Source:**Sys
  - USART2
    - **Mode:**Asynchronous
  - WWDG
    - Activated

Configure the below parameters :

Basic Parameters	
Baud Rate	115200 Bits/s
Word Length	8 Bits (including Parity)
Parity	None
Stop Bits	1
Advanced Parameters	
Data Direction	Receive and Transmit
Over Sampling	16 Samples

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 SW4STM32, 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

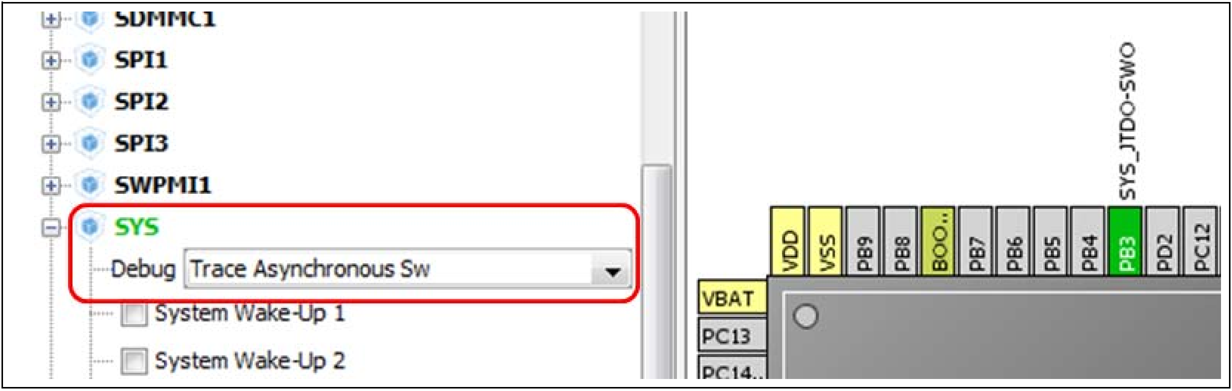
This chapter only addresses the printf-style debugging feature.

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 42*.

Figure 42. SWO Pin configuration with STM32CubeMX



Debug: Trace Asynchronous Sw

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

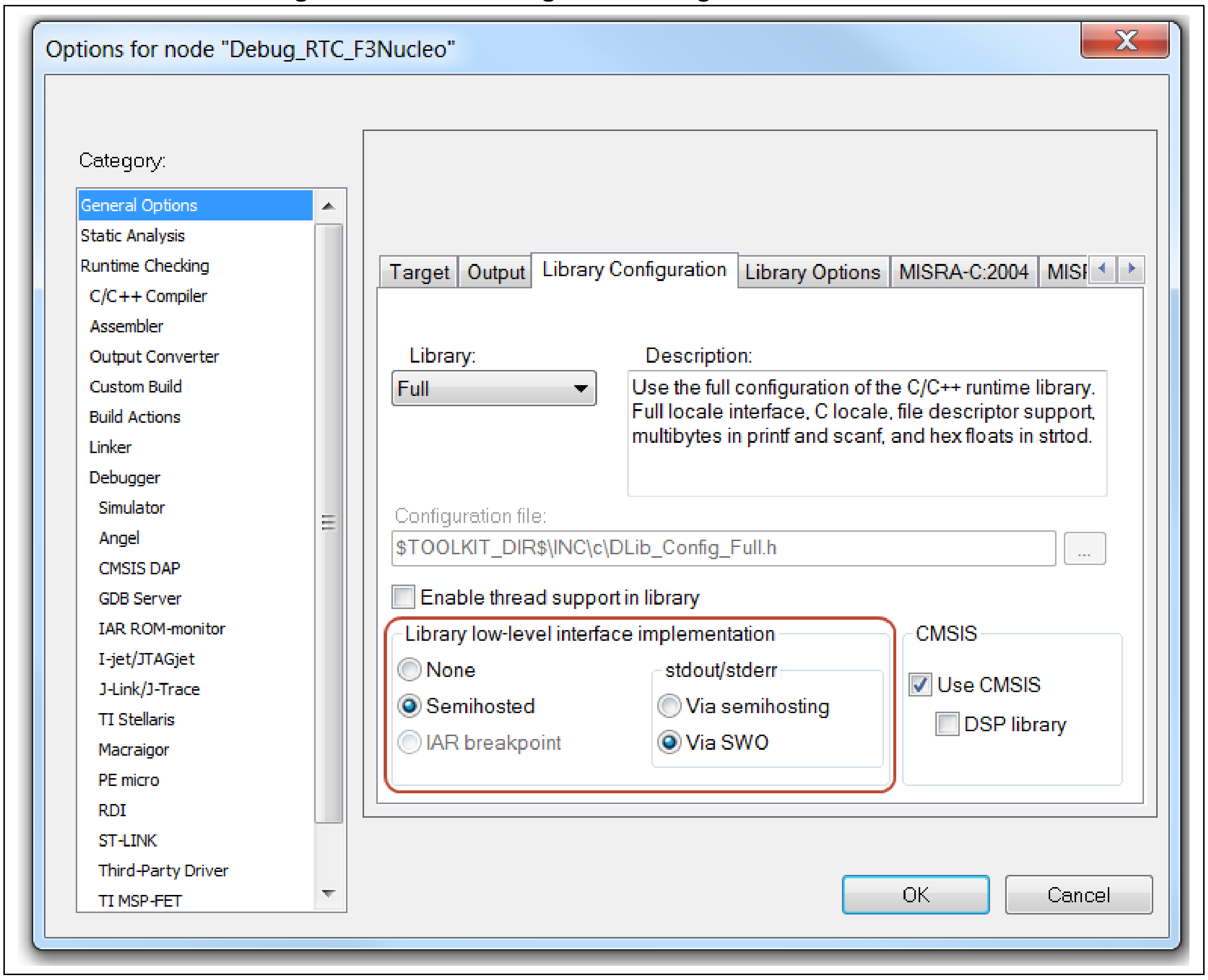
Other init steps are performed by the SWV integrated in the IDE or in the ST-LINK utility.

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 43*.

Figure 43. Semihosting/SWO configuration with IAR™



Library: Full

Description: Use the full configuration of the C/C++ runtime library. Full locale interface, C locale, file descriptor support, multibytes in printf and scanf, and hex floats in strtod.

Configuration file: $TOOLKIT_DIR$\INC\c\DLib_Config_Full.h

☐ Enable thread support in library

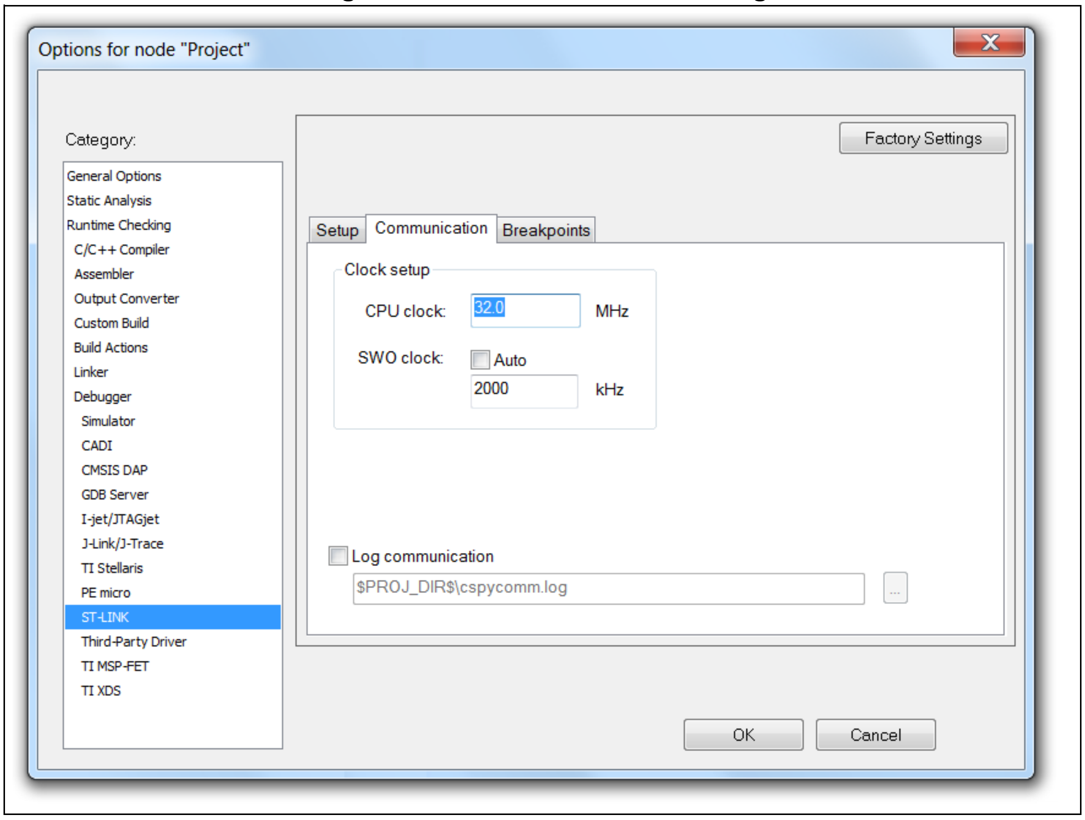
Library low-level interface implementation: None [✓ Semihosted] IAR breakpoint

stdout/stderr: Via semihosting [✓ Via SWO]

CMSIS: ☑ Use CMSIS ☐ DSP library

Care must be taken that clock setup is correct by using ***ST-LINK*** -> ***Communication Pane*** as illustrated in *Figure 44*.

Figure 44. IAR™ SWO Clock setting



Clock setup

CPU clock: 32.0 MHz

SWO clock: Auto [ ] 2000 kHz

Log communication [ ] $PROJ_DIR$\cspycomm.log

Once configured, IAR™ 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 59*)

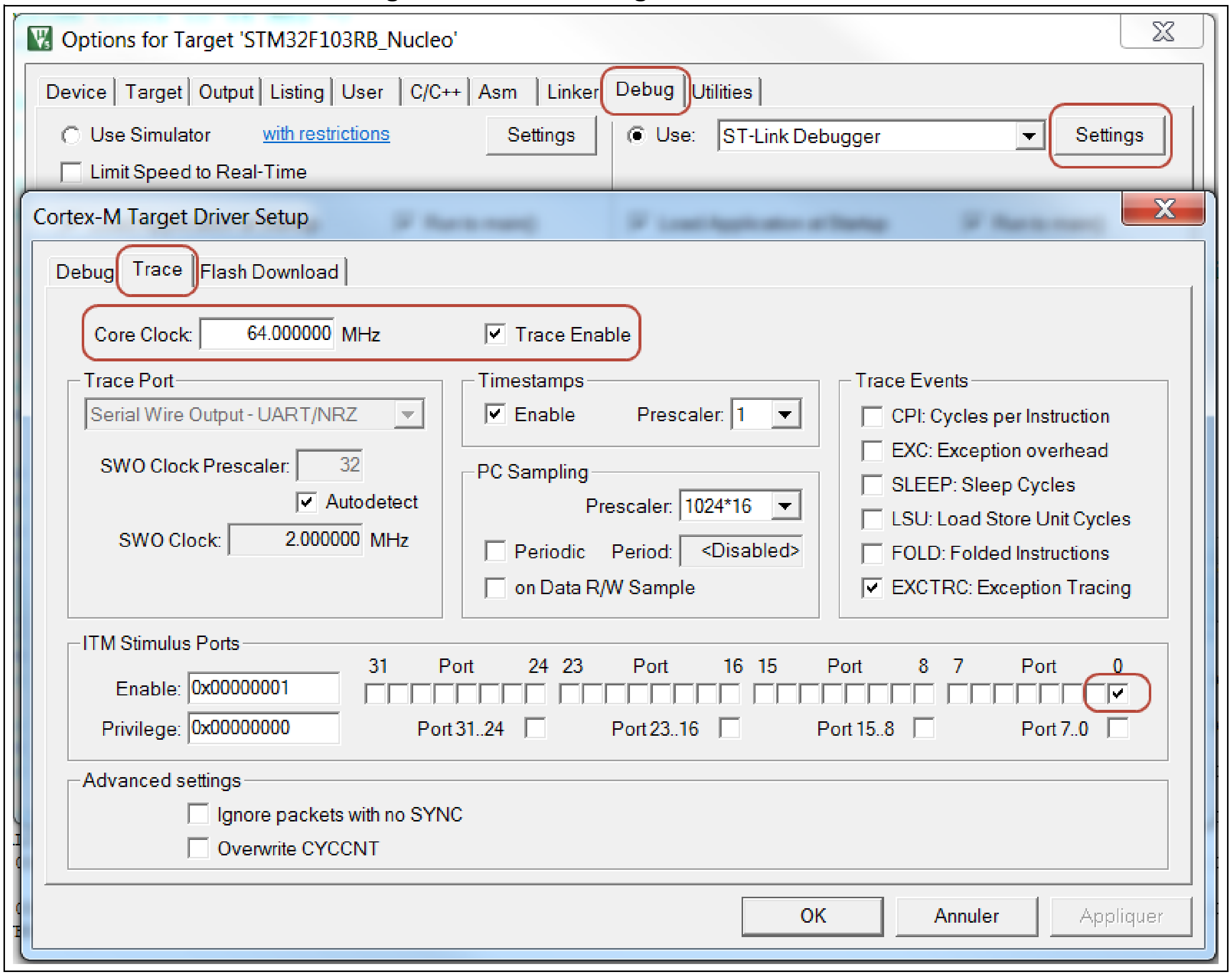
```
#include "stdio.h"

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 45*.

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

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

Figure 45. SWO Configuration with Keil®


Options for Target 'STM32F103RB_Nucleo'

Device | Target | Output | Listing | User | C/C++ | Asm | Linker | Debug | Utilities

Use Simulator with restrictions Settings ● Use: ST-Link Debugger Settings

Limit Speed to Real-Time

Cortex-M Target Driver Setup

Debug | Trace | Flash Download

Core Clock: 64.000000 MHz ☑ Trace Enable

Trace Port: Serial Wire Output - UART/NRZ

SWO Clock Prescaler: 32 ☑ Autodetect

SWO Clock: 2.000000 MHz

Timestamps: ☑ Enable Prescaler: 1

PC Sampling: Prescaler: 1024*16

☐ Periodic Period: <Disabled>

☐ on Data R/W Sample

Trace Events:

☐ CPI: Cycles per Instruction

☐ EXC: Exception overhead

☐ SLEEP: Sleep Cycles

☐ LSU: Load Store Unit Cycles

☐ FOLD: Folded Instructions

☑ EXCTRC: Exception Tracing

ITM Stimulus Ports

Enable: 0x00000001 (Port 0 ☑)

Privilege: 0x00000000

Port 31..24 ☐ Port 23..16 ☐ Port 15..8 ☐ Port 7..0 ☐

Advanced settings

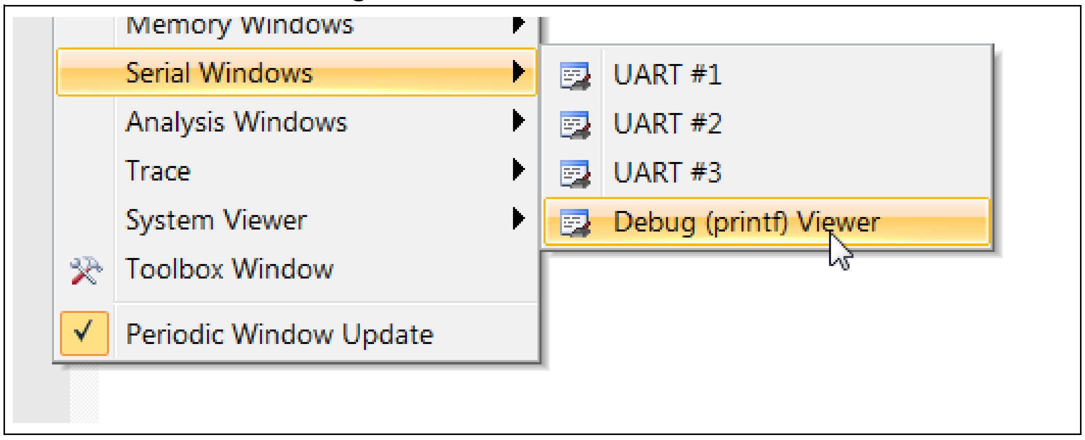
☐ Ignore packets with no SYNC

☐ Overwrite CYCCNT

OK Annuler Appliquer

SWV viewer is called "Debug (printf) Viewer" and is accessible while in debug through ***View*** -> ***Serial Windows*** -> ***Debug (prinf) Viewer*** as shown in *Figure 46*.

Figure 46. Access to SWV in Keil®



SW4STM32 (and all GCC based toolset)

With syscall.c integrated to the project:

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

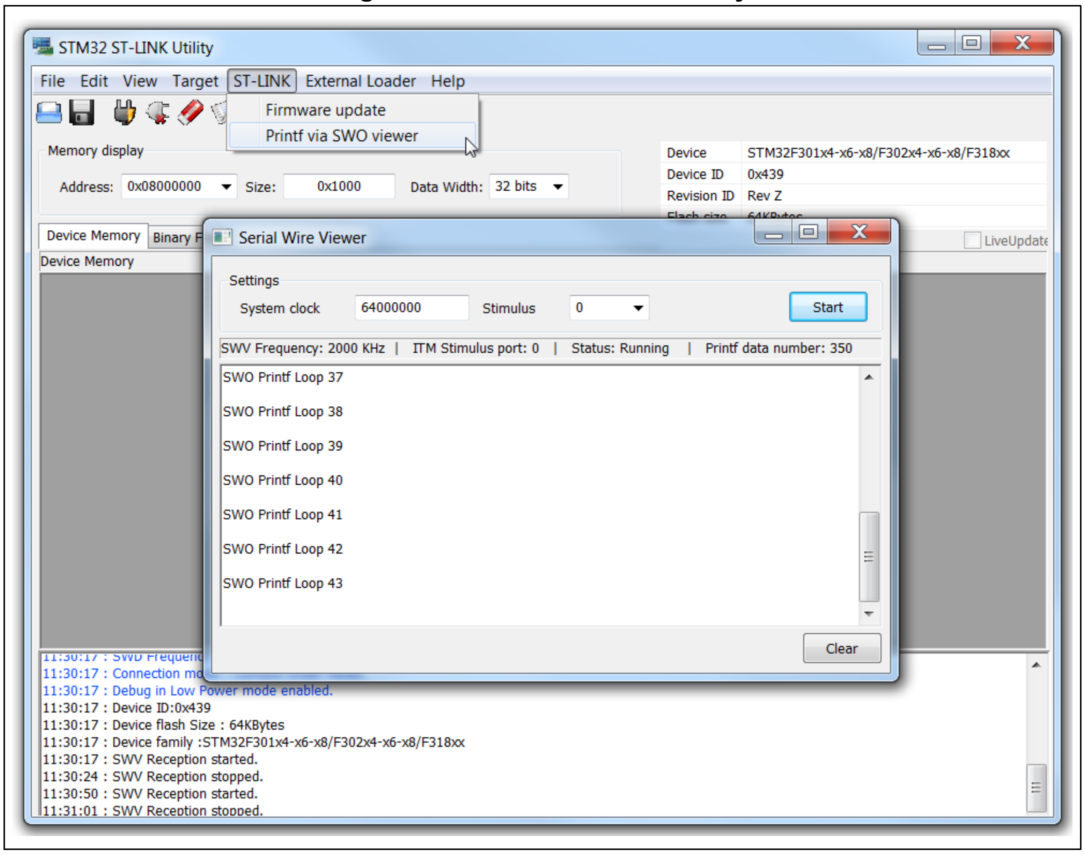
Without syscall, add:

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

ST-LINK utility

In case the IDE does not have embedded SWV, the one provided by ST-LINK Utilities can be used instead. *Figure 47* shows the use of SWV in ST-LINK Utilities.

Figure 47. SWV in ST-LINK utility



Address: 0x08000000 Size: 0x1000 Data Width: 32 bits

Device STM32F301x4-x6-x8/F302x4-x6-x8/F318xx
Device ID 0x439
Revision ID Rev Z

System clock 64000000 Stimulus 0

SWV Frequency: 2000 KHz | ITM Stimulus port: 0 | Status: Running | Printf data number: 350

```
SWO Printf Loop 37
SWO Printf Loop 38
SWO Printf Loop 39
SWO Printf Loop 40
SWO Printf Loop 41
SWO Printf Loop 42
SWO Printf Loop 43
```

```
11:30:17 : SWD Frequenc
11:30:17 : Connection mo
11:30:17 : Debug in Low Power mode enabled.
11:30:17 : Device ID:0x439
11:30:17 : Device flash Size : 64KBytes
11:30:17 : Device family :STM32F301x4-x6-x8/F302x4-x6-x8/F318xx
11:30:17 : SWV Reception started.
11:30:24 : SWV Reception stopped.
11:30:50 : SWV Reception started.
11:31:01 : SWV Reception stopped.
```

Refer to *STM32 ST-LINK utility software description user manual* (UM0892) for details.

Tip: Keil® MDK-ARM µVISION and ST-LINK utility SWV allows 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.

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.

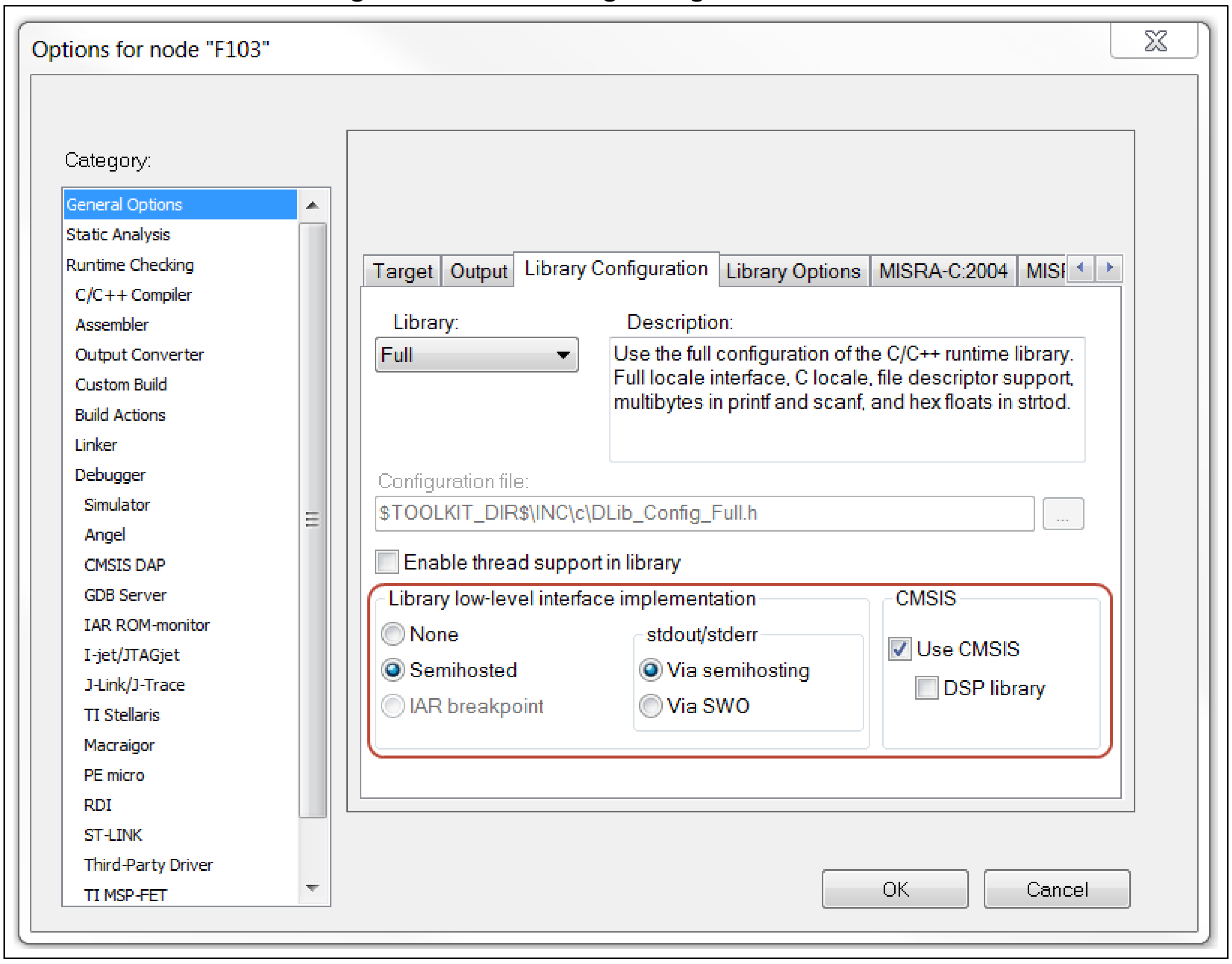
Semihosting depends on the library provided by the IDE. The next sections present how to set semihosting using the three main IDEs covered in this application note.

7.4.1 IAR™ EWARM

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

Figure 48 shows how to check if it is the case for the project in ***Options -> General options -> Library Configuration Pane.***

Figure 48. Semihosting configuration in IAR™


Options for node "F103"

Category: General Options

Library Configuration

Library: Full

Description: Use the full configuration of the C/C++ runtime library. Full locale interface, C locale, file descriptor support, multibytes in printf and scanf, and hex floats in strtod.

Configuration file: $TOOLKIT_DIR$\INC\c\DLib_Config_Full.h

☐ Enable thread support in library

Library low-level interface implementation: None [✓ Semihosted] IAR breakpoint

stdout/stderr: [✓ Via semihosting] Via SWO

CMSIS: ☑ Use CMSIS ☐ DSP library

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.

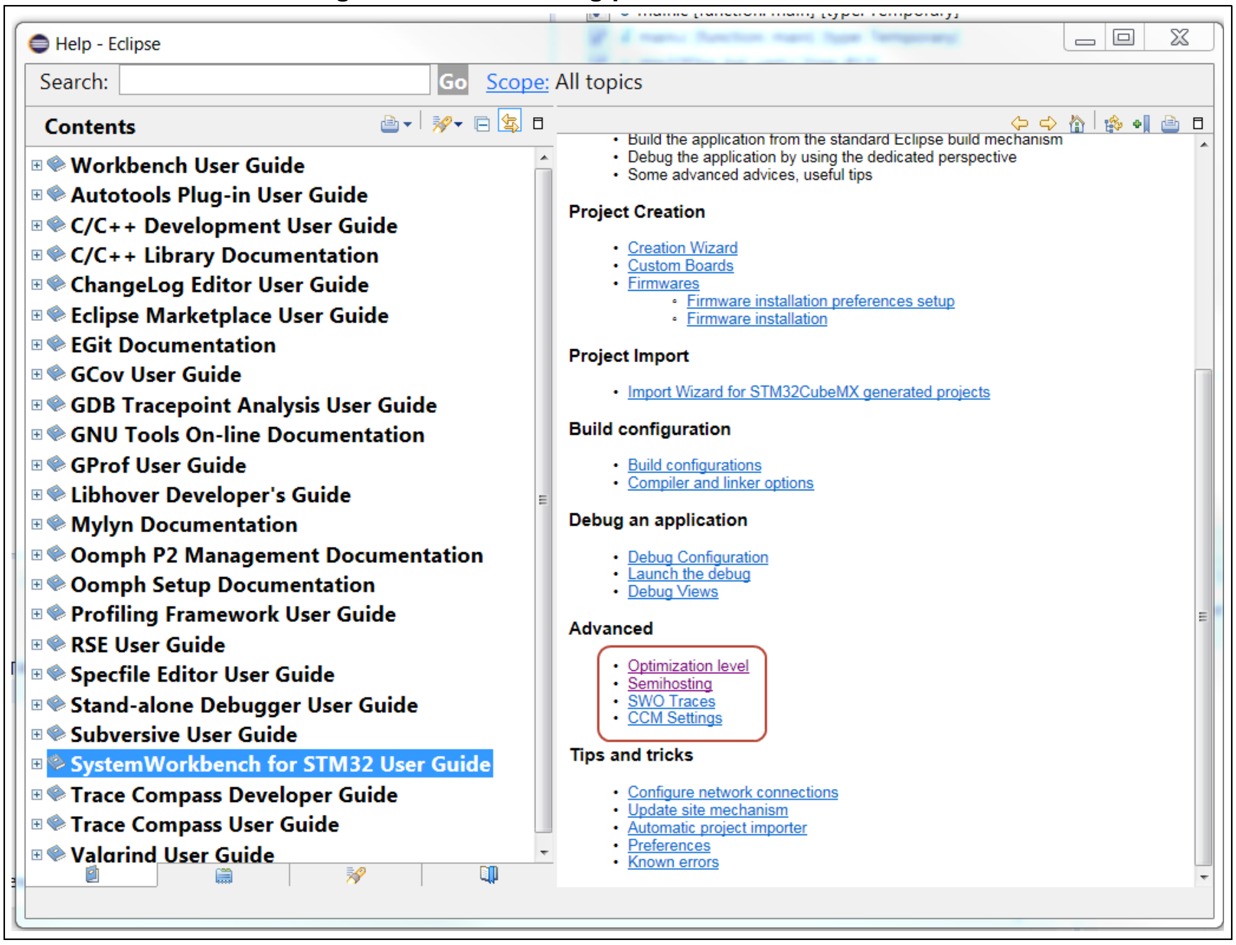
7.4.2 Keil® MDK-ARM µVISION

Keil® has no semihosting capability.

7.4.3 SW4STM32

From OpenSTM32 IDE version 1.12.0 and upper, a tutorial to set semihosting is accessible in the local SW4STM32 installation as shown in *Figure 49*. Follow ***Help*** -> ***Help Content*** -> ***SystemWorkbench*** for STM32 User Guide.

Figure 49. Semihosting procedure in SW4STM32



Help - Eclipse

Search: Go Scope: All topics

Contents

- Workbench User Guide
- Autotools Plug-in User Guide
- C/C++ Development User Guide
- C/C++ Library Documentation
- ChangeLog Editor User Guide
- Eclipse Marketplace User Guide
- EGit Documentation
- GCov User Guide
- GDB Tracepoint Analysis User Guide
- GNU Tools On-line Documentation
- GProf User Guide
- Libhover Developer's Guide
- Mylyn Documentation
- Oomph P2 Management Documentation
- Oomph Setup Documentation
- Profiling Framework User Guide
- RSE User Guide
- Specfile Editor User Guide
- Stand-alone Debugger User Guide
- Subversive User Guide
- SystemWorkbench for STM32 User Guide
- Trace Compass Developer Guide
- Trace Compass User Guide
- Valgrind User Guide

- Build the application from the standard Eclipse build mechanism
- Debug the application by using the dedicated perspective
- Some advanced advices, useful tips

Project Creation

- Creation Wizard
- Custom Boards
- Firmwares
  - Firmware installation preferences setup
  - Firmware installation

Project Import

- Import Wizard for STM32CubeMX generated projects

Build configuration

- Build configurations
- Compiler and linker options

Debug an application

- Debug Configuration
- Launch the debug
- Debug Views

Advanced

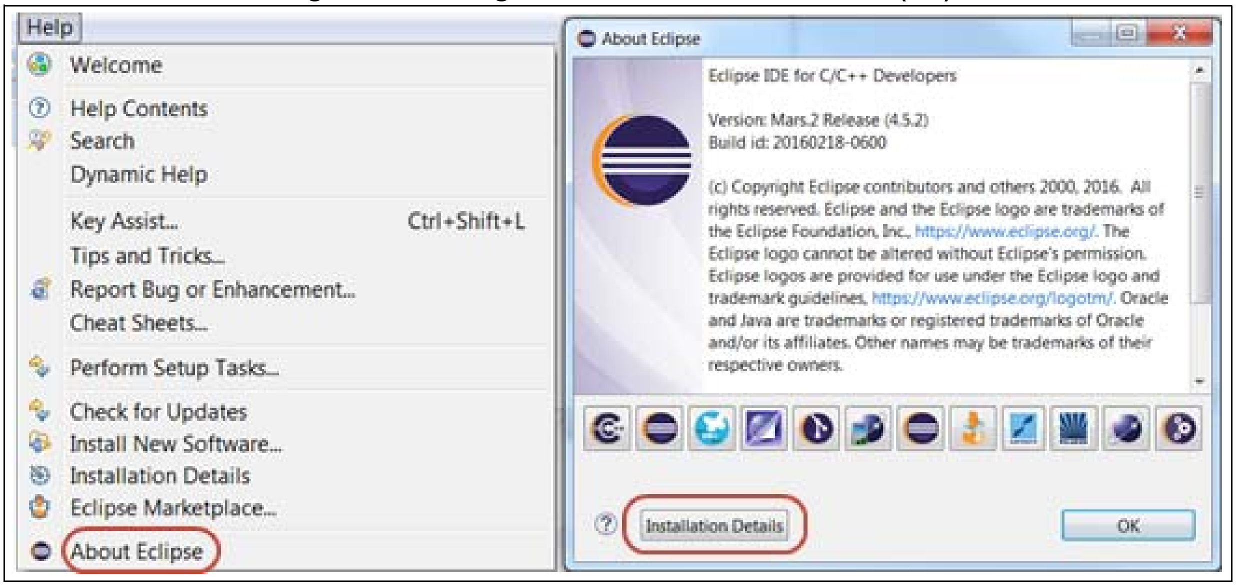
- Optimization level
- Semihosting
- SWO Traces
- CCM Settings

Tips and tricks

- Configure network connections
- Update site mechanism
- Automatic project importer
- Preferences
- Known errors

The local STM32 plugins versions are obtained through ***Help*** -> ***About Eclipse*** as shown in *Figure 50* and in *Figure 51*.

Figure 50. Getting SW4STM32 reference version (1/2)

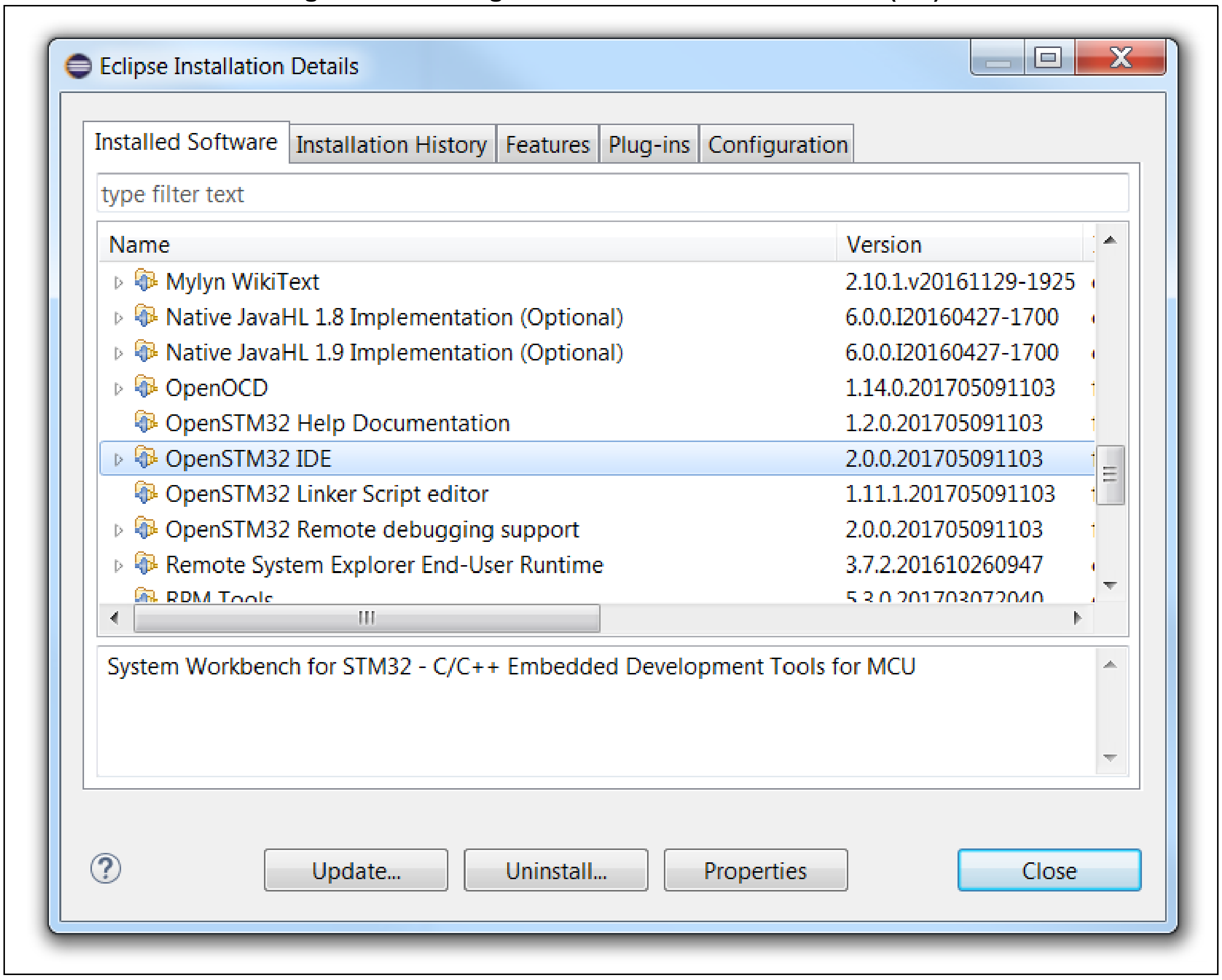


Eclipse IDE for C/C++ Developers

Version: Mars.2 Release (4.5.2)
Build id: 20160218-0600

(c) Copyright Eclipse contributors and others 2000, 2016. All rights reserved. Eclipse and the Eclipse logo are trademarks of the Eclipse Foundation, Inc., https://www.eclipse.org/. The Eclipse logo cannot be altered without Eclipse's permission. Eclipse logos are provided for use under the Eclipse logo and trademark guidelines, https://www.eclipse.org/logotm/. Oracle and Java are trademarks or registered trademarks of Oracle and/or its affiliates. Other names may be trademarks of their respective owners.

Figure 51. Getting SW4STM32 reference version (2/2)



Name	Version
Mylyn WikiText	2.10.1.v20161129-1925
Native JavaHL 1.8 Implementation (Optional)	6.0.0.I20160427-1700
Native JavaHL 1.9 Implementation (Optional)	6.0.0.I20160427-1700
OpenOCD	1.14.0.201705091103
OpenSTM32 Help Documentation	1.2.0.201705091103
OpenSTM32 IDE	2.0.0.201705091103
OpenSTM32 Linker Script editor	1.11.1.201705091103
OpenSTM32 Remote debugging support	2.0.0.201705091103
Remote System Explorer End-User Runtime	3.7.2.201610260947
RPM Tools	5.3.0.201703072040

System Workbench for STM32 - C/C++ Embedded Development Tools for MCU

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)

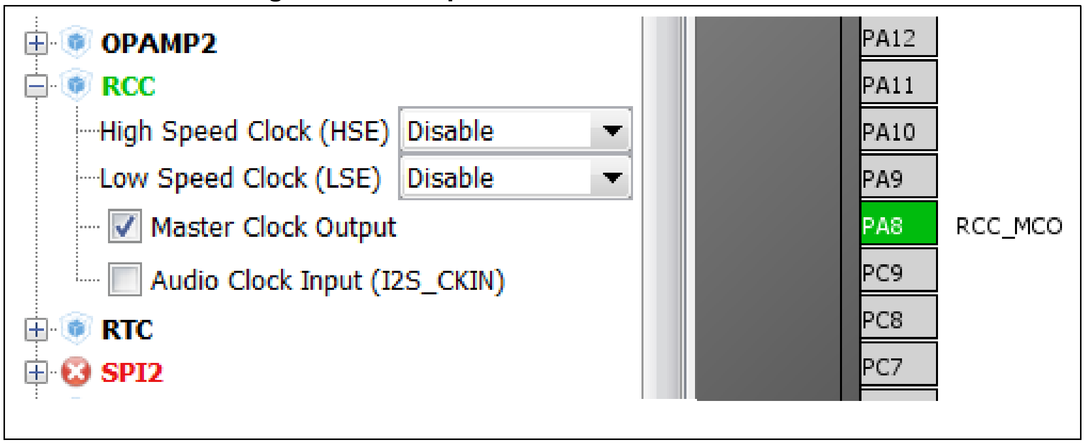
This feature allows to output one or more internal clock to one or more pins in order to enable measurement through an external tool, typically an oscilloscope.

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 52*.

Figure 52. MCO pin selection in STM32CubeMX



High Speed Clock (HSE) Disable
Low Speed Clock (LSE) Disable
✓ Master Clock Output
Audio Clock Input (I2S_CKIN)
PA8 RCC_MCO

This allocates a pin labeled RCC_MCO.

This is typically pin PA8 for all STM32 families.

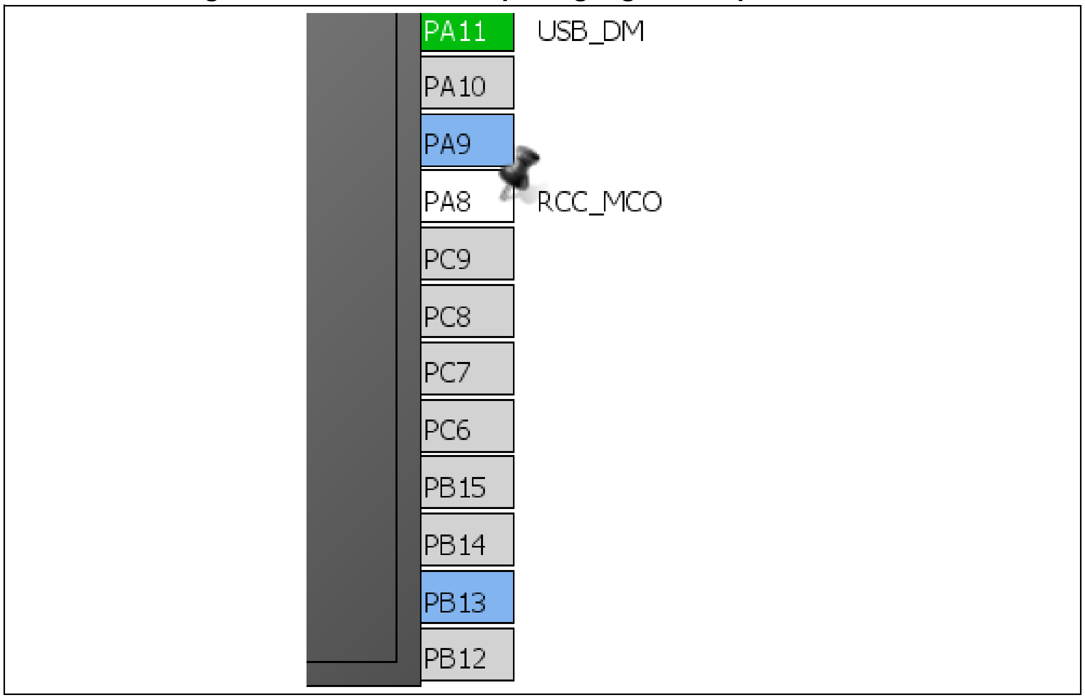
For Nucleo kits, the PA8 pin is accessible on the D7 pin of the Arduino™ connector.

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.

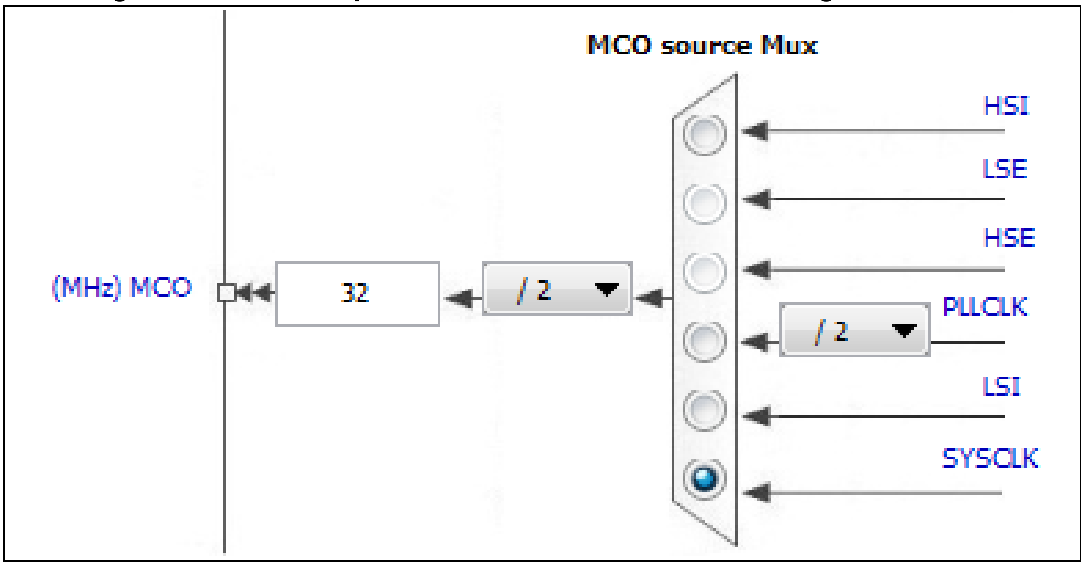
The *Ctrl* + *click* on RCC_MCO pin command sequence under STM32CubeMX highlights in blue the alternate pin. An example is shown in *Figure 53*.

Figure 53. 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 54*.

Figure 54. 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

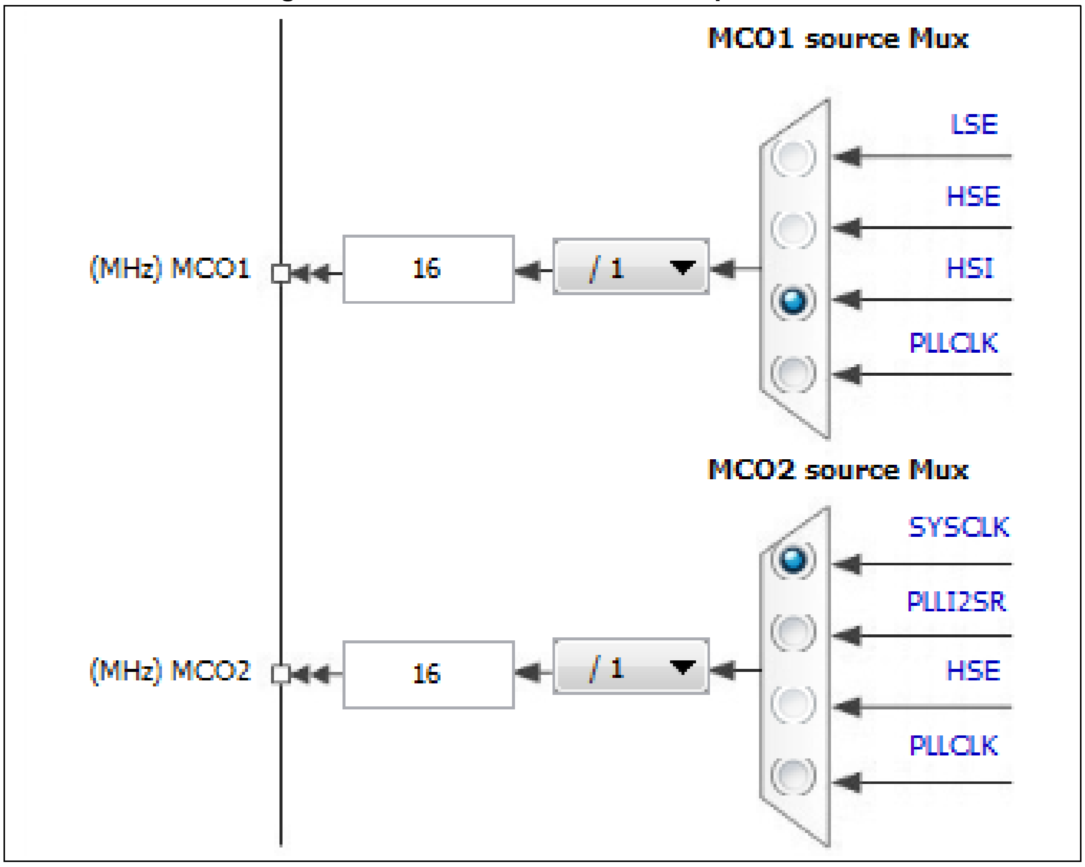
Note: LSCO is conflicting with UART2 TX (PA2). On Nucleo-64 board, the use of the LSCO board use and the use of the ST-LINK VCP are mutually exclusive.

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 55*. Refer also to *Appendix D: Cortex®-M debug capabilities reminder on page 97*.

Figure 55. STM32F4/F7 dual MCO capabilities



9 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.

10 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

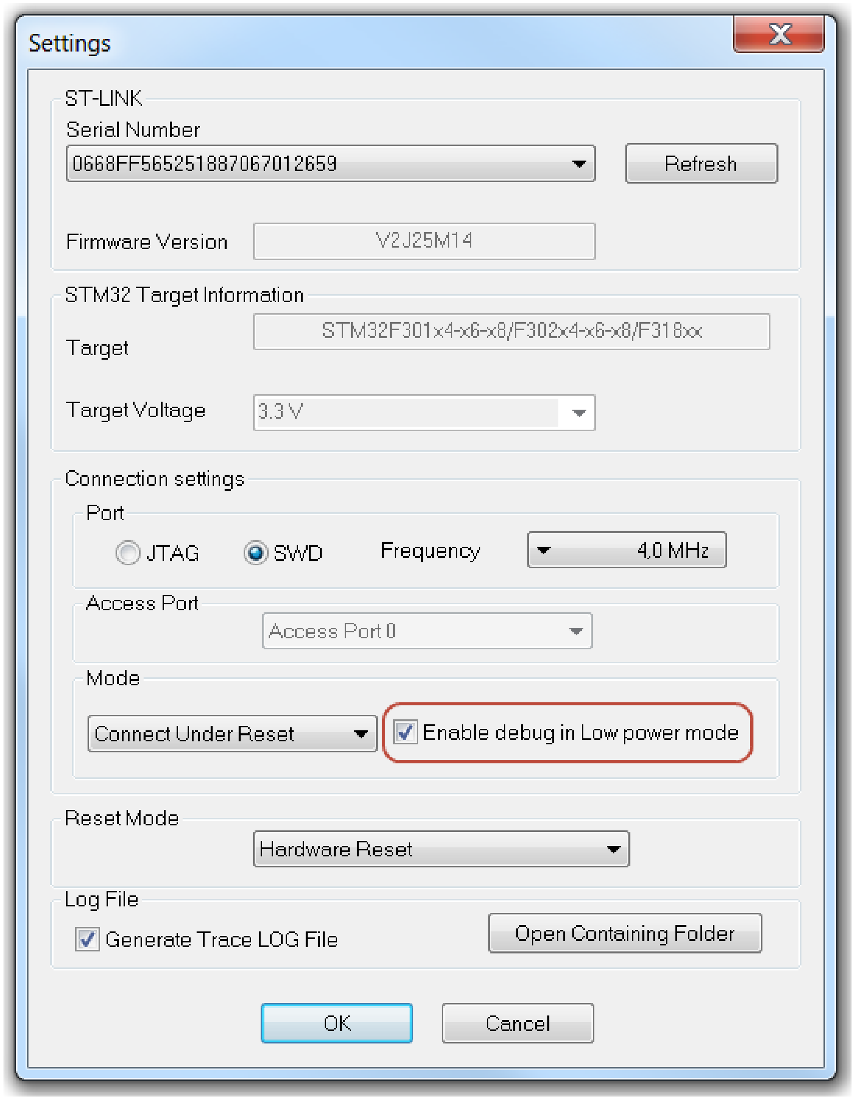
This appendix provides a tutorial for the different ways to Read/Write the DBGMCU registers with various tools and IDEs.

A.1 By the ST-LINK utility

DBMCU registers states are preserved in case of reset. (hard or soft).

This allow to set this register thanks to ST-LINK utility independently from the IDE as in *Figure 56*.

Figure 56. ST-LINK utility Enable Low-Power debug option



Settings

ST-LINK
Serial Number: 0668FF565251887067012659
Refresh
Firmware Version: V2J25M14

STM32 Target Information
Target: STM32F301x4-x6-x8/F302x4-x6-x8/F318xx
Target Voltage: 3.3 V

Connection settings
Port: ( ) JTAG [● SWD] Frequency: 4.0 MHz
Access Port: Access Port 0
Mode: Connect Under Reset [✓] Enable debug in Low power mode

Reset Mode: Hardware Reset

Log File
[✓] Generate Trace LOG File
Open Containing Folder

OK Cancel

ST-LINK provides a tick box allowing to set all the three bits of DBMCU_CR register at once: DBG_STANDBY, DBG_STOP, and DBG_SLEEP

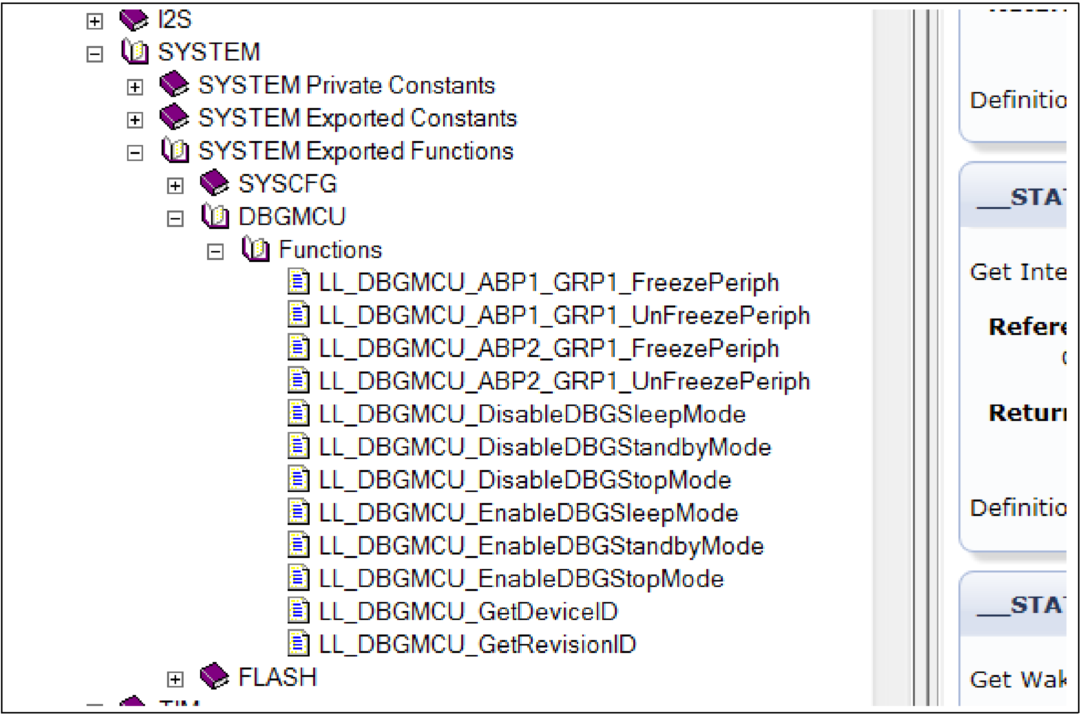
A.2 By software

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

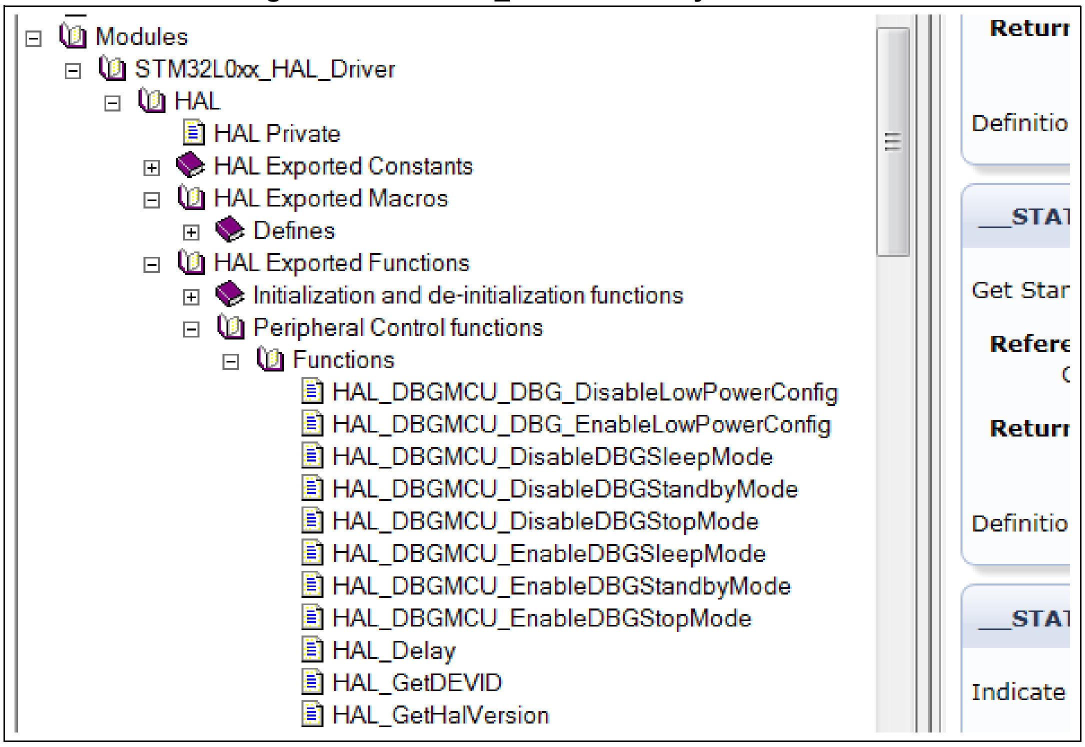
Figure 57 and *Figure 58* show the positions of the DBGMCU registers iwithin the LL and HAL libraries.

Figure 57. DBMCU Register LL Library Functions



```
⊞ I2S
⊟ SYSTEM
    ⊞ SYSTEM Private Constants
    ⊞ SYSTEM Exported Constants
    ⊟ SYSTEM Exported Functions
        ⊞ SYSCFG
        ⊟ DBGMCU
            ⊟ Functions
                LL_DBGMCU_ABP1_GRP1_FreezePeriph
                LL_DBGMCU_ABP1_GRP1_UnFreezePeriph
                LL_DBGMCU_ABP2_GRP1_FreezePeriph
                LL_DBGMCU_ABP2_GRP1_UnFreezePeriph
                LL_DBGMCU_DisableDBGSleepMode
                LL_DBGMCU_DisableDBGStandbyMode
                LL_DBGMCU_DisableDBGStopMode
                LL_DBGMCU_EnableDBGSleepMode
                LL_DBGMCU_EnableDBGStandbyMode
                LL_DBGMCU_EnableDBGStopMode
                LL_DBGMCU_GetDeviceID
                LL_DBGMCU_GetRevisionID
        ⊞ FLASH
```

Figure 58. DBGMCU_CR HAL Library Functions


Modules
STM32L0xx_HAL_Driver
HAL
HAL Private
HAL Exported Constants
HAL Exported Macros
Defines
HAL Exported Functions
Initialization and de-initialization functions
Peripheral Control functions
Functions
HAL_DBGMCU_DBG_DisableLowPowerConfig
HAL_DBGMCU_DBG_EnableLowPowerConfig
HAL_DBGMCU_DisableDBGSleepMode
HAL_DBGMCU_DisableDBGStandbyMode
HAL_DBGMCU_DisableDBGStopMode
HAL_DBGMCU_EnableDBGSleepMode
HAL_DBGMCU_EnableDBGStandbyMode
HAL_DBGMCU_EnableDBGStopMode
HAL_Delay
HAL_GetDEVID
HAL_GetHalVersion
Returr
Definitio
__STAT
Get Star
Refere
(
Returr
Definitio
__STAT
Indicate

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();
```

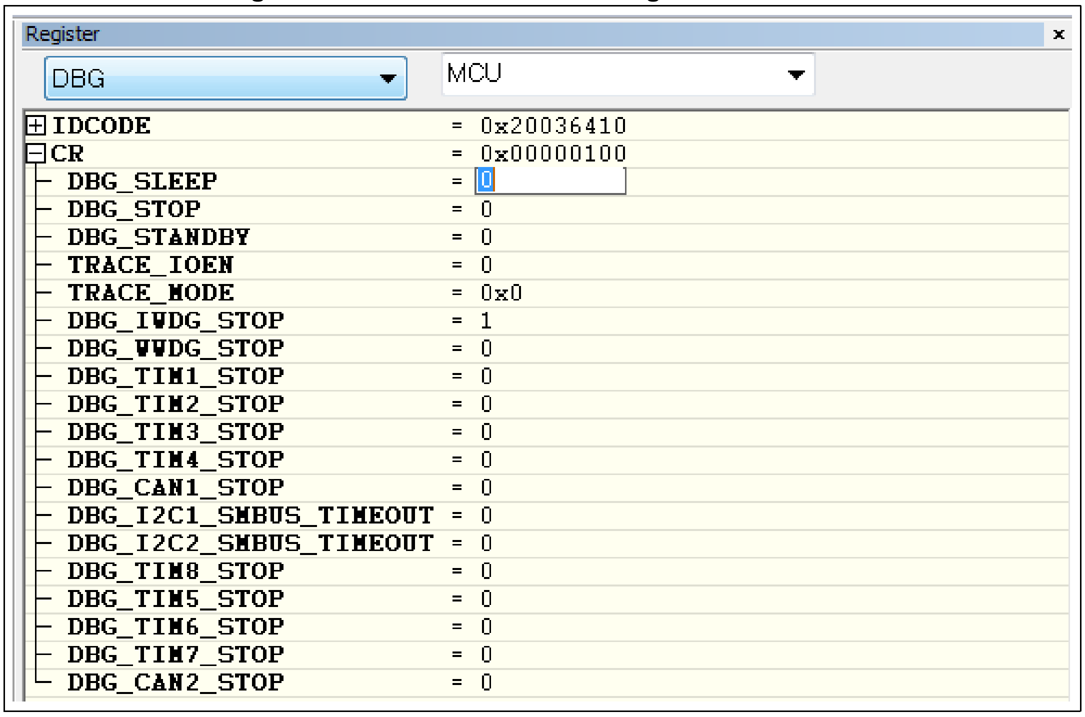
A.3 By debugger

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 59*:

Figure 59. Access to DBGMCU register with IAR™

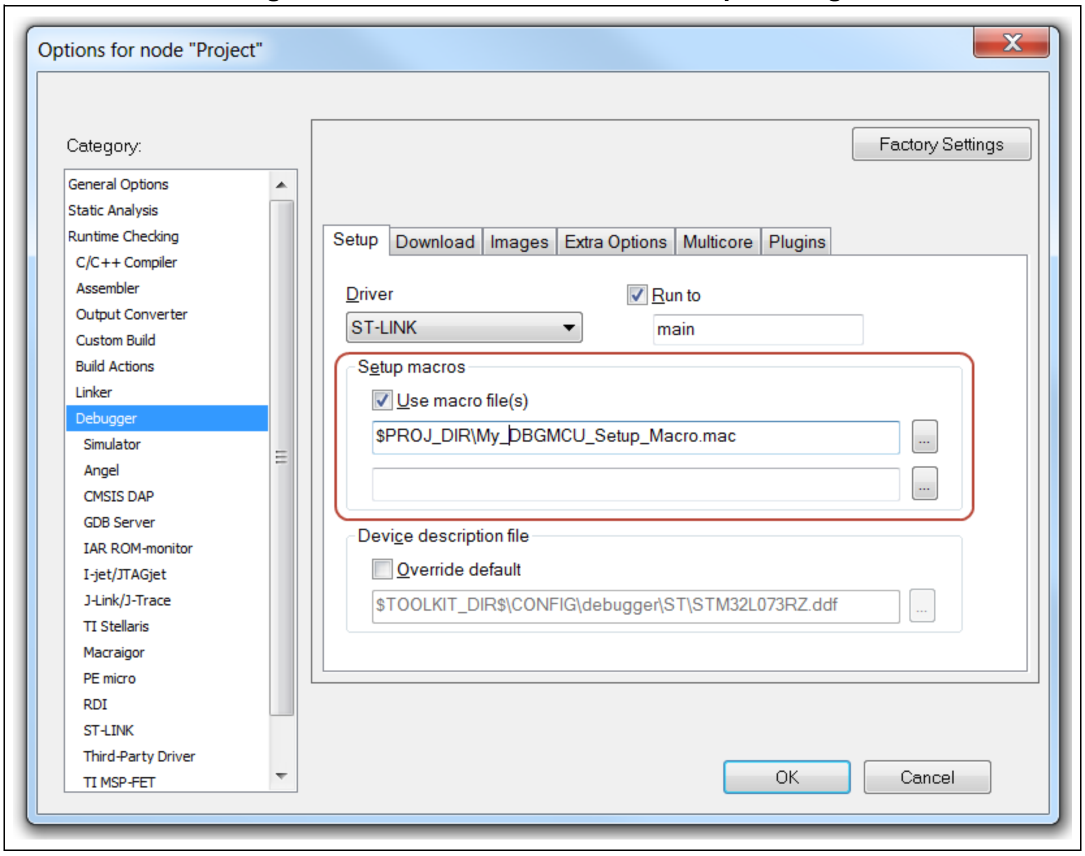


```
⊞IDCODE                      = 0x20036410
⊟CR                          = 0x00000100
 ├ DBG_SLEEP                 = 0
 ├ DBG_STOP                  = 0
 ├ DBG_STANDBY               = 0
 ├ TRACE_IOEN                = 0
 ├ TRACE_MODE                = 0x0
 ├ DBG_IWDG_STOP             = 1
 ├ DBG_WWDG_STOP             = 0
 ├ DBG_TIM1_STOP             = 0
 ├ DBG_TIM2_STOP             = 0
 ├ DBG_TIM3_STOP             = 0
 ├ DBG_TIM4_STOP             = 0
 ├ DBG_CAN1_STOP             = 0
 ├ DBG_I2C1_SMBUS_TIMEOUT    = 0
 ├ DBG_I2C2_SMBUS_TIMEOUT    = 0
 ├ DBG_TIM8_STOP             = 0
 ├ DBG_TIM5_STOP             = 0
 ├ DBG_TIM6_STOP             = 0
 ├ DBG_TIM7_STOP             = 0
 └ DBG_CAN2_STOP             = 0
```

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 60 shows the ***Project Option Debugger*** -> ***Setup Pane***.

Figure 60. EWARM C-SPY® Macro script setting


Options for node "Project"

Category:

General Options
Static Analysis
Runtime Checking
C/C++ Compiler
Assembler
Output Converter
Custom Build
Build Actions
Linker
Debugger
Simulator
Angel
CMSIS DAP
GDB Server
IAR ROM-monitor
I-jet/JTAGjet
J-Link/J-Trace
TI Stellaris
Macraigor
PE micro
RDI
ST-LINK
Third-Party Driver
TI MSP-FET

Factory Settings

Setup | Download | Images | Extra Options | Multicore | Plugins

Driver: ST-LINK

☑ Run to: main

Setup macros

☑ Use macro file(s)

$PROJ_DIR\My_DBGMCU_Setup_Macro.mac

Device description file

☐ Override default

$TOOLKIT_DIR$\CONFIG\debugger\ST\STM32L073RZ.ddf

OK Cancel

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

}
```

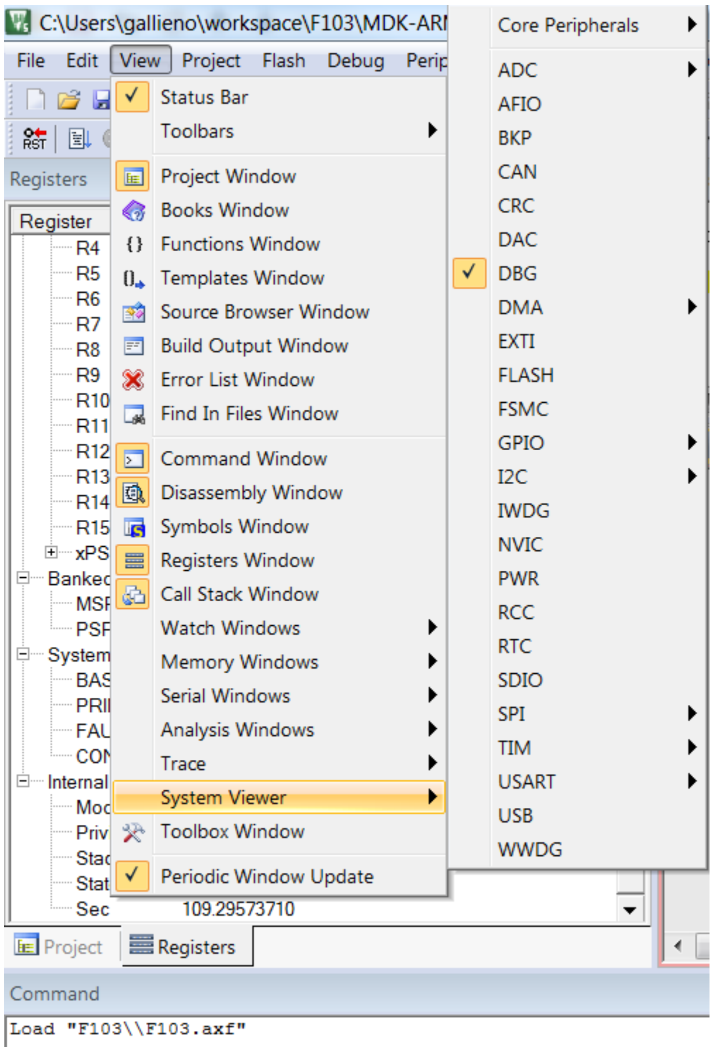
For further information about feature offer by C-SPY® macros please refer to *C-SPY® Debugging Guide* available in IAR Help Menu and on www.iar.com.

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

Keil® MDK-ARM µVision

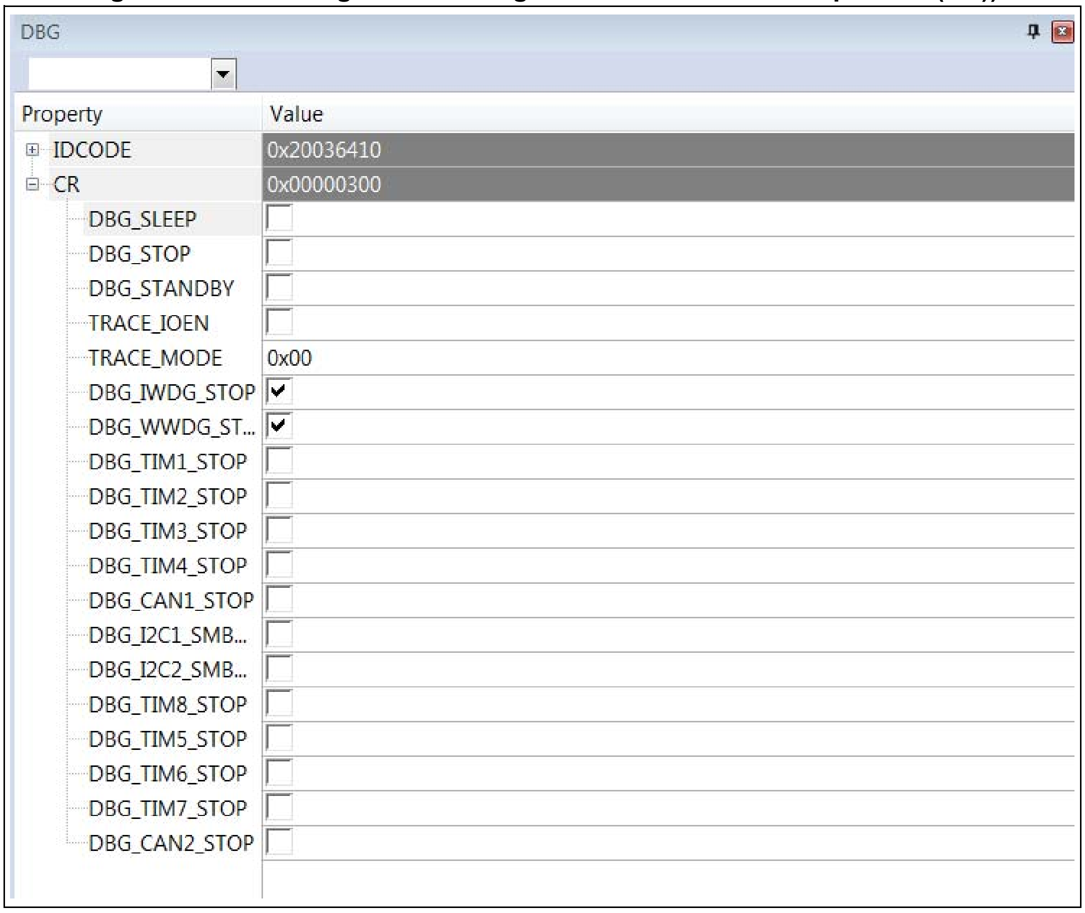
At runtime, access to the DBGMCU register is possible through ***View*** -> ***System Viewer*** -> ***DBG***.

Figure 61. Accessing DBGMCU register in Keil® MDK-ARM µVision (1/2)



```
Load "F103\\F103.axf"
```

Figure 62. Accessing DBGMCU register in Keil® MDK-ARM µVision (2/2))



DBG

Property	Value
⊞ IDCODE	0x20036410
⊟ CR	0x00000300
DBG_SLEEP	☐
DBG_STOP	☐
DBG_STANDBY	☐
TRACE_IOEN	☐
TRACE_MODE	0x00
DBG_IWDG_STOP	✓
DBG_WWDG_ST...	✓
DBG_TIM1_STOP	☐
DBG_TIM2_STOP	☐
DBG_TIM3_STOP	☐
DBG_TIM4_STOP	☐
DBG_CAN1_STOP	☐
DBG_I2C1_SMB...	☐
DBG_I2C2_SMB...	☐
DBG_TIM8_STOP	☐
DBG_TIM5_STOP	☐
DBG_TIM6_STOP	☐
DBG_TIM7_STOP	☐
DBG_CAN2_STOP	☐

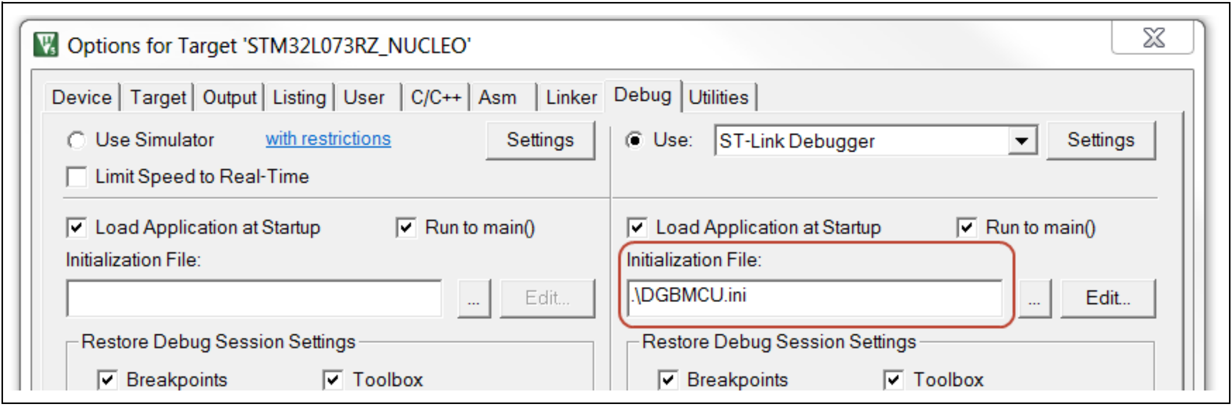
Each bit in the register can be set or reset independently.

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

Figure 63. Keil® Initialization script setting


Options for Target 'STM32L073RZ_NUCLEO'

Device | Target | Output | Listing | User | C/C++ | Asm | Linker | Debug | Utilities

( ) Use Simulator   with restrictions   Settings
[ ] Limit Speed to Real-Time
[✓] Load Application at Startup   [✓] Run to main()
Initialization File:
... Edit...
Restore Debug Session Settings
[✓] Breakpoints   [✓] Toolbox

(●) Use: ST-Link Debugger   Settings
[✓] Load Application at Startup   [✓] Run to main()
Initialization File:
.\DGBMCU.ini
... Edit...
Restore Debug Session Settings
[✓] Breakpoints   [✓] Toolbox

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.

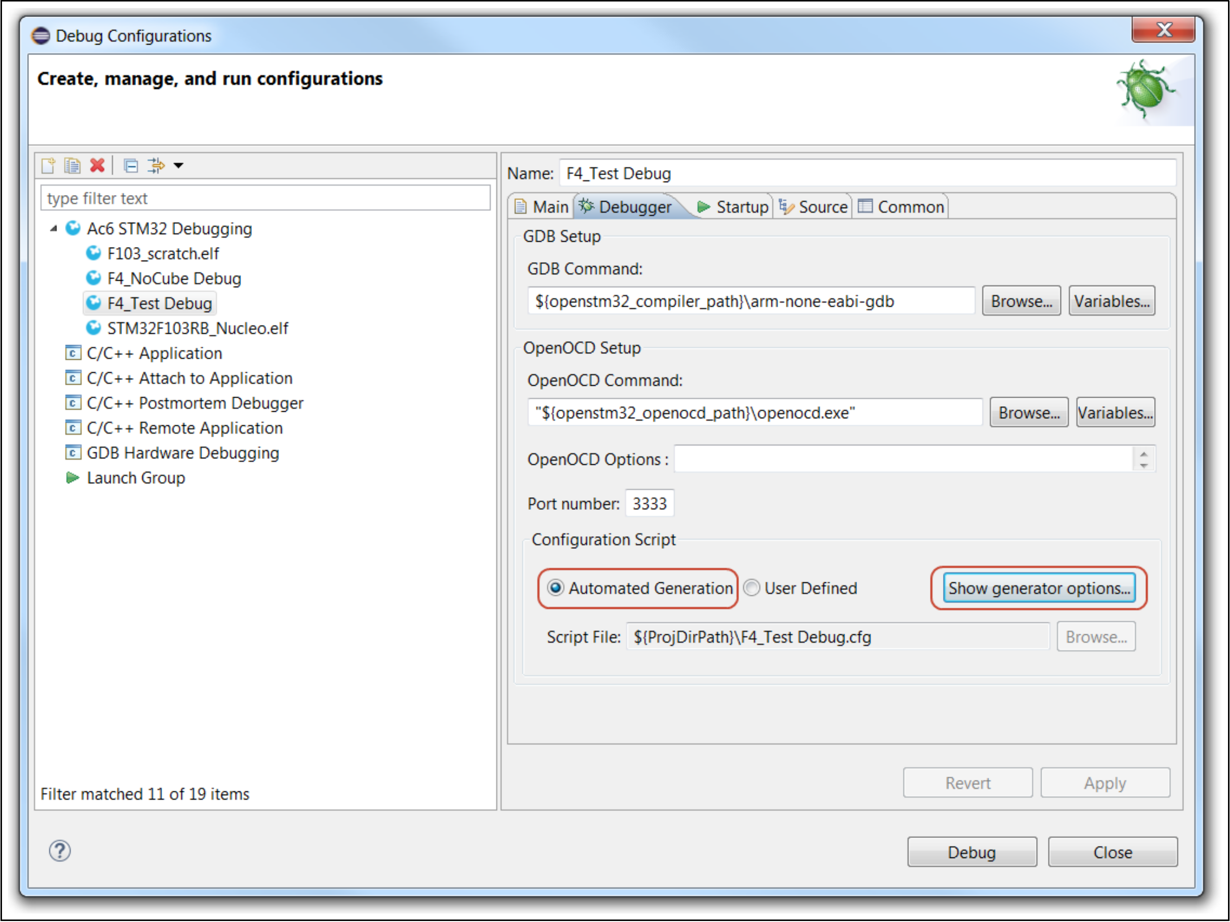
SW4STM32

By default, SW4STM32 performs the following configuration of DBGMCU:

- Enable Low-Power debug
- Freeze IWDG and WWDG while halt.

Since version V2.0.0 of SW4STM32, this default setting can be changed through the OpenOCD Generator Options GUI in ***Debug Configuration*** -> ***Debugger Pane*** by clicking on the Show generator options as presented in *Figure 64*.

Figure 64. Access to Generator Options in SW4STM32 V2.0.0



Debug Configurations

Create, manage, and run configurations

Name: F4_Test Debug

GDB Setup

GDB Command: ${openstm32_compiler_path}\arm-none-eabi-gdb

OpenOCD Setup

OpenOCD Command: "${openstm32_openocd_path}\openocd.exe"

OpenOCD Options :

Port number: 3333

Configuration Script

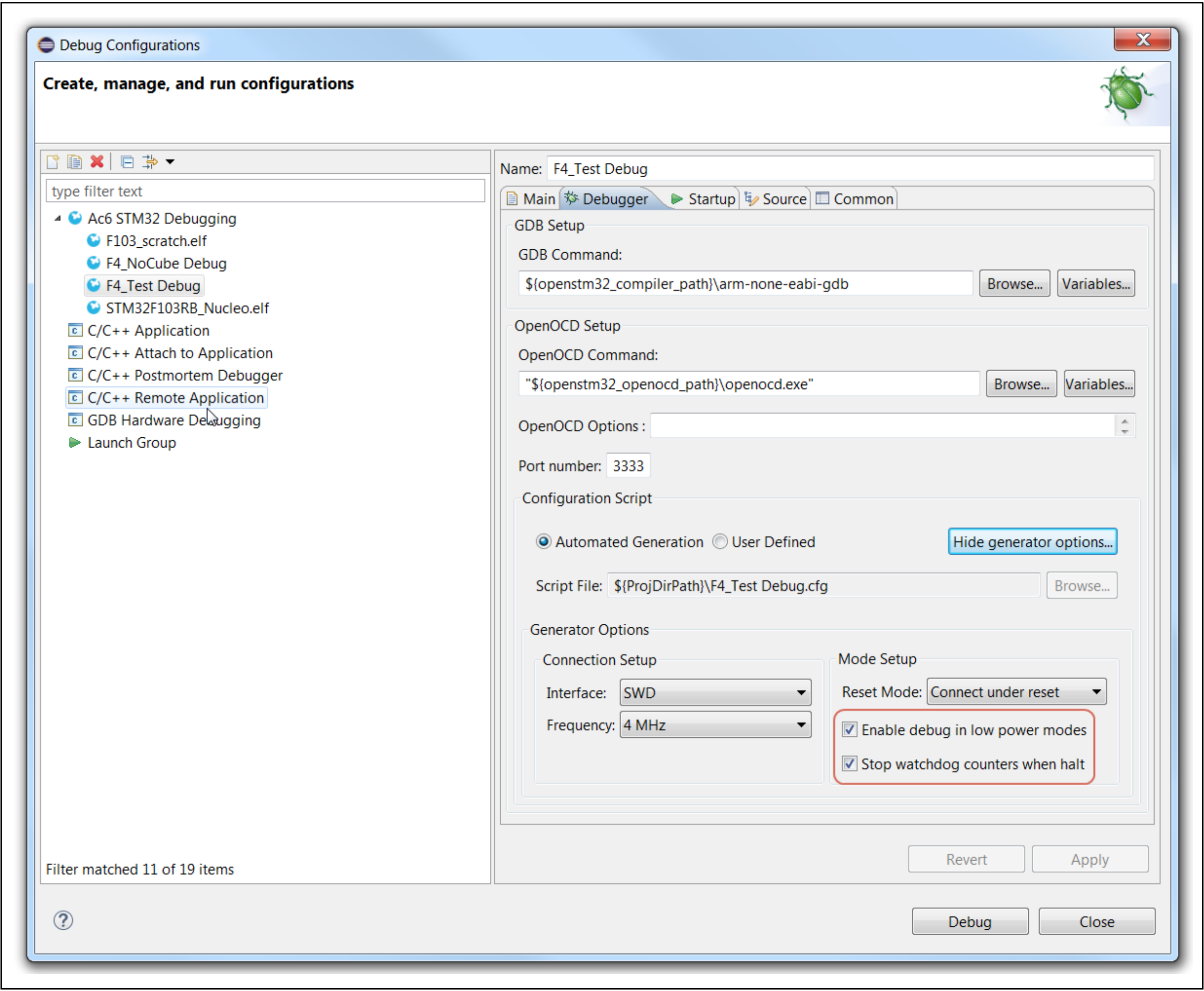
[✓ Automated Generation] User Defined

Script File: ${ProjDirPath}\F4_Test Debug.cfg

Filter matched 11 of 19 items

DBGMCU options are available under Reset Mode in the Mode Setup group as shown in *Figure 65*.

Figure 65. Generator Options debug MCU in SW4STM32 V2.0.0



Name: F4_Test Debug

GDB Command: ${openstm32_compiler_path}\arm-none-eabi-gdb

OpenOCD Command: "${openstm32_openocd_path}\openocd.exe"

OpenOCD Options :

Port number: 3333

[✓ Automated Generation] User Defined

Script File: ${ProjDirPath}\F4_Test Debug.cfg

Interface: SWD

Frequency: 4 MHz

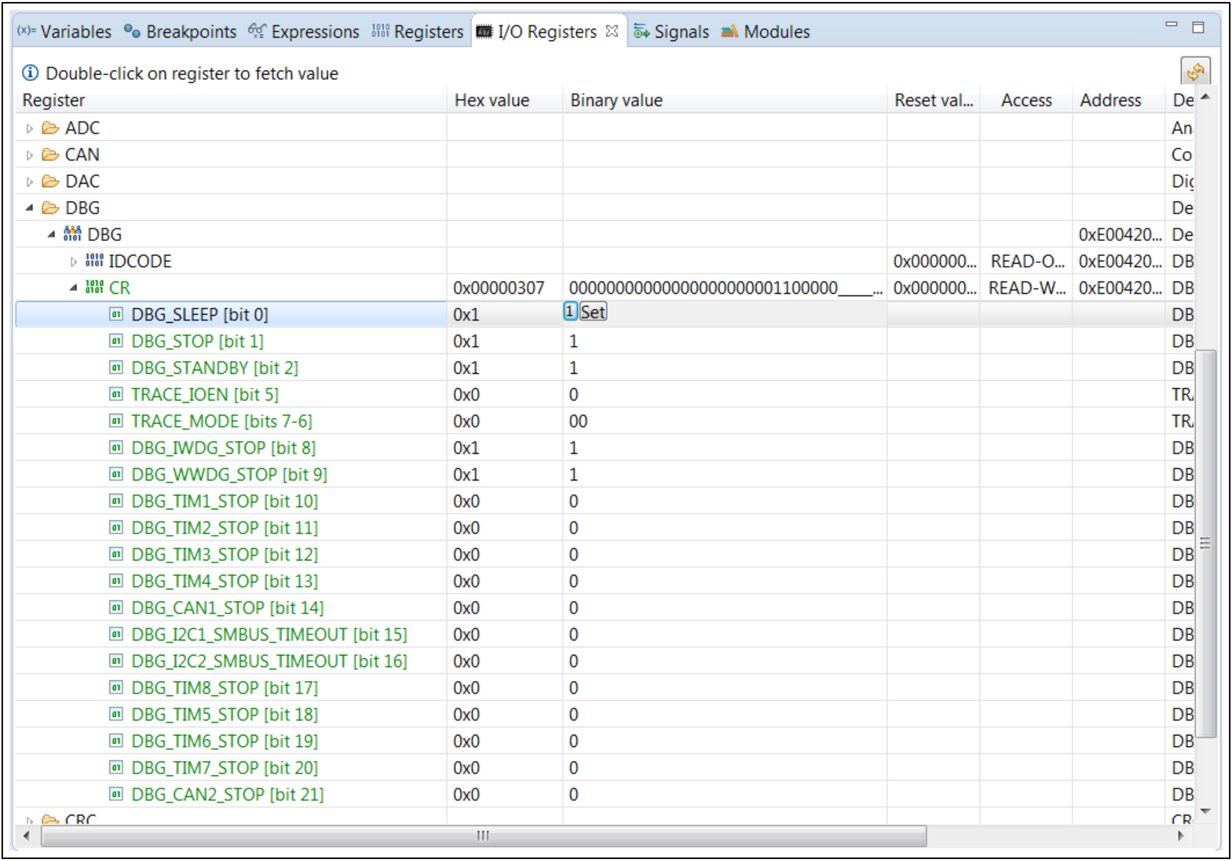
Reset Mode: Connect under reset

☑ Enable debug in low power modes

☑ Stop watchdog counters when halt

If needed, the DBGMCU value can be changed at run time through the I/O Registers window as shown in *Figure 66*.

Figure 66. Runtime R/W access to DBGMCU register with SW4STM32



Double-click on register to fetch value

Register	Hex value	Binary value	Reset val...	Access	Address	De
ADC						An
CAN						Co
DAC						Dig
DBG						De
DBG					0xE00420...	De
IDCODE			0x000000...	READ-O...	0xE00420...	DB
CR	0x00000307	00000000000000000000001100000____...	0x000000...	READ-W...	0xE00420...	DB
DBG_SLEEP [bit 0]	0x1	1 Set				DB
DBG_STOP [bit 1]	0x1	1				DB
DBG_STANDBY [bit 2]	0x1	1				DB
TRACE_IOEN [bit 5]	0x0	0				TR
TRACE_MODE [bits 7-6]	0x0	00				TR
DBG_IWDG_STOP [bit 8]	0x1	1				DB
DBG_WWDG_STOP [bit 9]	0x1	1				DB
DBG_TIM1_STOP [bit 10]	0x0	0				DB
DBG_TIM2_STOP [bit 11]	0x0	0				DB
DBG_TIM3_STOP [bit 12]	0x0	0				DB
DBG_TIM4_STOP [bit 13]	0x0	0				DB
DBG_CAN1_STOP [bit 14]	0x0	0				DB
DBG_I2C1_SMBUS_TIMEOUT [bit 15]	0x0	0				DB
DBG_I2C2_SMBUS_TIMEOUT [bit 16]	0x0	0				DB
DBG_TIM8_STOP [bit 17]	0x0	0				DB
DBG_TIM5_STOP [bit 18]	0x0	0				DB
DBG_TIM6_STOP [bit 19]	0x0	0				DB
DBG_TIM7_STOP [bit 20]	0x0	0				DB
DBG_CAN2_STOP [bit 21]	0x0	0				DB
CRC						CR

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: From debug to release on page 76).

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

As stated in *Section 7.2: Printf via UART on page 59*, it is required to have an adapter between MCU and PC to setup a proper serial connection.

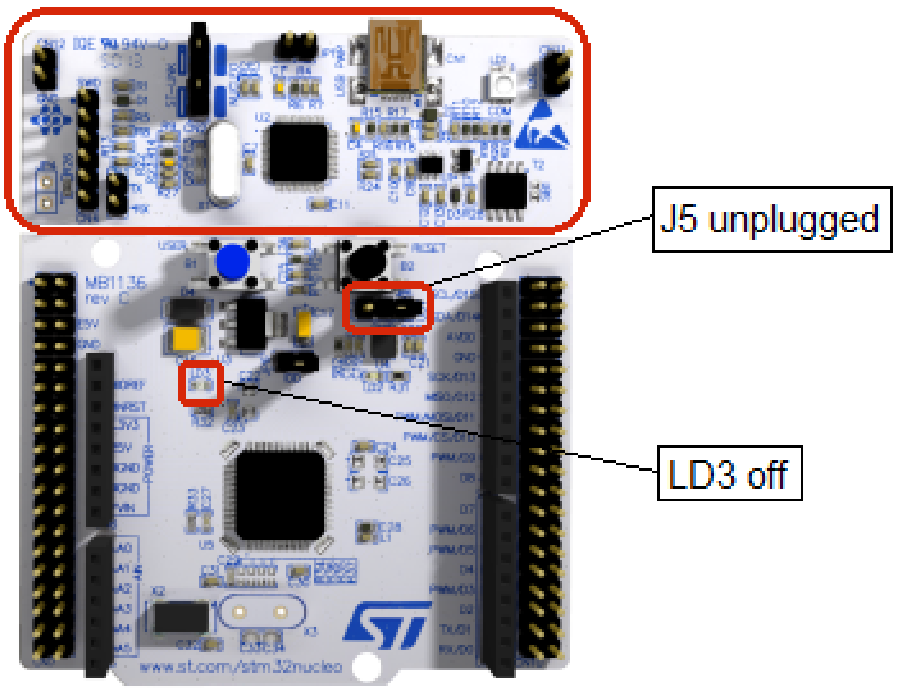
Design constraints may prevent to use the default UART connected to VCP, or may require another serial connection with the PC.

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 67*.

Figure 67. 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

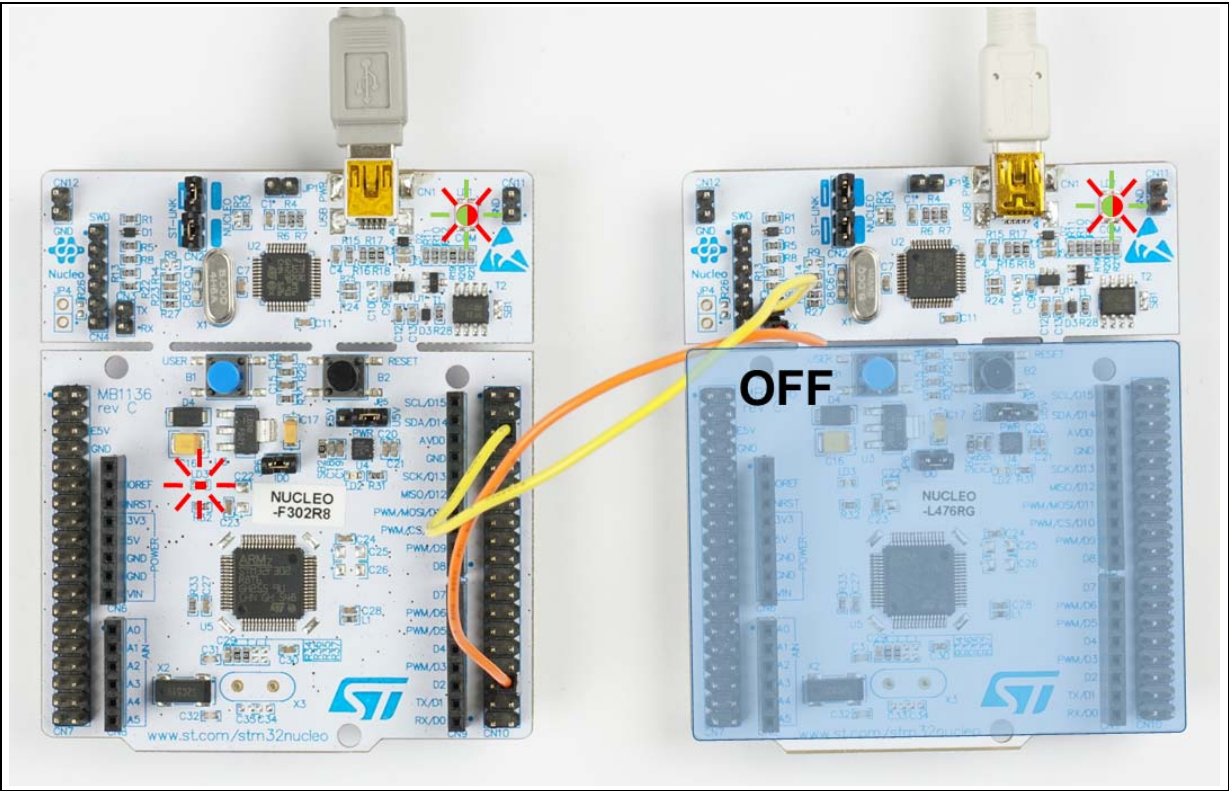
Any available UART of the STM32 application can be connected to the CN3 connector of the ST-LINK part.

Figure 68 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.

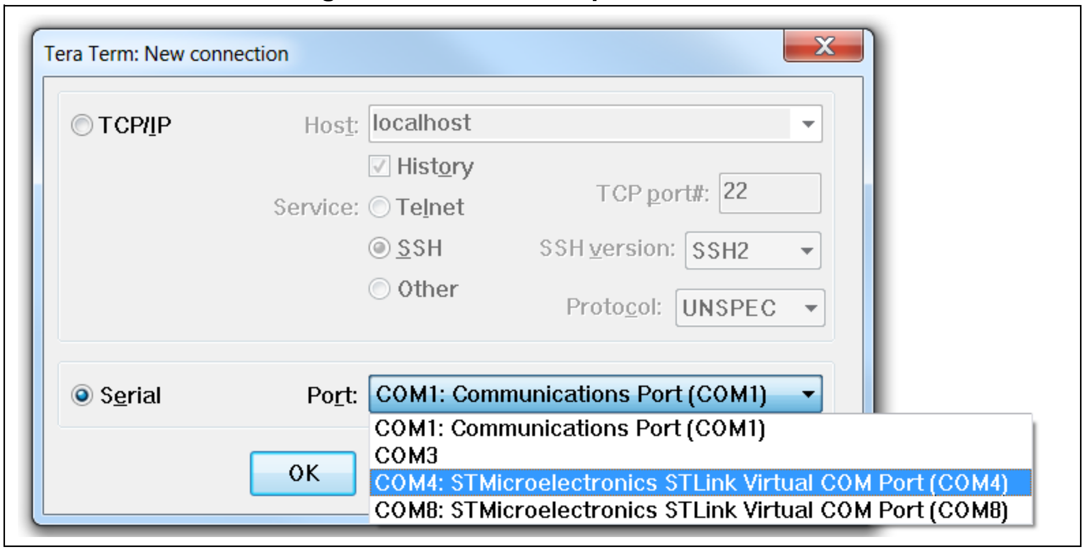
Figure 68. Using ST-LINK stand-alone part of Nucleo-L476RG as VCP



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 69. Virtual-COM port on PC side



Tera Term: New connection

TCP/IP Host: localhost

History

Service: Telnet TCP port#: 22

[✓ SSH] SSH version: SSH2

Other Protocol: UNSPEC

[✓ Serial] Port: COM1: Communications Port (COM1)

COM1: Communications Port (COM1)
COM3
COM4: STMicroelectronics STLink Virtual COM Port (COM4)
COM8: STMicroelectronics STLink Virtual COM Port (COM8)

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.

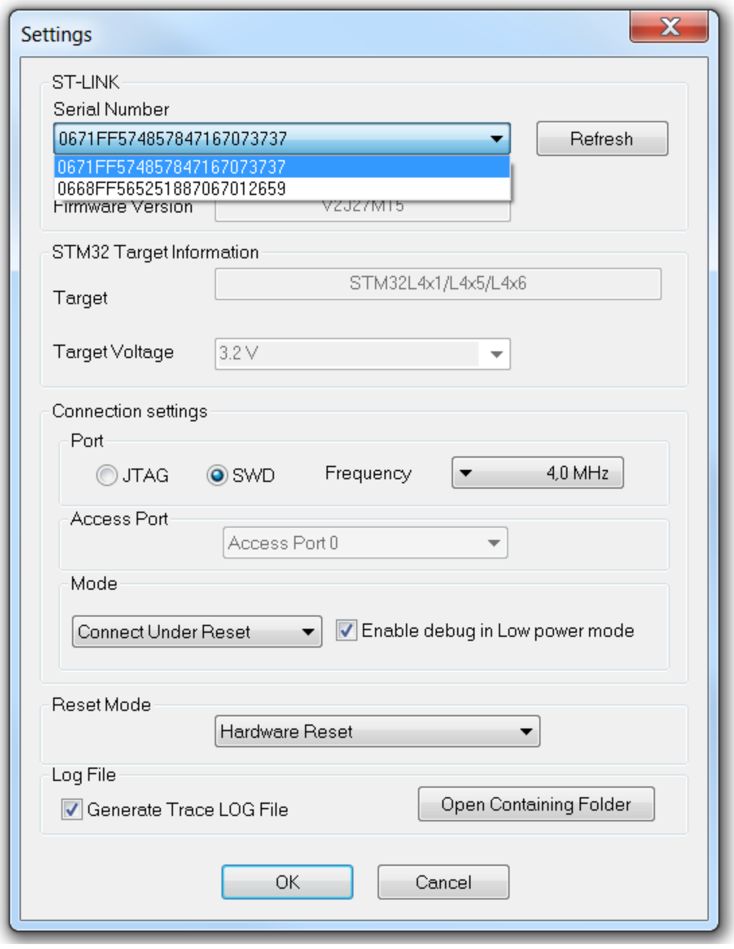
Each ST-LINK connection is identified by a serial number.

In order to correlate a serial number with a board, it is advised to use the ST-LINK utility.

Open the setting pane (***Target*** -> ***Settings***).

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 70. ST-LINK utility target selection pick list



Settings

ST-LINK
Serial Number: 0671FF574857847167073737
- 0671FF574857847167073737
- 0668FF565251887067012659

Firmware Version: V2J27M15

STM32 Target Information
Target: STM32L4x1/L4x5/L4x6
Target Voltage: 3.2 V

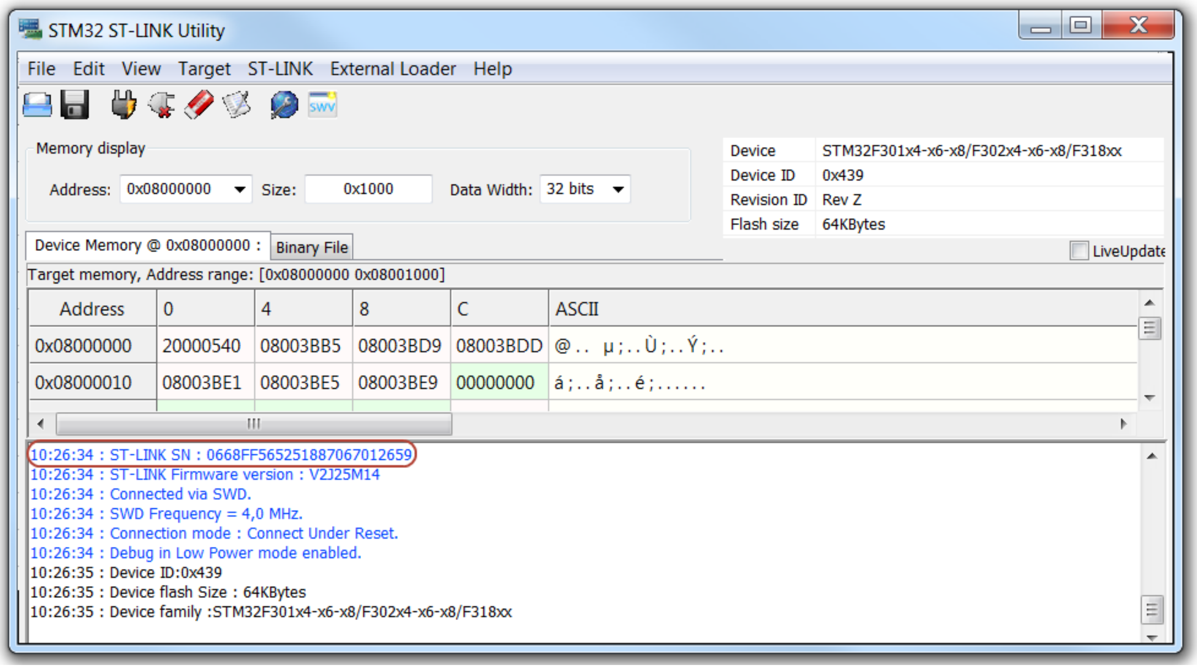
Connection settings
Port: JTAG [✓ SWD]
Frequency: 4.0 MHz
Access Port: Access Port 0
Mode: Connect Under Reset
✓ Enable debug in Low power mode

Reset Mode: Hardware Reset

Log File
✓ Generate Trace LOG File

Once the target is identified, it is possible to copy the S/N from the console in the clip-board as shown in *Figure 71*.

Figure 71. Getting target ST-LINK S/N from the console



Device	STM32F301x4-x6-x8/F302x4-x6-x8/F318xx
Device ID	0x439
Revision ID	Rev Z
Flash size	64KBytes

Target memory, Address range: [0x08000000 0x08001000]

Address	0	4	8	C	ASCII
0x08000000	20000540	08003BB5	08003BD9	08003BDD	@.. µ;..Ù;..Ý;..
0x08000010	08003BE1	08003BE5	08003BE9	00000000	á;..å;..é;......

```
10:26:34 : ST-LINK SN : 0668FF565251887067012659
10:26:34 : ST-LINK Firmware version : V2J25M14
10:26:34 : Connected via SWD.
10:26:34 : SWD Frequency = 4,0 MHz.
10:26:34 : Connection mode : Connect Under Reset.
10:26:34 : Debug in Low Power mode enabled.
10:26:35 : Device ID:0x439
10:26:35 : Device flash Size : 64KBytes
10:26:35 : Device family :STM32F301x4-x6-x8/F302x4-x6-x8/F318xx
```

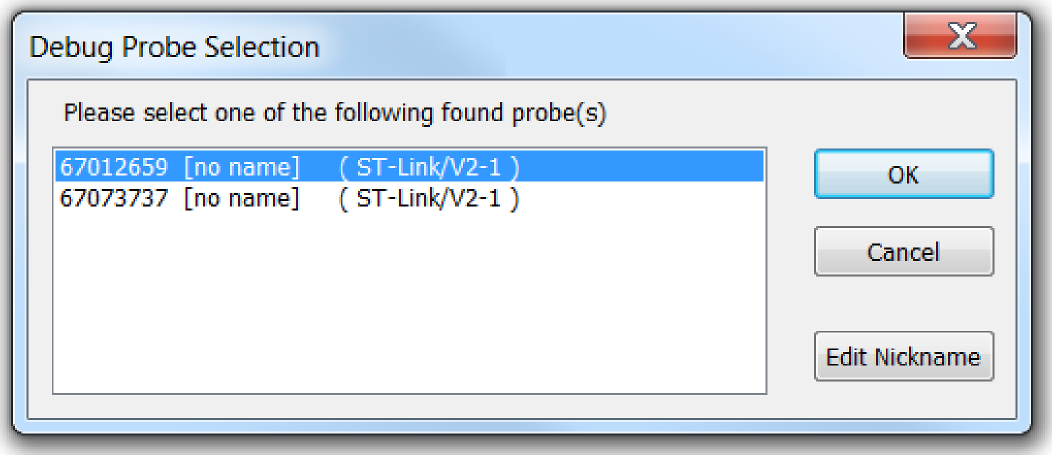
The next sections detail the selection of a specific target with each of the main IDEs considered in this application note.

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 72*.

Figure 72. IAR™ Debug Probe Selection pop-up window

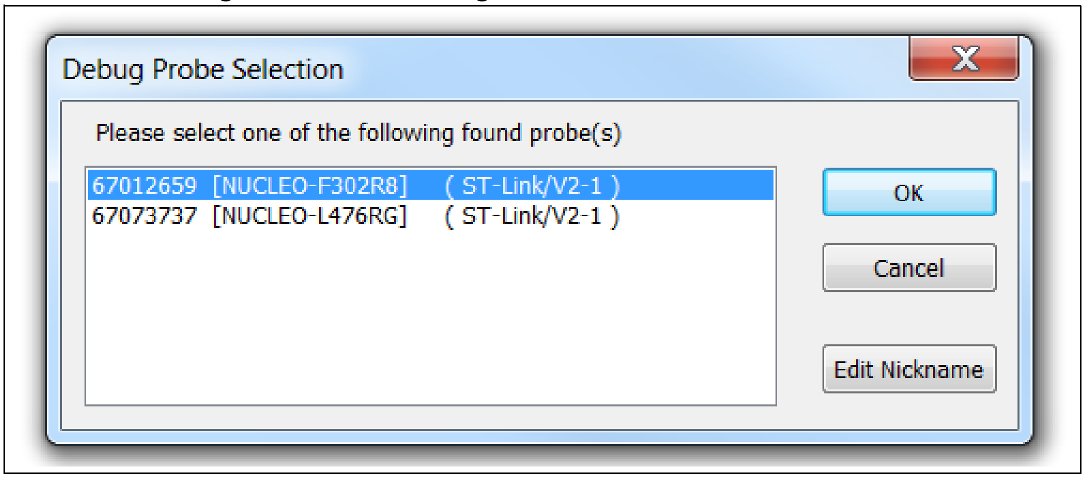


Please select one of the following found probe(s)

```
67012659  [no name]    ( ST-Link/V2-1 )
67073737  [no name]    ( ST-Link/V2-1 )
```

It is recommended to use the Edit Nickname feature to ease board identification in anticipation of further connection as shown in *Figure 73*.

Figure 73. IAR™ Debug Probe Selection with nickname



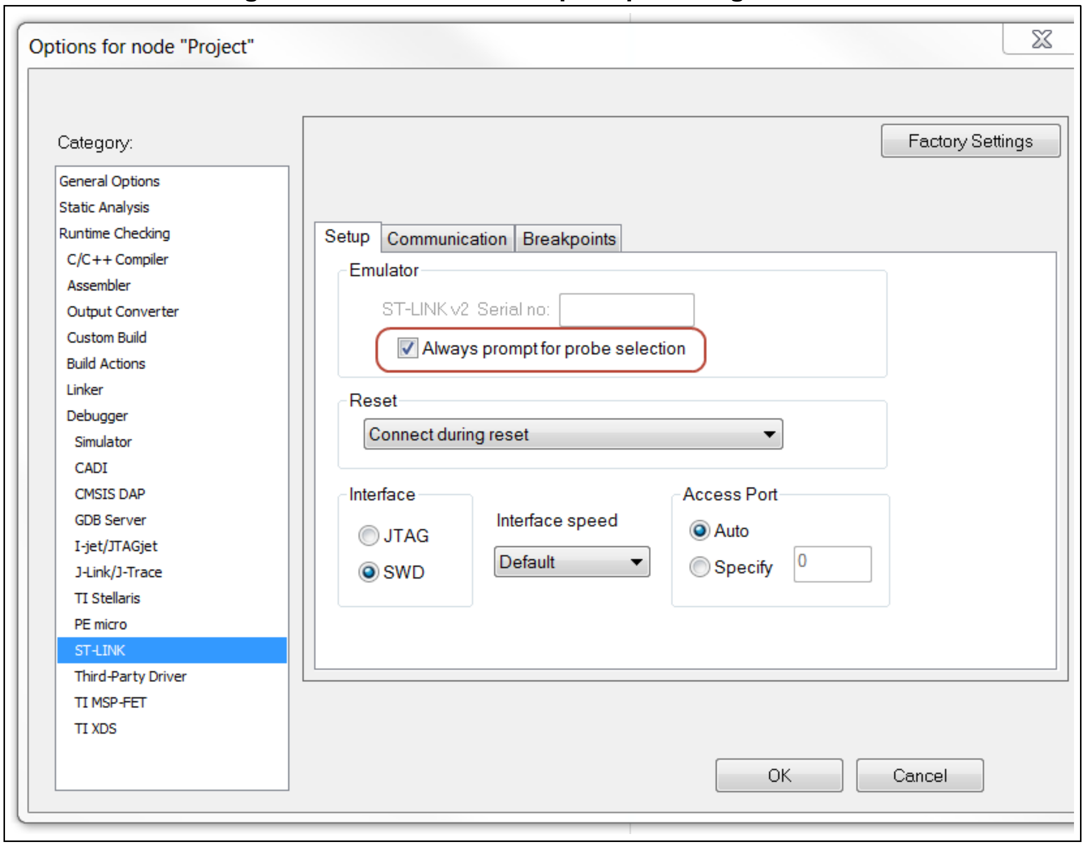
Debug Probe Selection

Please select one of the following found probe(s)

```
67012659 [NUCLEO-F302R8]   ( ST-Link/V2-1 )
67073737 [NUCLEO-L476RG]   ( ST-Link/V2-1 )
```

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 74*.

Figure 74. Probe selection prompt setting on IAR™



Options for node "Project"

Category: ST-LINK

Setup tab:

Emulator
- ST-LINK v2 Serial no:
- [✓] Always prompt for probe selection

Reset: Connect during reset

Interface: JTAG [✓ SWD]

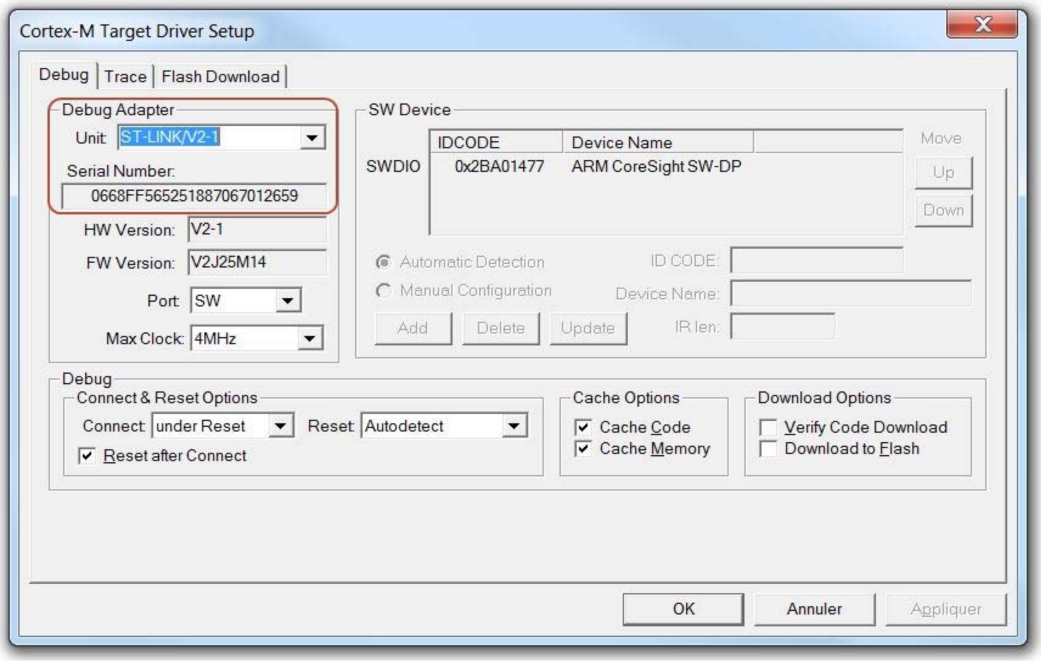
Interface speed: Default

Access Port: [✓ Auto] Specify 0

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 75*.

Figure 75. Keil® ST-LINK selection



Cortex-M Target Driver Setup

Debug | Trace | Flash Download

Debug Adapter
Unit: ST-LINK/V2-1
Serial Number: 0668FF565251887067012659
HW Version: V2-1
FW Version: V2J25M14
Port: SW
Max Clock: 4MHz

SW Device

	IDCODE	Device Name
SWDIO	0x2BA01477	ARM CoreSight SW-DP

Automatic Detection (selected) / Manual Configuration

Debug
Connect & Reset Options
Connect: under Reset
Reset: Autodetect
[✓] Reset after Connect

Cache Options
[✓] Cache Code
[✓] Cache Memory

Download Options
[ ] Verify Code Download
[ ] Download to Flash

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.

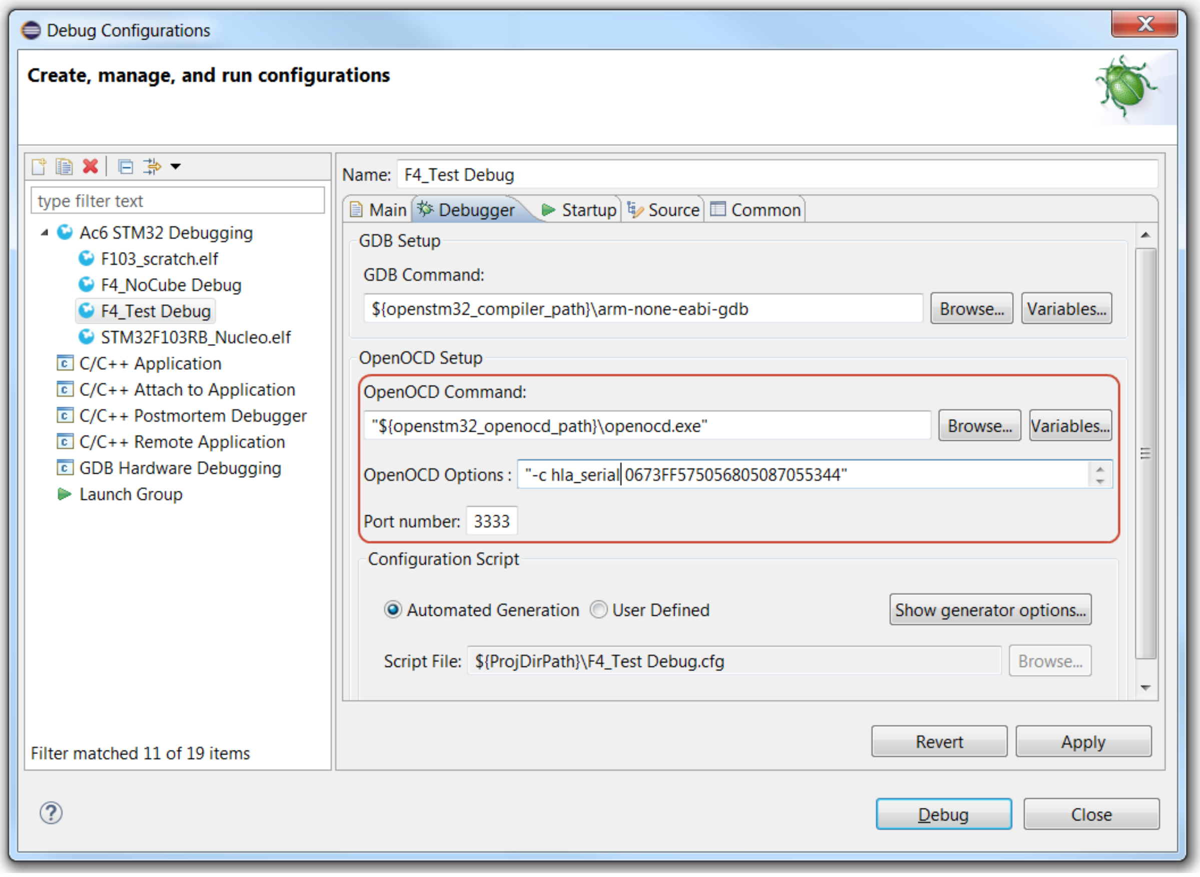
SW4STM32

It is possible to force the connection to a specific target using the ST-LINK S/N. In ***Debug Configurations*** -> ***Debugger Pane***, add the following OpenOCD option:

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

Figure 76 illustrates the setting of an OpenOCD option for forcing a connection.

Figure 76. Forcing specific ST-LINK S/N with SW4STM32



Name: F4_Test Debug

GDB Command: ${openstm32_compiler_path}\arm-none-eabi-gdb

OpenOCD Command: "${openstm32_openocd_path}\openocd.exe"

OpenOCD Options : "-c hla_serial 0673FF575056805087055344"

Port number: 3333

[✓ Automated Generation] User Defined

Script File: ${ProjDirPath}\F4_Test Debug.cfg

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 71*)
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.

Revision history

Table 8. Document revision history

Date	Revision	Changes
16-Jun-2017	1	Initial release.
29-Jun-2017	2	Added *Table 1: Applicable products*.