STM32H7x5/x7 dual-core microcontroller debugging

Introduction

The STM32H7x5/x7 dual-core microcontroller lines as described in Table 1. Applicable products (named STM32H7x5/x7 microcontrollers in this document) are based on the high-performance Arm® Cortex®-M7 and Cortex®-M4 32-bit RISC cores.

The STM32H7x5/x7 microcontrollers require a specific development approach so the application can take maximum advantage dual core architecture.

This application note provides guidelines for the debug of custom applications which run on the STM32H7x5/x7 microcontrollers using the software Toolchains: EWARM, MDK-ARM and SW4STM32.

The following subjects are addressed;

- An overview of the new IDE features which supports the dual core products.
- How to use EWARM, MDK-ARM and SW4STM32 with a dual-core device using the ST-LINK debug.

For more information on the STM32H7x5/x7 microcontrollers, refer to the following documents on www.st.com:

- STM32H745xI/G datasheet (DS12923)
- STM32H755xI datasheet (DS12919)
- *STM32F7 Series and STM32H7 Series Cortex®-M7 processor programming manual* (PM0253)

Table 1. **Applicable products**

Type	Product lines
Microcontrollers	STM32H745/755, STM32H747/757

1 General information

This document applies to the STM32H7x5/x7 dual-core Arm®-based microcontroller lines.

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

2 Dual debug main features

A multicore processor is composed of either heterogeneous cores (which means different cores) or homogenous cores (identical cores).

The STM32H7x5/x7 dual-core microcontrollers use an heterogeneous core architecture. It is composed of an Arm® Cortex®-M7 core and an Arm® Cortex®-M4 core.

The two cores boot either separately or together. This is configurable using dedicated option bytes BCM7 and BCM4.

2.1 Access port

The STM32H7x5/x7 microcontrollers contain four access ports (AP) attached to the debug port (DP):

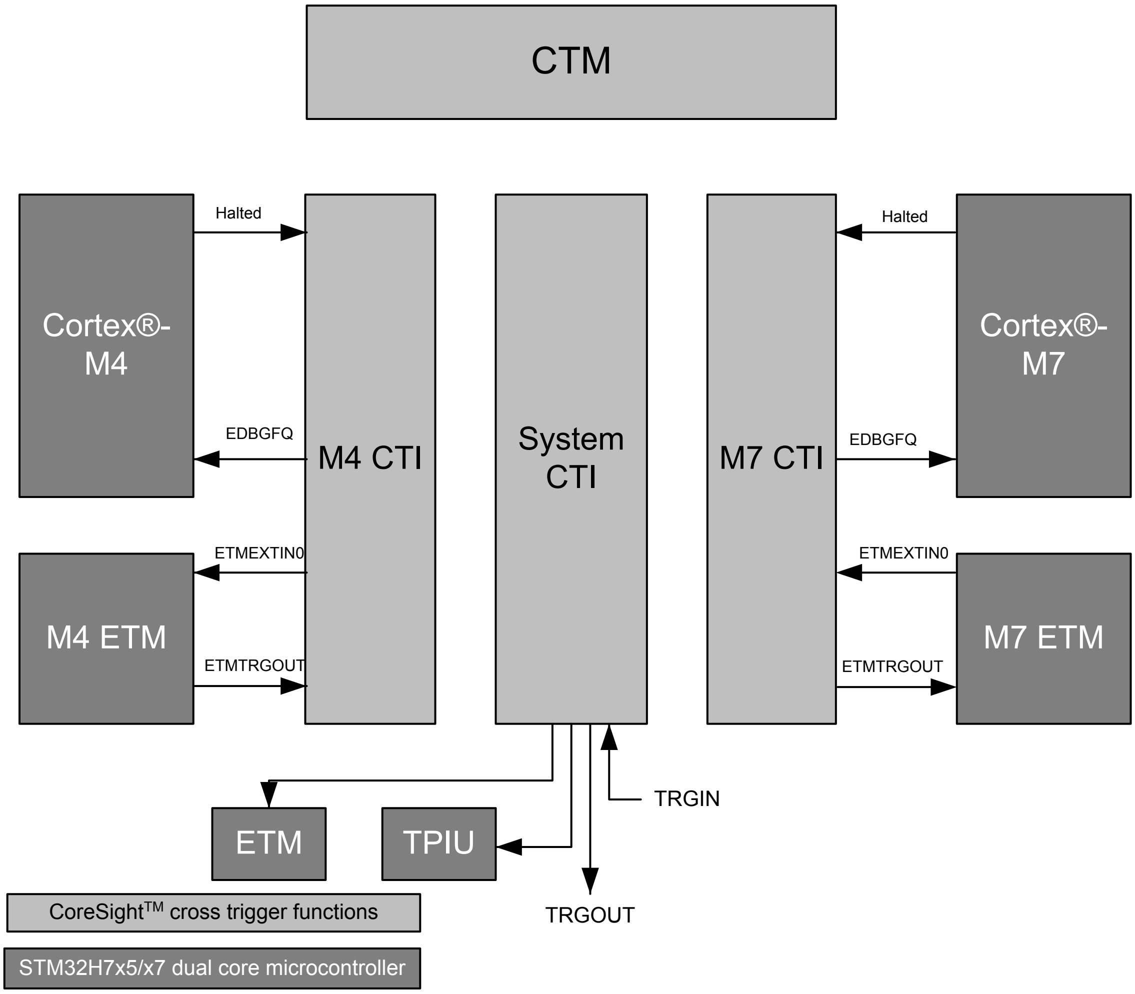
1. AP0: Cortex®-M7 access port (AHB-AP) gives access to the debug and trace features integrated in the Cortex®-M7 processor core via an AHB-Lite bus connected to the AHBD port of the processor.
2. AP1: D3 access port (AHB-AP) which allows access to the bus matrix in the D3 domain. The D3 domain memory and peripherals are be visible when the D1 and D2 domains are switched off. No Arm® CoreSight™ components are accessible via this port.
3. AP2: System access port (APB-AP) allows access to the debug and trace features on the APB debug system bus, that includes all components that are not part of any of the processor cores.
4. AP3: Cortex®-M4 access port (AHB-AP) allows access to the debug and trace features integrated in the Cortex®-M4 processor core via its internal AHB bus.

2.2 Cross trigger interface (CTI) and cross trigger matrix (CTM)

The cross trigger interfaces (CTI) and cross trigger matrix (CTM) together form the Arm® embedded cross trigger feature. There are three CTI components, one at system level, one dedicated to the Cortex®-M7 and one dedicated to the Cortex®-M4. The three CTIs are connected to each other via the CTM as illustrated in Figure 1. The system-level and the Cortex®- M4 CTIs are accessible to the debugger via the system access port and associated APB-D.

The Cortex®-M7 CTI is physically integrated in the Cortex®-M7 core, and is accessible via the Cortex®-M7 access port and associated AHBD.

Figure 1. **Embedded cross trigger**



The CTIs allow events from various sources to trigger debug and trace any activity. For example, a break-point reached in one of the processor cores stops the other processor, or a transition detected on an external trigger input starts code trace.

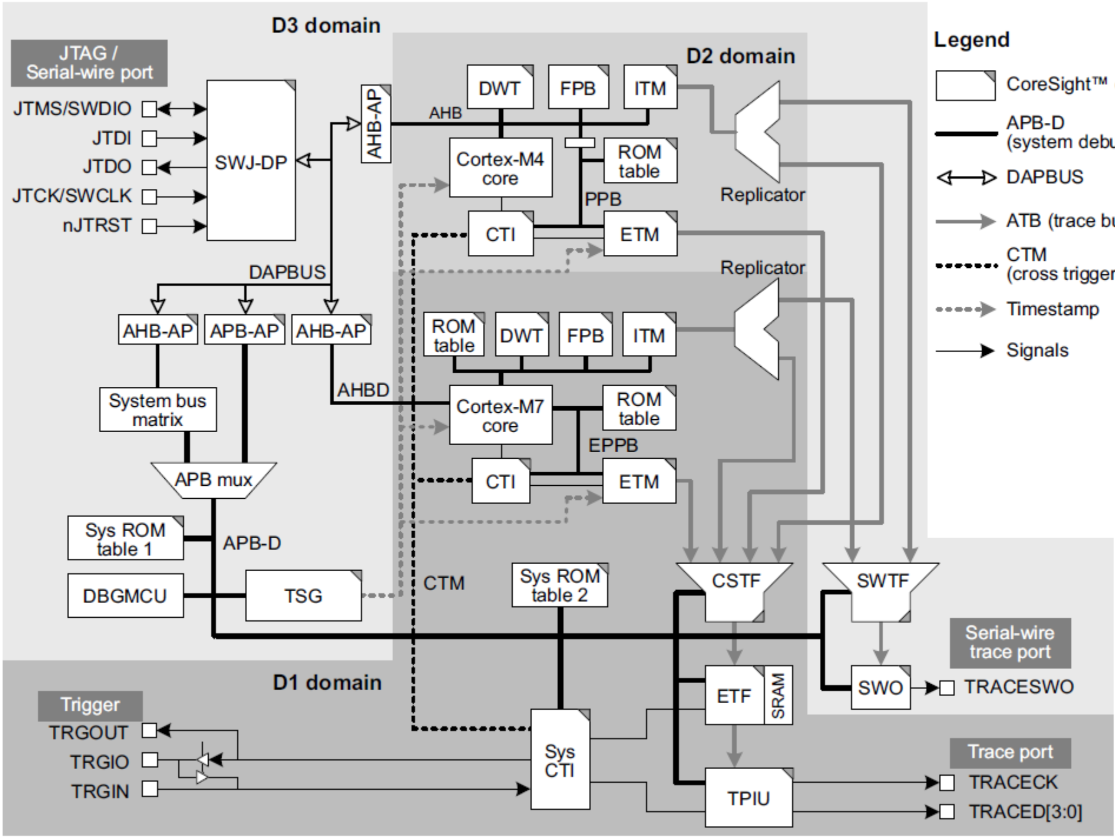
Each CTI has up to 8 trigger inputs and 8 trigger outputs. Any input can be connected to any output, on the same CTI, or on another CTI via the CTM.

For more information on cross trigger input and output signals for each CTI, refer to the reference manual *STM32H745/755 and STM32H747/757 advanced Arm®-based 32-bit MCUs* (RM0399).

2.3 Debug power domain

The debug components are distributed across the power domains D1 , D2 and D3 as illustrated in Figure 2.

Figure 2. Debug power domain

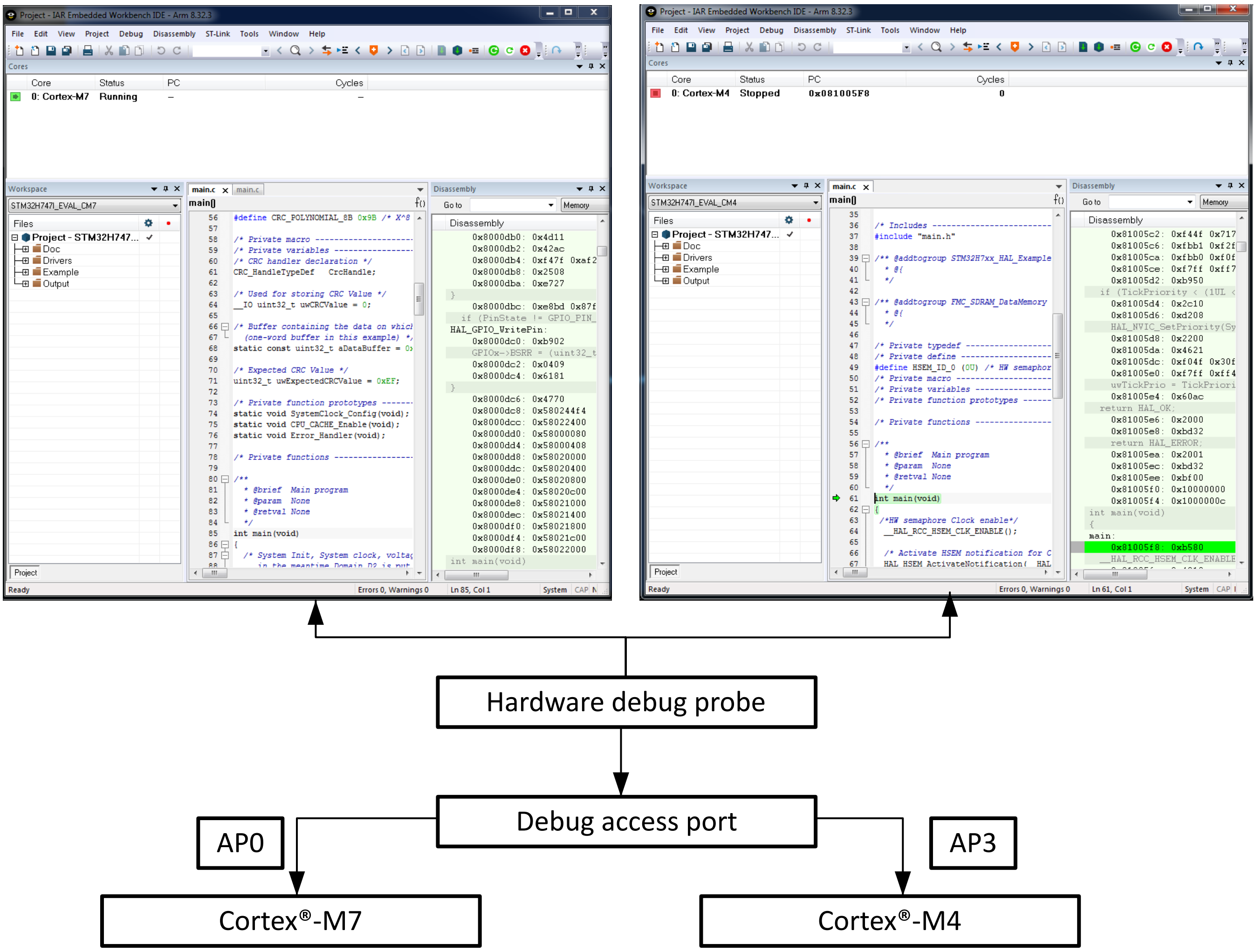


- D1 domain (highlighted in dark gray): contains the Cortex®-M7 core with the associated debug and trace components. It also contains the system trace components located on the APB-D which are common to both processors. This power domain therefore needs to be active whenever the Cortex®-M7 debug access is required, or when the trace functionality is active on either processor.
- D2 domain (highlighted in mid gray) only contains the debug and trace components dedicated to the Cortex®-M4. Removing this power domain does not impact the Cortex®-M7 debug and trace.
- D3 Domain (highlighted in light gray) must be active when the debugger is connected to it as it contains the serial wire JTAG debug Port (SWJDP). This ensures the debugger does not loose the connection with the system on chip (SoC) when one or both of the other power domains are switched off. In addition, it contains the time-stamp generator, the DBG_MCU and the serial wire trace features which allow the Cortex®-M7 core to be switched off while still allowing basic trace on the Cortex®-M4.

3 Debuggers (software and hardware)

Dual-core debugging allows the simultaneous debugging of both cores using a single hardware debug probe. The debug information for both cores is displayed either in a single IDE GUI or have an IDE GUI instance for each core. A representation of separate IDE GUI instances is illustrated in Figure 3.

Figure 3. Dual debugger



To ensure dual debug, a debugger must offer:

- access port selection capability
- simultaneous multiple core connection capability using the same debug probe
- visibility of all cores
- support of cross trigger Arm® component
- possibility to switch between access ports during the same debug session to visualize memory and peripherals statutes in the other domains.

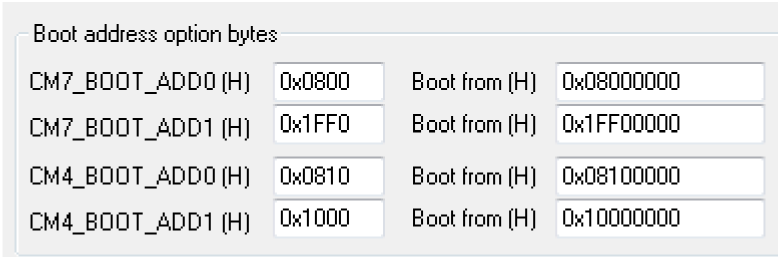
The supported dual-core management features differ from a debugger to another.

In the STM32H7x5/x7 microcontrollers, the Flash memory is divided into two banks:

- CM7 uses bank1 from 0x0800 0000 to 0x080F FFFF
- CM4 uses bank2 from 0x0810 0000 to 0x081F FFFF.

Figure 4 illustrates the boot address options.

Figure 4. Boot address



4 Using IAR Embedded Workbench® for Arm®

The latest version of IAR Embedded Workbench® for Arm® (EWARM) is available for download from the official web site of IAR Systems® System https://www.iar.com.

Note: Dual debug is supported using IAR starting from version 8.30.

4.1 Dual debugging on EWARM

IAR Embedded Workbench® offers the possibility to simultaneously debug two cores with different architectures, asymmetric multicore processing (AMP) using the same project with different configurations for each core or from two separate workspaces.

4.2 Steps for dual debug with ST-LINK/V2

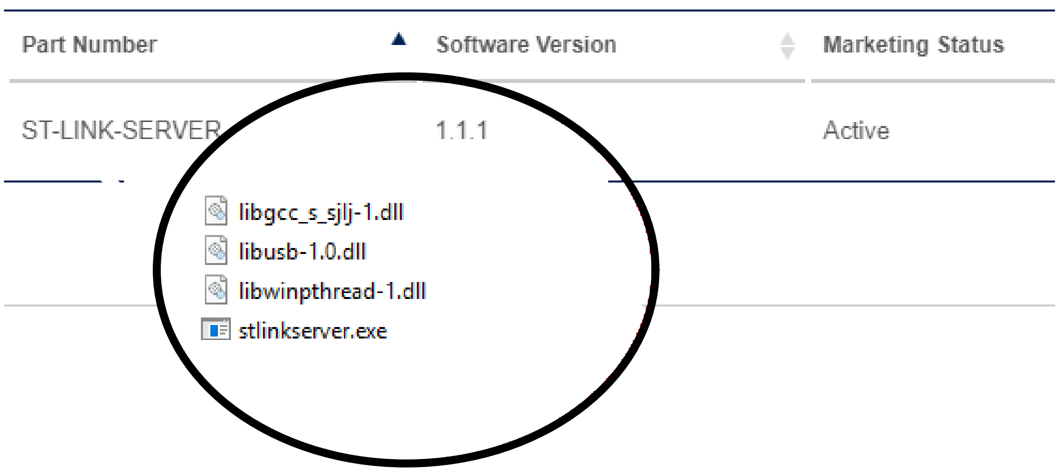
Once the debug tools are installed, the debug operations can start. Here is the step by step instructions working with the STM32H7x5/x7 microcontrollers using EWARM v8.32.3 and ST-LINK as debug probe.

Core project configuration

This example creates one project for each core and to set up dual-core debugging using an ST-LINK/V2:

1. Install the latest version of the STLINK server available on www.st.com as illustrated in Figure 5.

Figure 5. **ST-LINK server installation**



Note: Dual debug is supported using ST-LINK server starting from version v1.1.1-3.

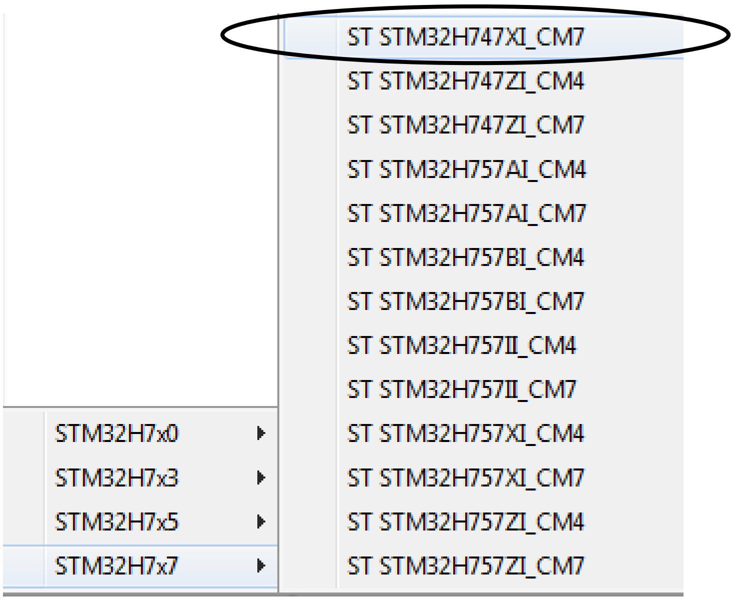
The ST-LINK USB driver installs automatically. To manually install the tool chain:

a. Go to the tool-chains installation directory and run the STLink_V2_USB.exe from **(INSTALL PATH)\IAR Systems\Embedded Workbench 8. 2\arm\ drivers\ST-Link \ST-Link_V2_USBdriver.exe**

b. Follow the procedure for EWARM with the STM32H7x5/x7 microcontrollers using the ST-LINK V2/V3 debug probe. First step is to configure the CM7_project.

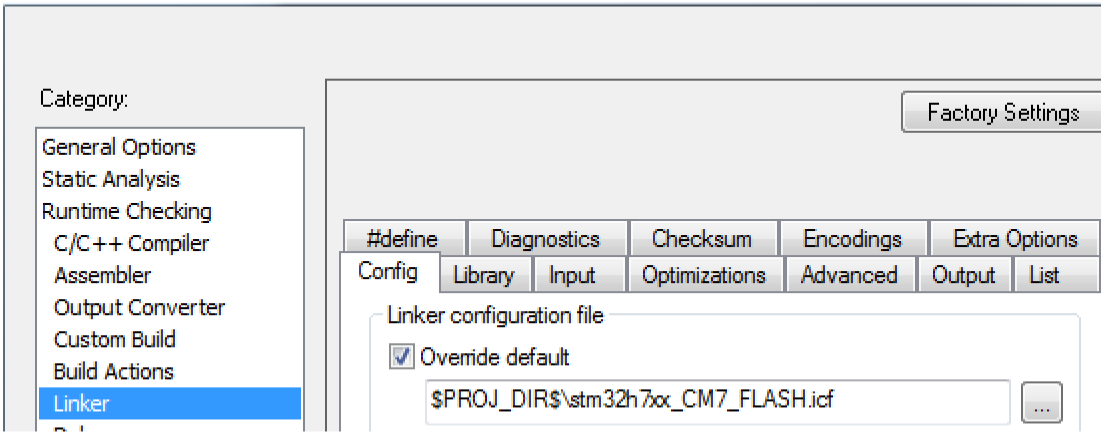
2. Open the IAR Embedded Workbench® and create one project for each core. Ensure the settings are compatible for both debug probe in each project. For the CM7-project. Choose the correct device using **Project** → **Options** → **Target Device** (Figure 6).

Figure 6. Device selection



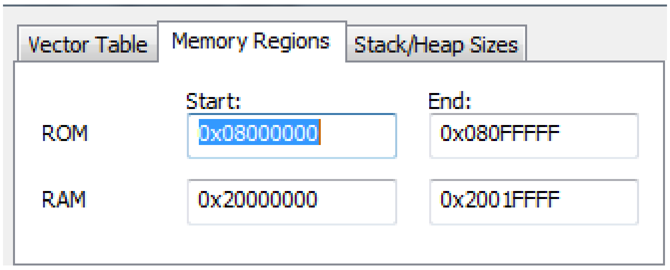
3. Go to: **Project** → **Options** → **Linker** → **Config** (Figure 7):

Figure 7. Linker → Config tab



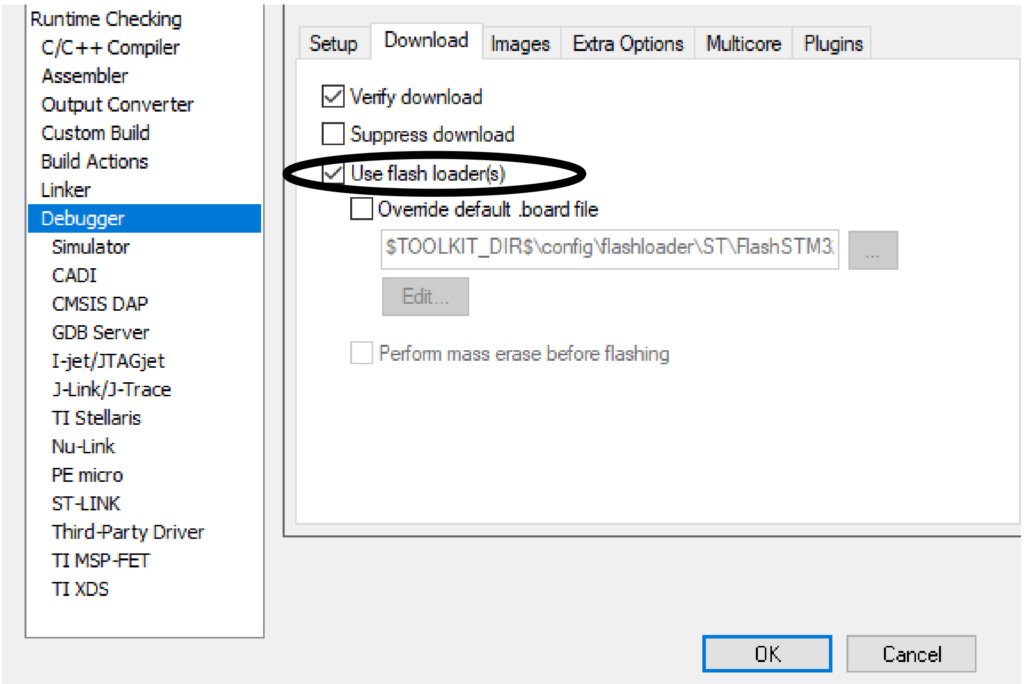
   a. *Click Edit to display the linker configuration file editor.*
   b. *Check the application address is correctly configured in the linker configuration file.*
   c. *Open the Vector Table tab and set the* ***intvec*** *start variable to 0x08000000.*
   d. *Open the Memory Regions tab, and enter the variables (Figure 8).*

Figure 8. Memory Regions tab



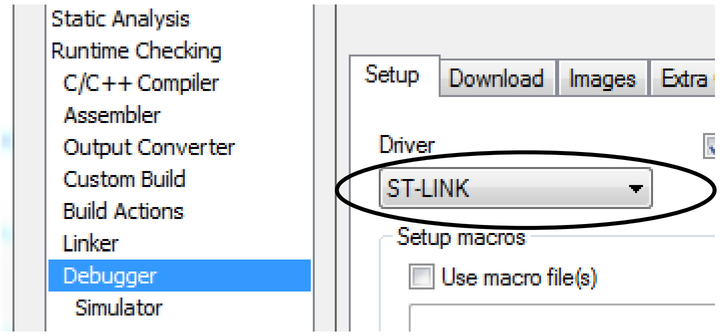
4. Open the Debugger Category from: **Project → Options → Debugger.**
   Select the Download tab and tick the Use flash loader(s) check box (Figure 9).

Figure 9. Select Flash loader



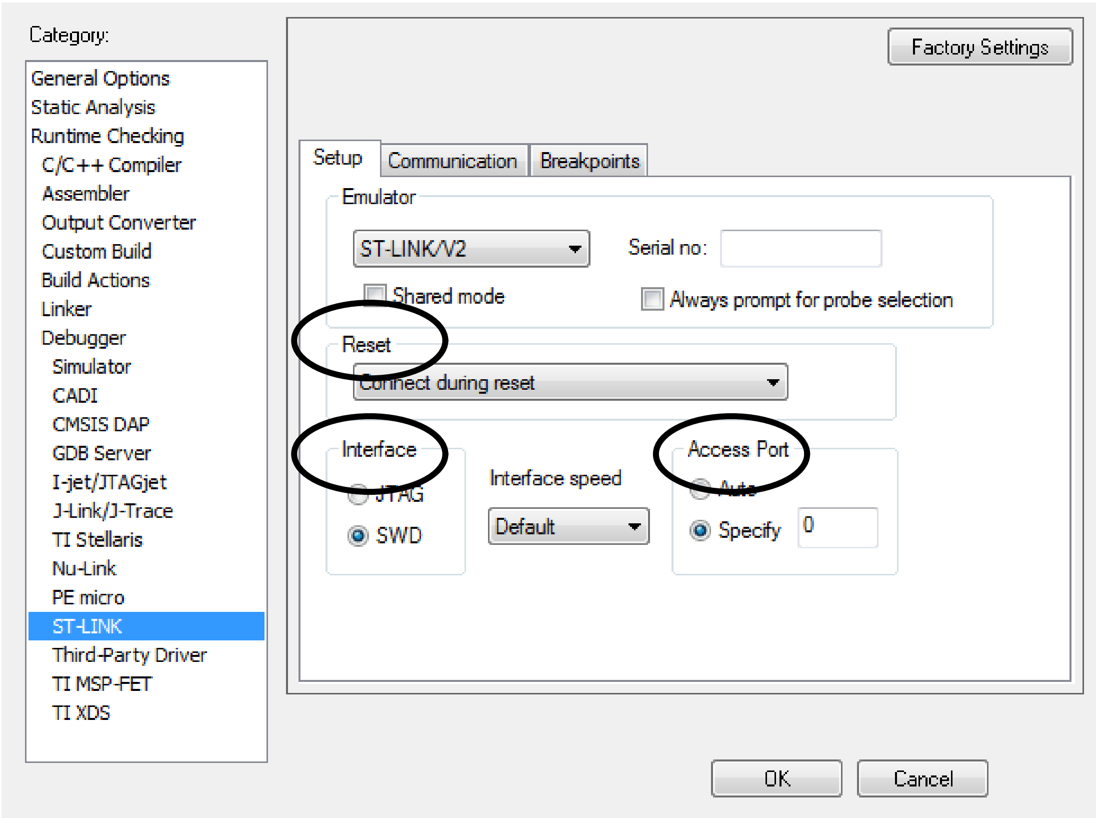
5. Select ST-LINK as a debug probe from: **Setup tab** (Figure 10).

Figure 10. Debugger → Setup tab



6. From **Project** →**Options** → **ST-LINK** → **Setup** (Figure 11)

Figure 11. ST-LINK → Setup tab



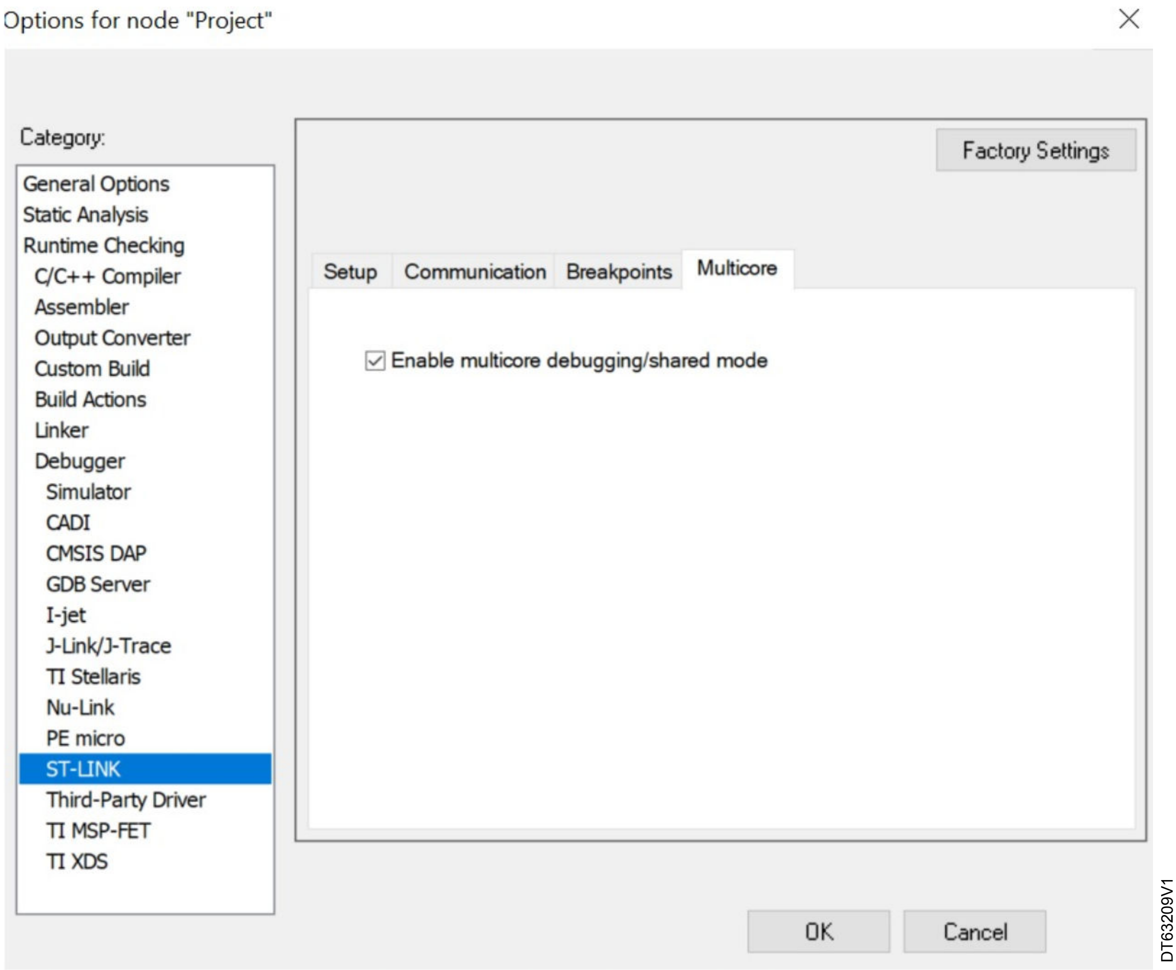
a. *Select the Reset type:*
   - *System reset: resets the cores and peripherals.*
   - *Core reset: resets the core via the VECTRESET bit, the peripheral units are not affected.*
   - *Software reset: sets the PC to the program entry address.*
   - *Hardware reset: the probe toggles the nSRST/nRESET line on the JTAG connector to reset the device. This reset usually resets the peripheral units also.*
   - *Connect during reset: ST-LINK connects to the target while keeping Reset active. Reset is pulled low and remains low while connecting to the target.*

b. *Select the communication interface:*
   - *JTAG: to use the JTAG interface.*
   - *SWD: to use the SWO interface, which requires fewer pins than JTAG. Select SWD to use the serial-wire output (SWO) communication channel.*

c. *Select the access port : AP0 for CM7 or by selecting the Auto option*
   - *Auto: automatically uses access port 0 for CM.*
   - *Manual: the user specifies the access port to be used.*

7. Enable the use of ST-Link server by selecting shared mode from : **Project** → **Options** → **ST-LINK** → **Multicore** (Figure 12).

Figure 12. ST-LINK → Multicore

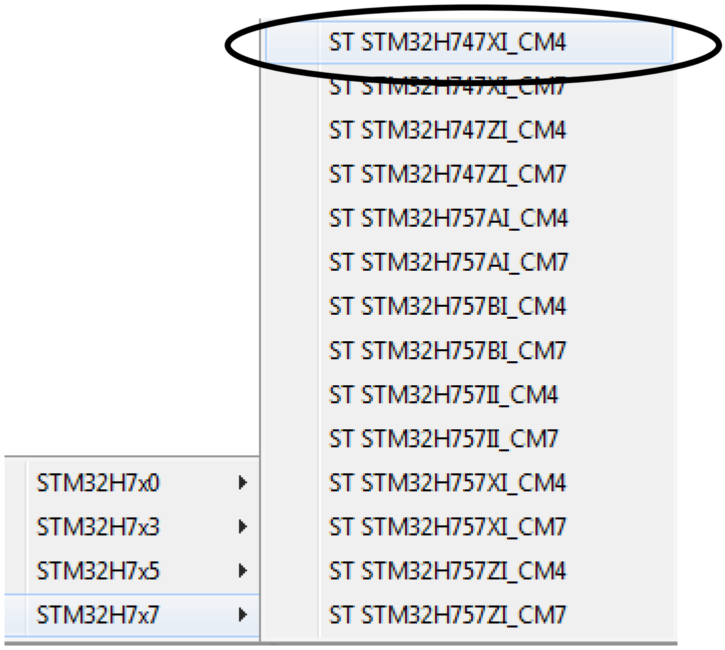


CM4-project configuration

To configure the CM4-project follow the same steps mentioned above for CM7 and make the following changes:

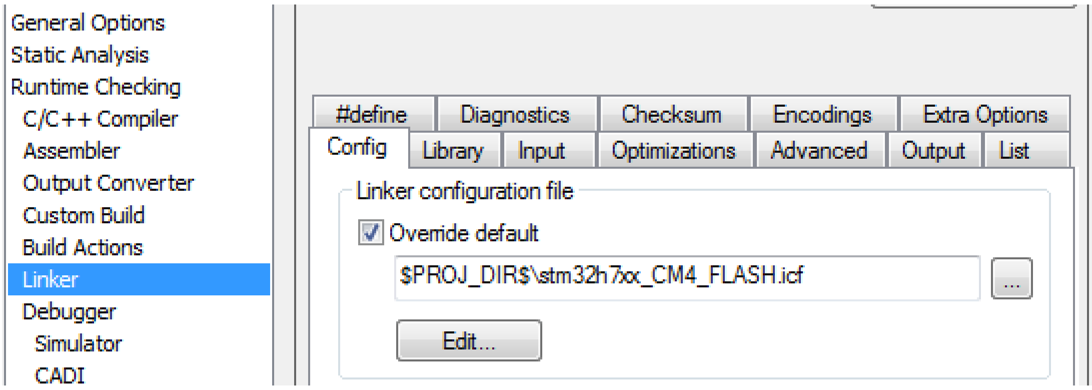
1. Choose the correct device **Project** → **Options** →**Target** → **Device** (Figure 13).

Figure 13. Device selection



2. Go to **Project** → **Options** → **Linker** → **Config**. Examine the linker configuration file to make sure that the application has been linked to the right address (Figure 14).

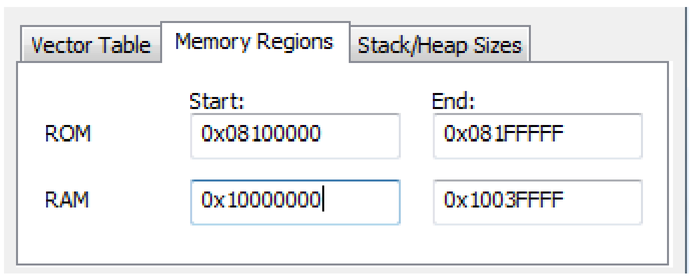
Figure 14. Linker → Config Tab



To change the linker configuration;

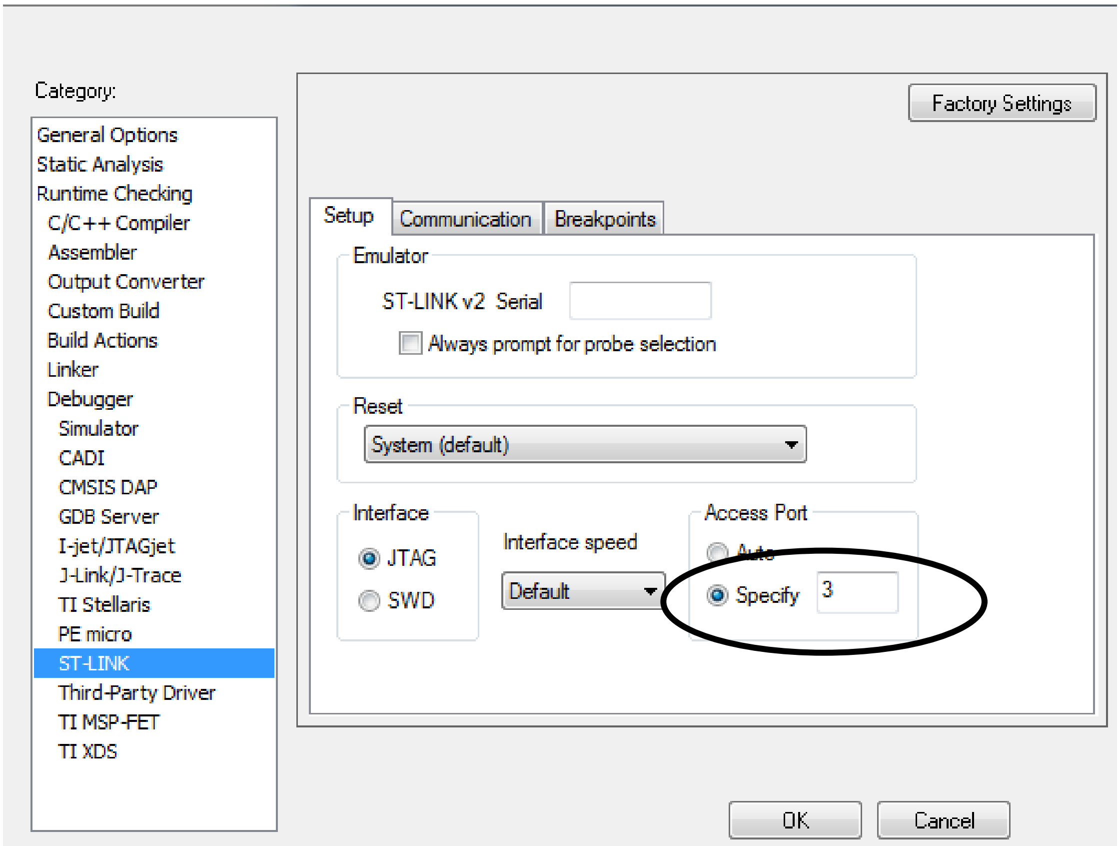
– Click Edit to display the linker configuration file editor.
– Open the Vector Table tab and set the **intvec** start variable to 0x08100000 .
– Open the Memory Regions tab, and enter the variables (Figure 15).

Figure 15. Memory Regions tab



3. From **Project** → **Options** → **ST-LINK** → **Setup** select the Access Port: AP 3 for CM4 or by selecting the Auto option (Figure 16).

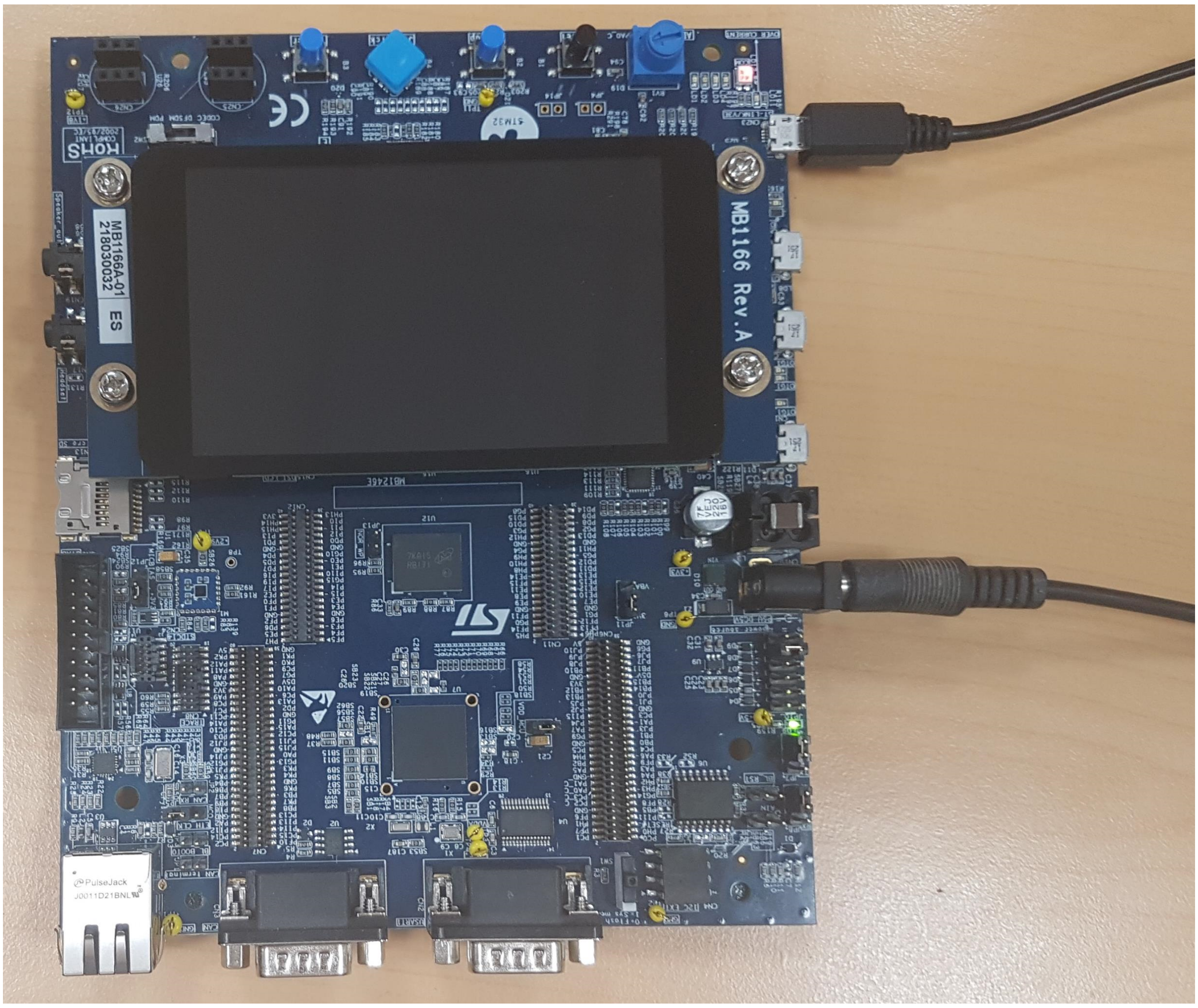
Figure 16. ST-LINK → Setup



4. Before downloading the project, connect to the STM32H747I-EVAL board:
   – Power the board up using an external power and ensure that the jumper (jp20) connected to the PSU pins.
   – Connect the ST-LINK/V2-1 programming and debugging tool on the STM32H747I-EVAL board. Plug the USB cable to the CN23 ST-LINK/V2-1 USB connector of the board: LD12 lights up red when the ST-LINK is connected.

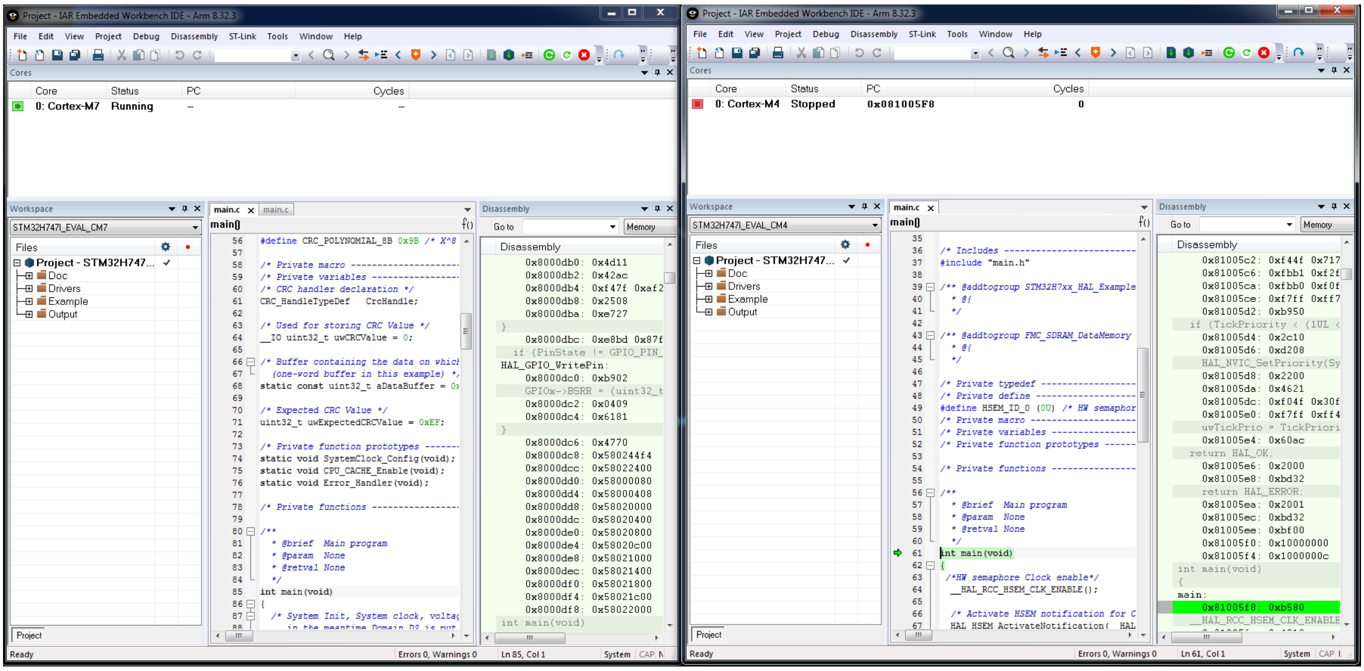
   Figure 17 shows the STM32H747I-EVAL board with a USB cable that is connect to an ST-LINK which is outside the picture .

Figure 17. Hardware environment



5. Start a debug session by clicking the Download and Debug button in the toolbar to program the flash memory and start debugging
   The core window is available from the View menu: both cores can be started and stopped individually or simultaneously from each instance. Each instance of the IDE displays debug information for the respective core as illustrated in Figure 18.

Figure 18. Start debug session



4.3 Steps for dual debug with I-jet™

This section provides step by step instructions on how to work with the STM32H7x5/x7 microcontrollers using EWARM v8.32.3 and I-jet™ as a debug probe.

The development environment allows both cores to be run individually or simultaneously from each IDE instance. Everything is therefore performed in one single development environment.

Two different use cases are outlined in Section 4.3.1 and Section 4.3.2.

4.3.1 Symmetric multicore debugging

This is debugging a symmetric device using only one instance of the IDE. The debug windows (watch window, memory windows and so on) display the information that is related to the core being focused on.

4.3.2 Asymmetric multicore debugging

Debugging an asymmetric device using one IDE instance for each core, one acting as master and the other acting as a slave.

Note: Master and Slave are just naming used to define which project starts first.

In this example, one project is created for each core.

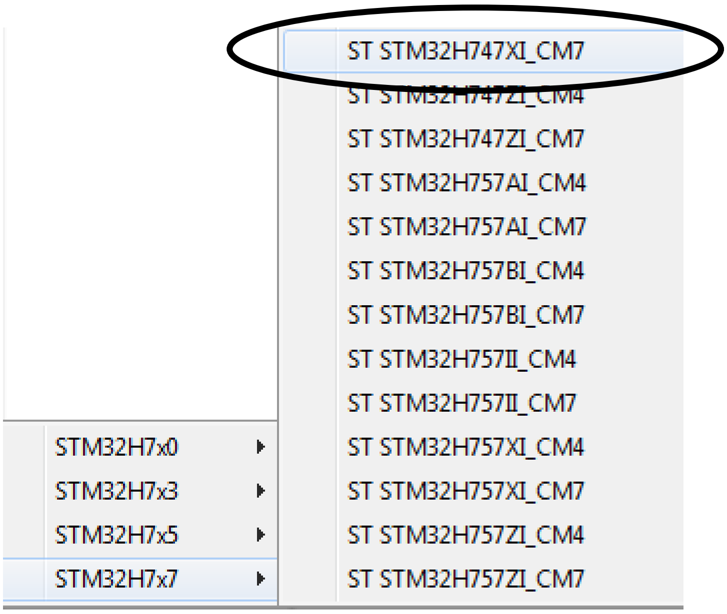
I-jet™ probe project

To set up for a multicore debugging using an I-jet™ probe:

1. Open IAR Embedded Workbench® and the application. Choose CM7 as a master and CM4 as a slave. Start with the master project configuration (CM7).

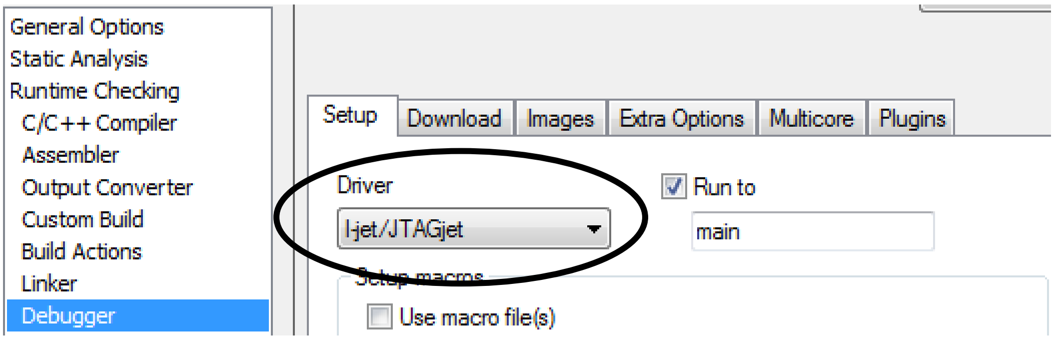
2. Choose the correct device **Project** → **Options** → **Target** → **Device** (Figure 19).

Figure 19. Target device



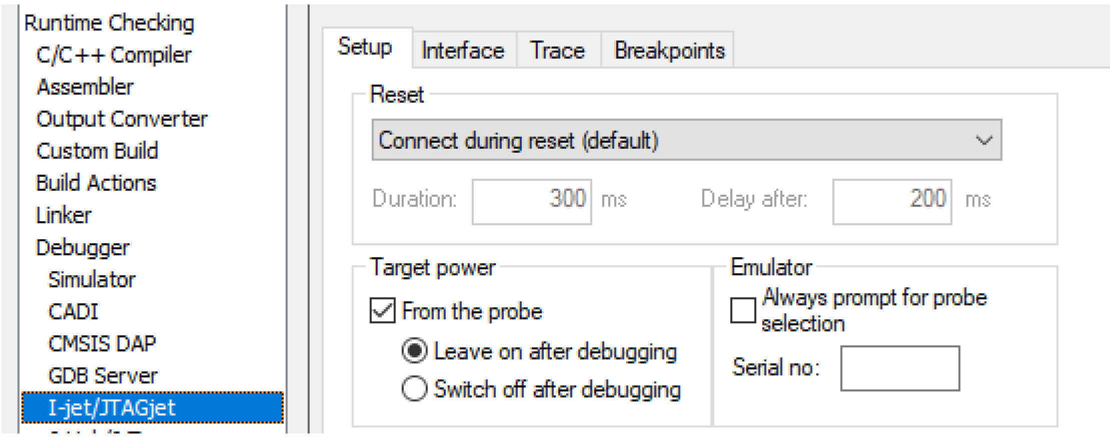
3. Select I-jet as a debugger from: **Project** → **Options** → **Debugger** → **Setup** (Figure 20).

Figure 20. Debugger → Setup tab



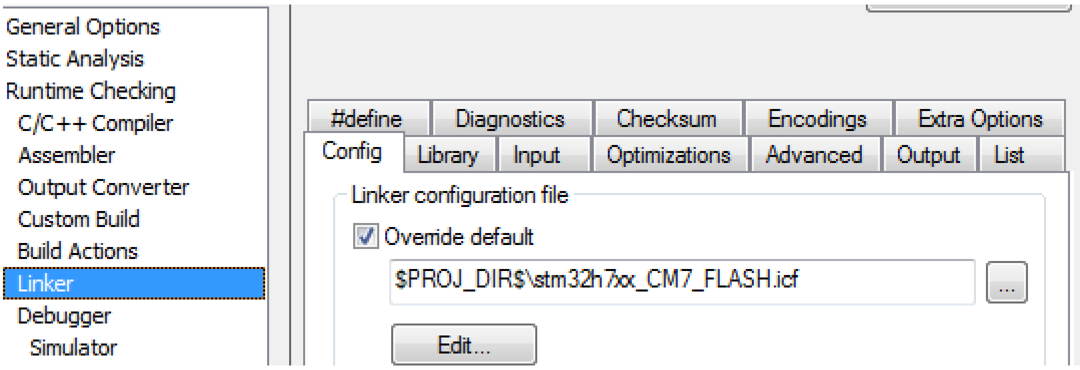
4. Select the Reset type from **Project** → **Options** → **I-jet/JTAGjet** → **Setup** (Figure 21). For more information on the reset type refer to Section 4.2.

Figure 21. I-JET/JTAGJET → Setup tab



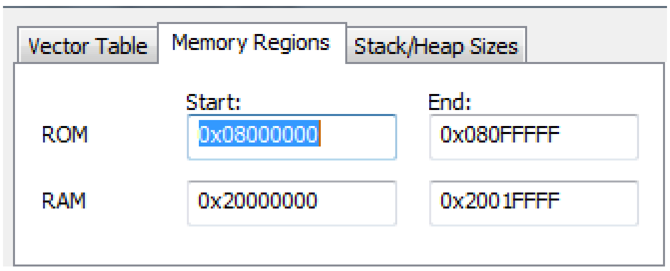
5. Go to **Project** → **Options** → **Linker** → **Config**. Check the linker configuration file to make sure the application is linked to the correct address (Figure 22).

Figure 22. Linker → Config tab



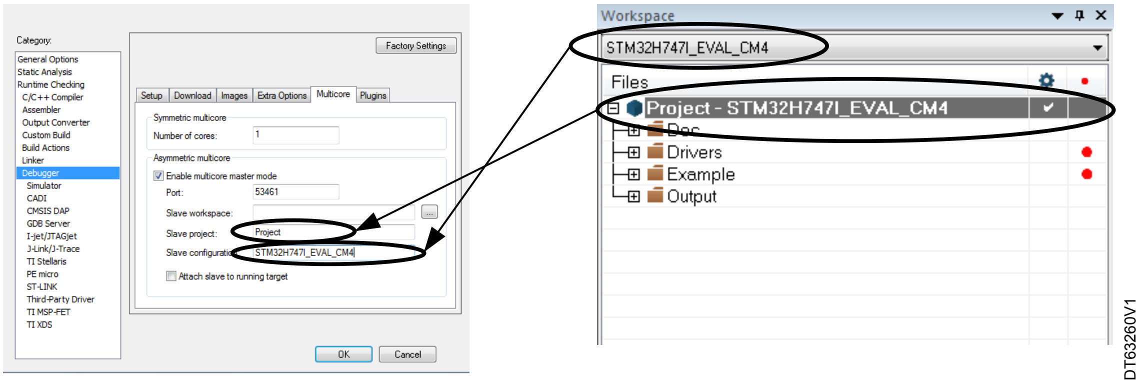
Click Edit to display the Linker configuration file editor. Check the linker configuration file to ensure the application is linked to the right address Open the Memory Regions tab, and enter the variables as shown in Figure 23.

Figure 23. Memory Regions tab



6. Choose **Project** → **Options** → **Debugger** → **Multicore** and select Enable multicore master mode (Figure 24).

Figure 24. Debugger → Multicore tab



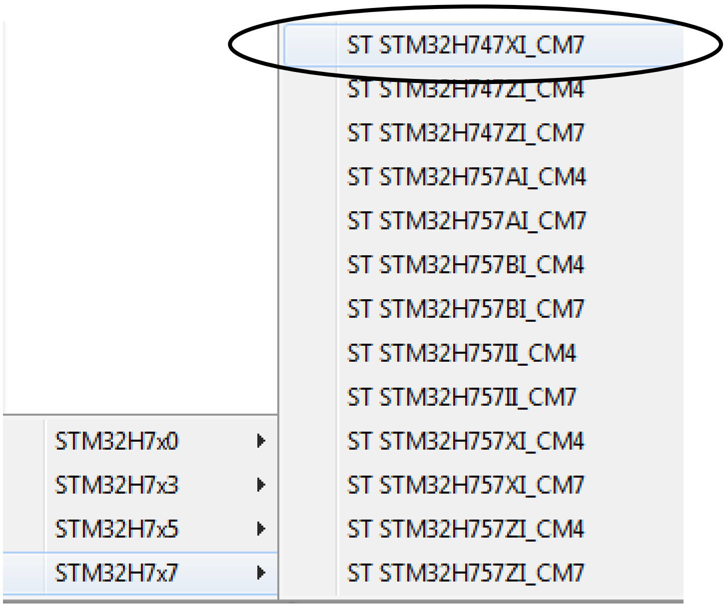
Specify the workspace path, project name, and configuration name to use when starting the slave session

CM4 slave configuration

To configure the Slave (CM4) follow the same steps mentioned above for CM7 and make sure to use compatible settings are used for the debug probe on both projects

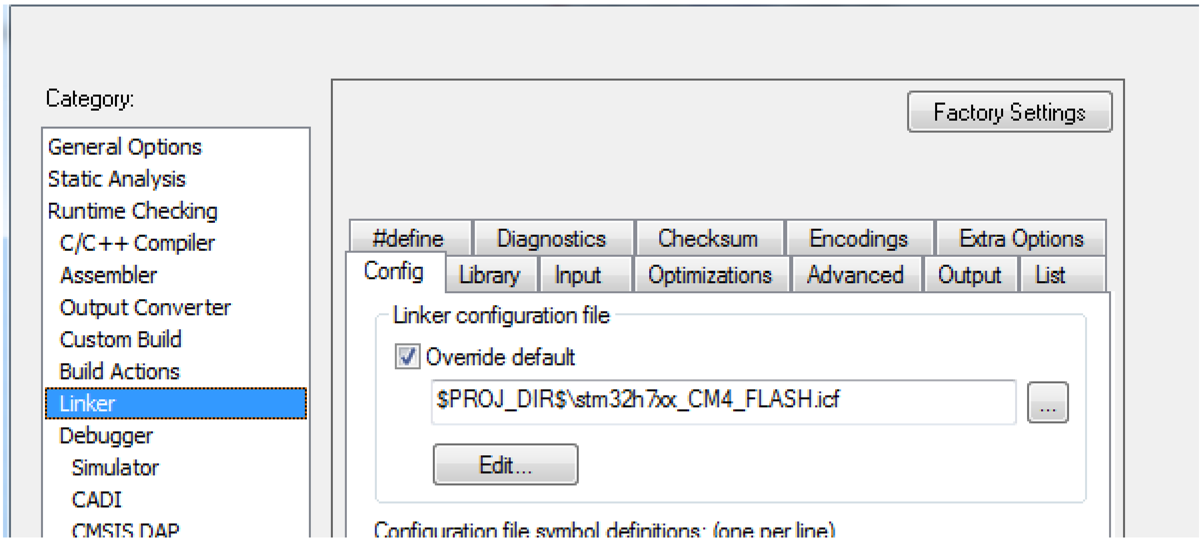
1. Choose the correct device **Project** → **Options** → **Target** → **Device** (Figure 25).

Figure 25. Device selection



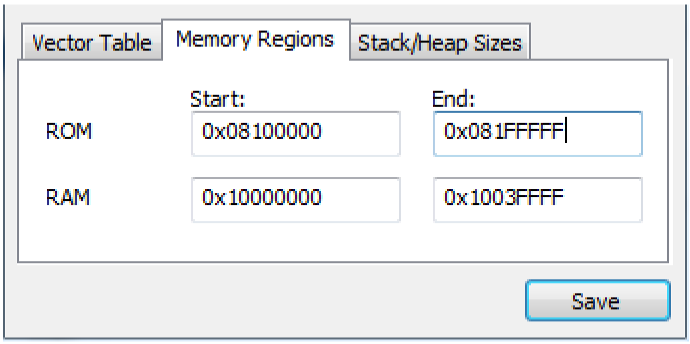
2. Go to **Project** → **Options** → **Linker** → **Config** (Figure 26).

Figure 26. Linker → Config tab



3. Click Edit to display the Linker configuration file editor. Open the Memory Regions tab, and enter the variables as shown in Figure 27.

Figure 27. Memory Regions tab



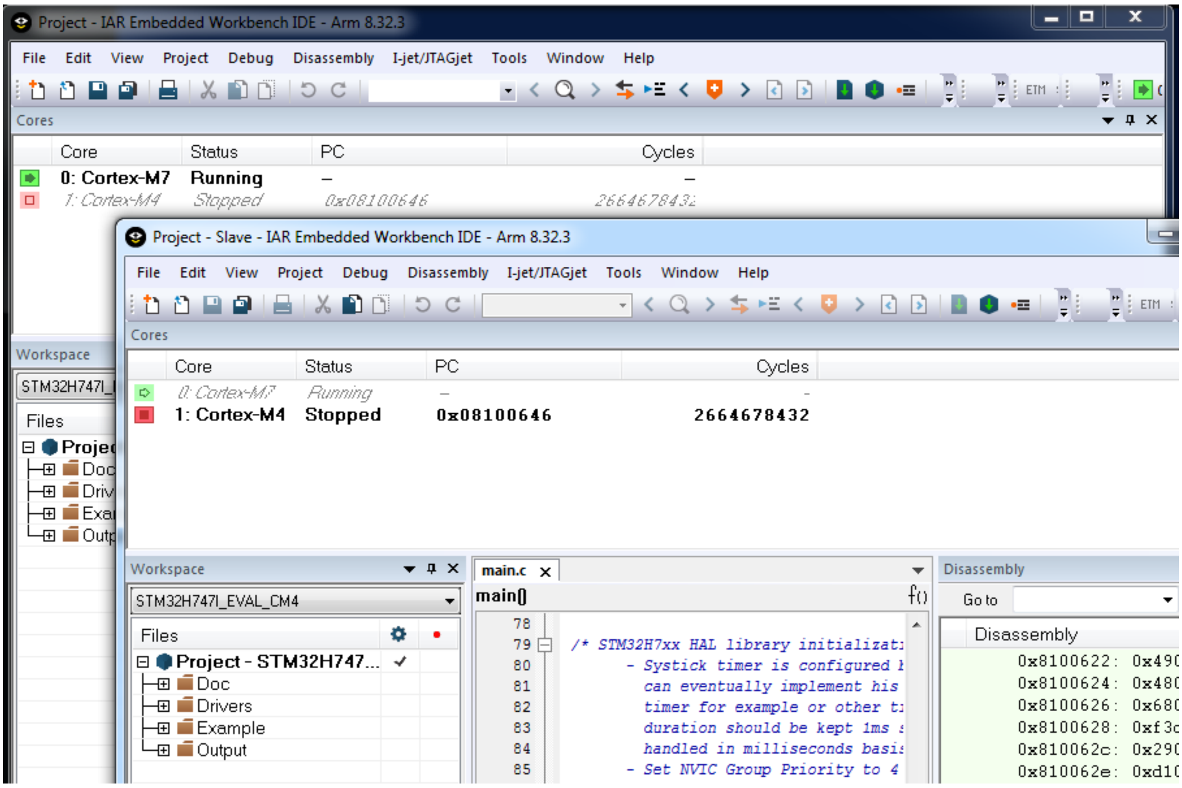
4. Before downloading the project, first setup the connection with the STM32H7xxI-EVAL board with the following procedure:
   – Power the board up using an external power and ensure that the jumper (JP20) is connected to the PSU pins.
   – Connect the I-jet programming and debugging tool on the STM32H7xxI-EVAL board. Plug the JTAG cable to the CN9 (JTAG) connector of the board.

   Figure 28 shows how to connect the I-jet™ to the STM32H7xxI-EVAL board.

Figure 28. Hardware environment

5. Open the project (Figure 29):

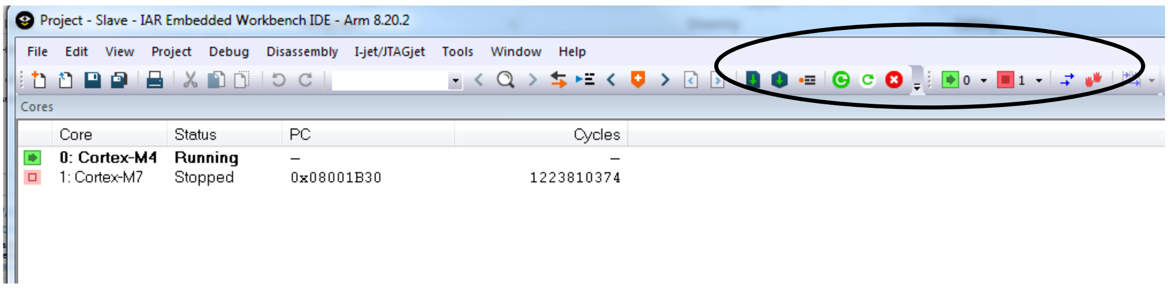
Figure 29. **Debug session**

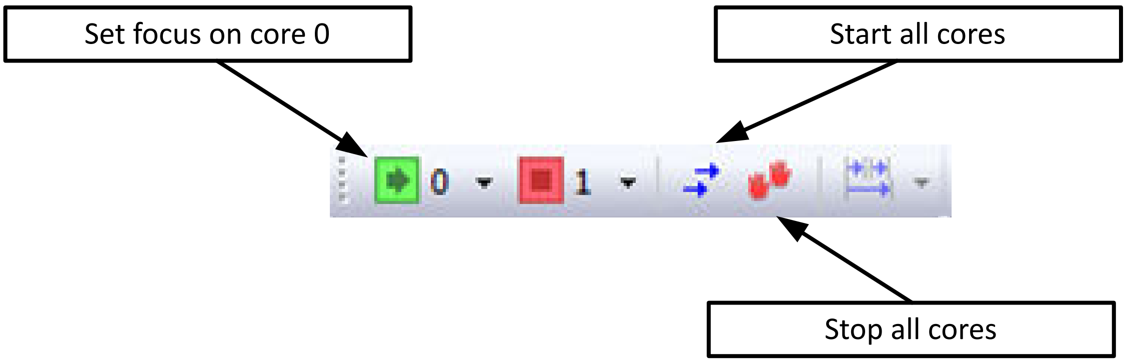


6. Start a debug session by clicking the Download and Debug button. The master and slave instances are indicated in the main window as shown in Figure 29 above. The whole application can be run in several different modes:
   – Both cores run simultaneously.
   – The cores run independently of each other if needed.

   In both cases, the process can be stopped and started automatically as needed.
   Each IDE instance displays the debug information for the respective cores. Each core windows display the status of the available cores, and the information such as the respective core execution states. The core toolbar is a complement to the core window. The toolbar is illustrated in Figure 30. When a break-point is set, it is relative to one core only and when the break-point is triggered, that core is stopped.

Figure 30. Toolbar menu





4.4 Access port switch

EWARM and I-jet™ offers the possibility of switching between the different access ports from the debug session to memory visualization and other domain peripheral.

Go to View → memory → memory1 (Figure 31).

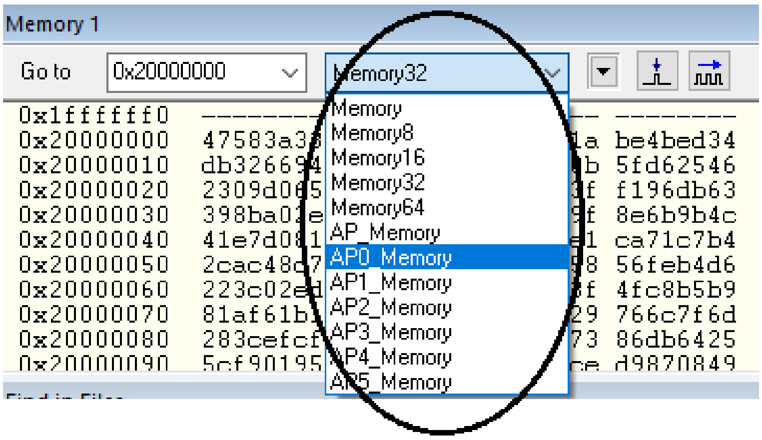


Figure 31. View → Memory tab

4.5 Embedded cross trigger (ECT) on EWARM

In this example, the CTI is configured to stop both cores simultaneously.

CM7 -project configuration

Configure the CM7-project and CM4-project by using the procedure Section 4.2.

Create a macro file (.mac) to configure the CTI for each core making sure the macro file must contain the following steps:

1. Enable the `CTI_CMx`
2. Connect `CTITRIGIN0` (Halt) to one of CTM channels (Channel 0 in this example)
3. Connect `CTITRIGOUT0 (EDBGRQ)` to CTM channel (Channel 0 in this example):

```
CTI_Config()
  {
    writeMemory32(0x1,0xe0043000,"Memory"); // Enable CTI_CM4
    writeMemory32(0x1,0xe0043020,"Memory"); // // Enable TRIGIN0 (HALTED) to channelx
    writeMemory32 (0x1, 0xe00430a0, “Memory”); // Enable channel0 to TRIGOUT0 (EDBGRQ)
  }
```

To run both cores simultaneously, the debugger must use the APPPULSE register in both CTIs.
The debugger then generates a pulse on any of the four CTM channels.
To generate a stop request, generate a pulse on channel 0 by writing 0x01 to the APPULSE register in either CTI. Example of the sample code is given here:

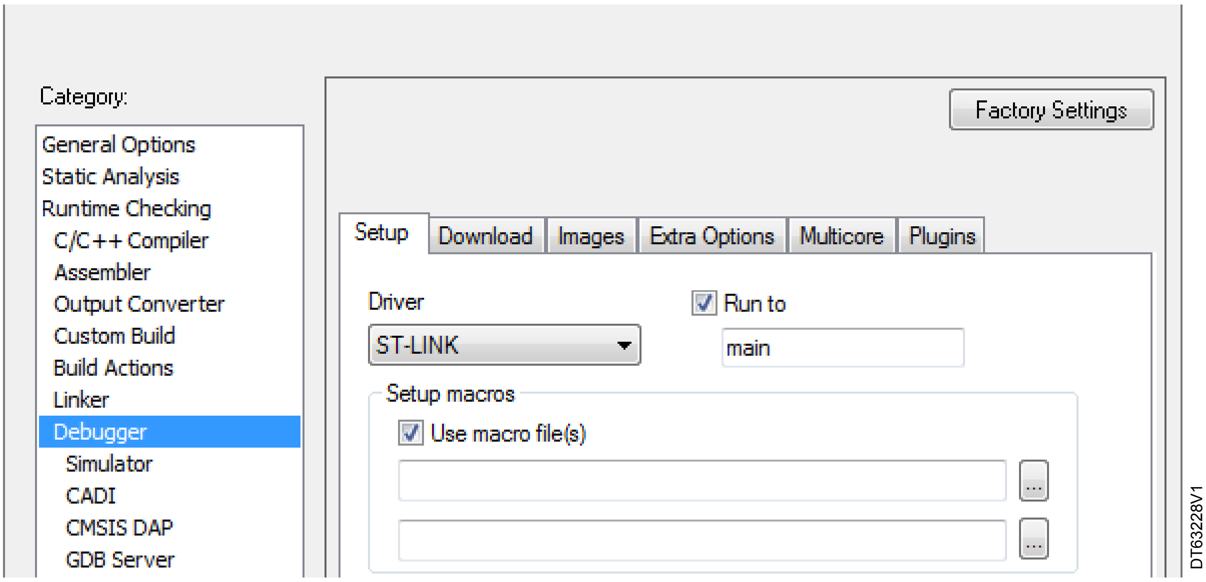
```
HaltBothCores()
  {
    if( __readMemory32(0xe0043000,"Memory") == 1)
      {
        message "Run both cores using CTI_M7 APPPulse on CH0.\n";
        writeMemory32(0x1,0xe004301C,"Memory"); // APPPulse on CH0
      }
    else
    message "Error: CTI_M7 is not enabled!\n";
  }
```

4. And finally, execute the functions mentioned above using the routine below:

```
execUserExecutionStopped()
  {
    CTI_Config();
    HaltBothCores ();
  }
```

Go to **Project** → **Options** → **Debugger** → **Setup** → **Setup macros** and add the macro file created above by checking Use macro files (Figure 32).

Figure 32. **Options → Debugger tab**



To configure the CM4-project follow the same steps mentioned above for CM7_project.

Establish the connection with the STM32H7xxI-EVAL board then download the applications from **Flash** → **Download**.

Start a debug session from **Debug** → **Start/Stop Debug Session**.

Both cores must run simultaneously.

5 Using MDK-ARM

The latest version of MDK-ARM (Keil®) is available for download from the official web site of Arm® Keil®.

MDK-ARM (Keil®) is installed by default in the C:\Keil directory on the PC local hard disk; the installer creates a start menu µVision® 5 shortcut.

5.1 Dual debugging on MDK-ARM

As described above, the STM32H7x5/x7 microcontrollers integrate a multi-core system consisting of Arm® Cortex®-M7 and Cortex®-M4 processors as described in the introduction.

This asymmetric multi-core system requires additional features from the development tools, in particular the debugger. These features are supported in MDK-ARM IDE from v4.14

5.2 Steps for dual debug with ST-LINK V2/V3

This section provides a step by step set of instructions to work with the STM32H7x5/x7 microcontrollers using MDK-ARM v5.27.0.0 and ST-LINK V2/V3 debug probes.

Note: Dual debug is supported using MDK-ARM version v5.25 and later.

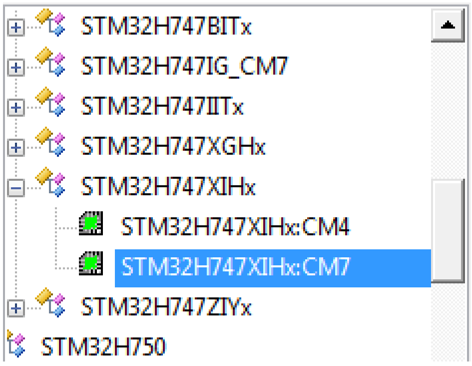
In this example, one project is created for each core.

To set up for a multicore debugging using an ST-LINK V2/V3. Install the latest STLINK server available on www.st.com.

Procedure for using MDK-ARM with the STM32H7x5/x7 microcontrollers with ST-LINK/V2 debug probe.

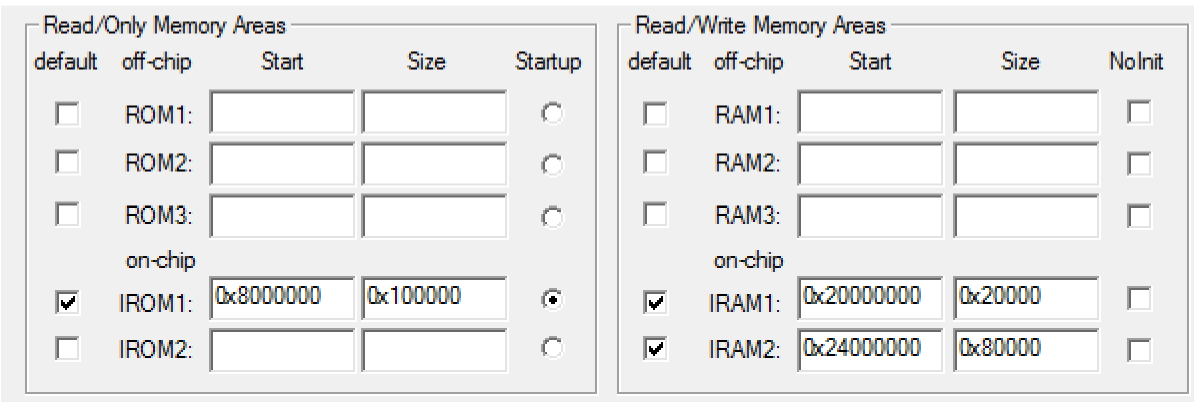
1. Starting with the CM7_project configuration:
   a. Open MDK-ARM Keil® and create a new project.
   b. Select the correct device from **Project → Options for Target → Device** (Figure 33).

Figure 33. **Options for Target → Device tab**



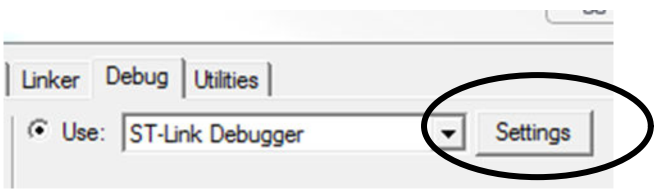
   c. Ensure the right memory area is selected from **Project Options for Target →Target** (Figure 34):
      - CM7 Boot address 0: Flash at 0x08000000
      - CM7 Boot address 1 : DTCM-RAM at 0x20000000.

Figure 34. **Options for Target → Target tab**



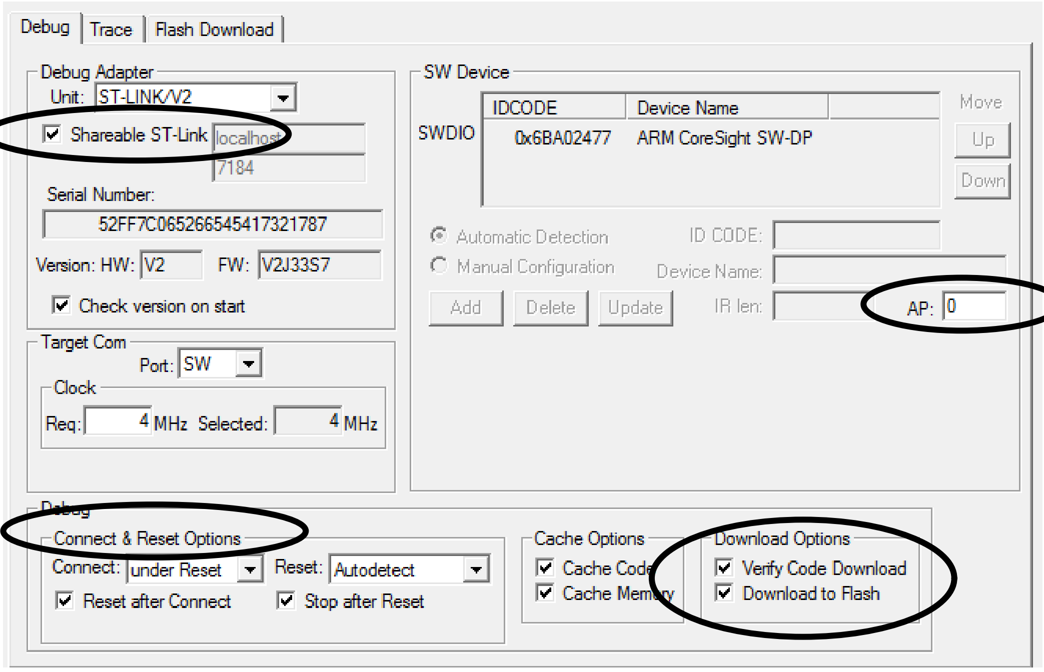
   d. Select ST-LINK as a debugger from: **Project → Options for Target → Debug** (Figure 35).

Figure 35. **Options for Target → Debug**



   e. From debug settings (Figure 36):

Figure 36. Debug settings



i. *Enable the dual-core debug by checking the option Shareable ST-LINK.*

ii. *Select the Access Port: AP 0 for CM7.*

iii. *Select the downloads options from the same window.*
   - *Verify code download: enabling this option stops the CPU after the currently executed instruction .*
   - *Download to Flash: download code to all memory area.*

iv. *Select the connection options:*
   - *Normal: stops the CPU at the currently executed instruction after connecting.*
   - *With pre-reset: applies a hardware reset before connecting to the device.*
   - *Under Reset: holds the hardware reset signal active while connecting to the device.*

v. *Select the Reset options:*
   - *Auto-detect: the system selects the best suitable reset method for the target device.*
   - *HW RESET: performs a hardware reset by triggering a hardware reset signal.*
   - *SYSRESETREQ: performs a software reset by setting the SYSRESETREQ bit. The Cortex-Mx core and on-chip peripherals are reset.*
   - *VECTRESET: performs a software reset by setting the VECTRESET bit. Only the core is reset.*

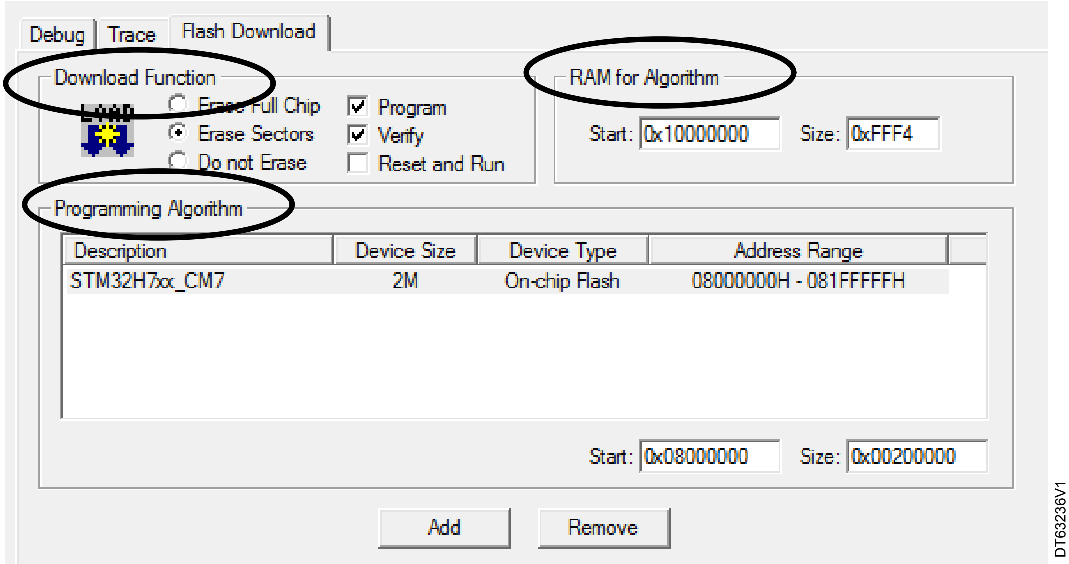
f. From the Flash Download window (Figure 37) select:

i. Download function: to set the Flash operations.

ii. RAM for algorithm: defines the address space where programming algorithms are loaded and executed. Usually, the address space is located in the on-chip RAM.

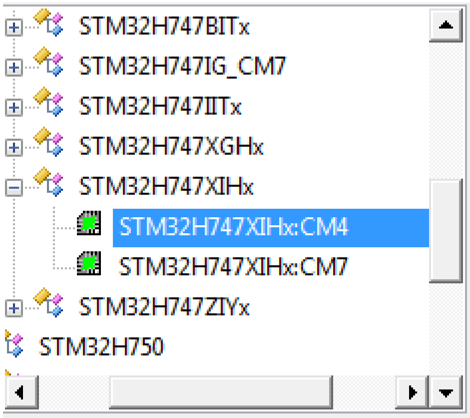
iii. Program algorithm: contains the definitions for programming Flash.

Figure 37. Flash Download Tab



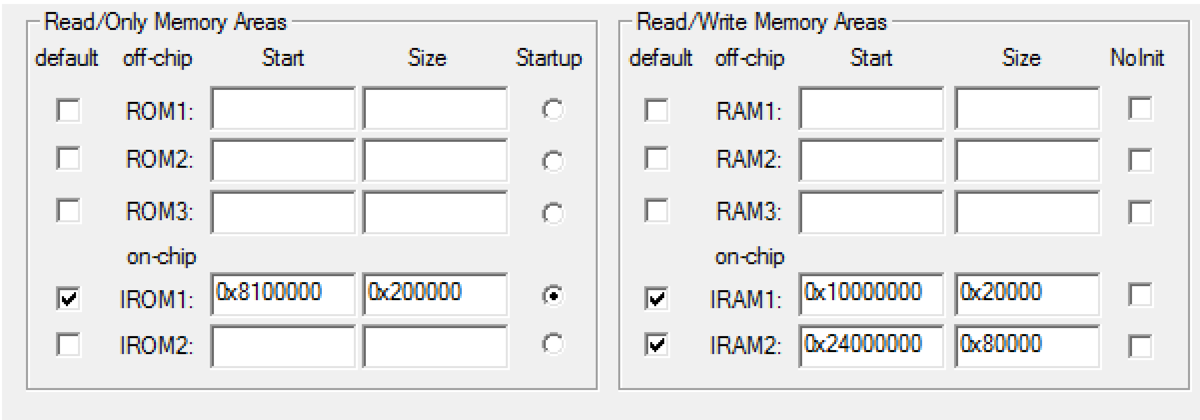
2. CM4-configuration, follow the same steps mentioned above for CM7 and make the changes stated below:
   a. Select the correct device from **Project** → **Options for Target** → **Device** (Figure 38).

Figure 38. Device selection



   b. Make sure the right memory area is selected from **Project** → **Options for Target** → **Target** (Figure 39):

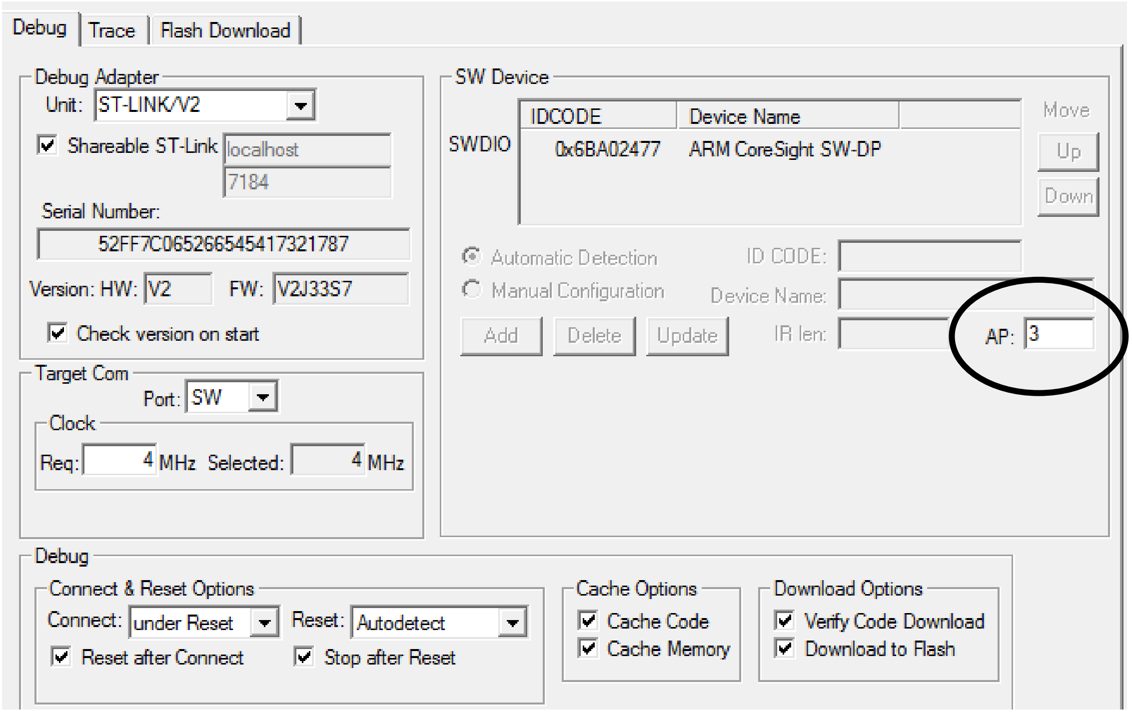
Figure 39. Options for Target → Target Tab



- CM4 boot address 0: Flash memory at 0x08100000
- CM4 boot address 1: SRAM1 at 0x10000000.

   c. From debug settings (Figure 40):

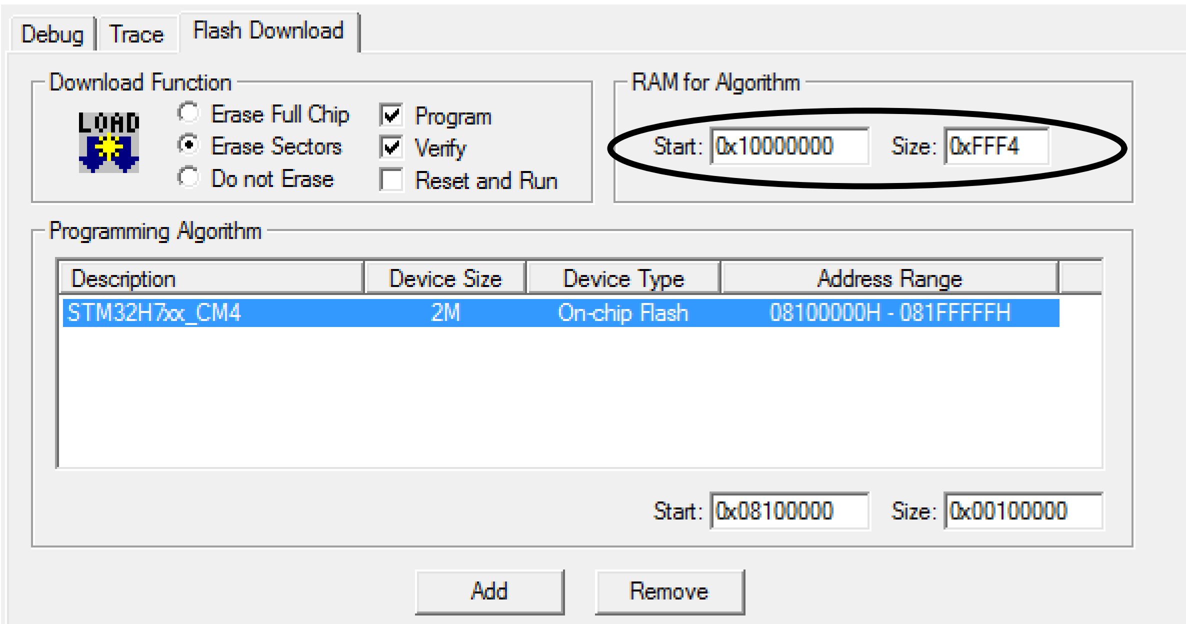
Figure 40. **Debug settings tab**



- Enable the dual-core debug by checking the option Shareable ST-LINK.
- Select the Access Port: AP 3 for CM4.

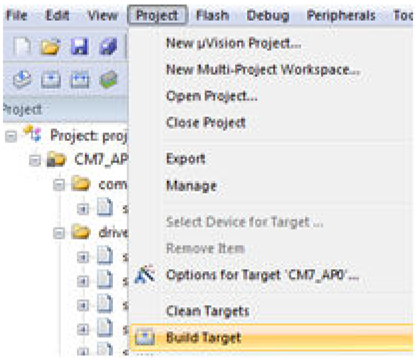
d. From the Flash Download window (Figure 41) change the boot address to SRAM1 0x10000000.The address 0x20000000 is not accessible by CM4.

Figure 41. **Flash download tab**



e. Build each project from **Project** → **Build Target** (Figure 42).

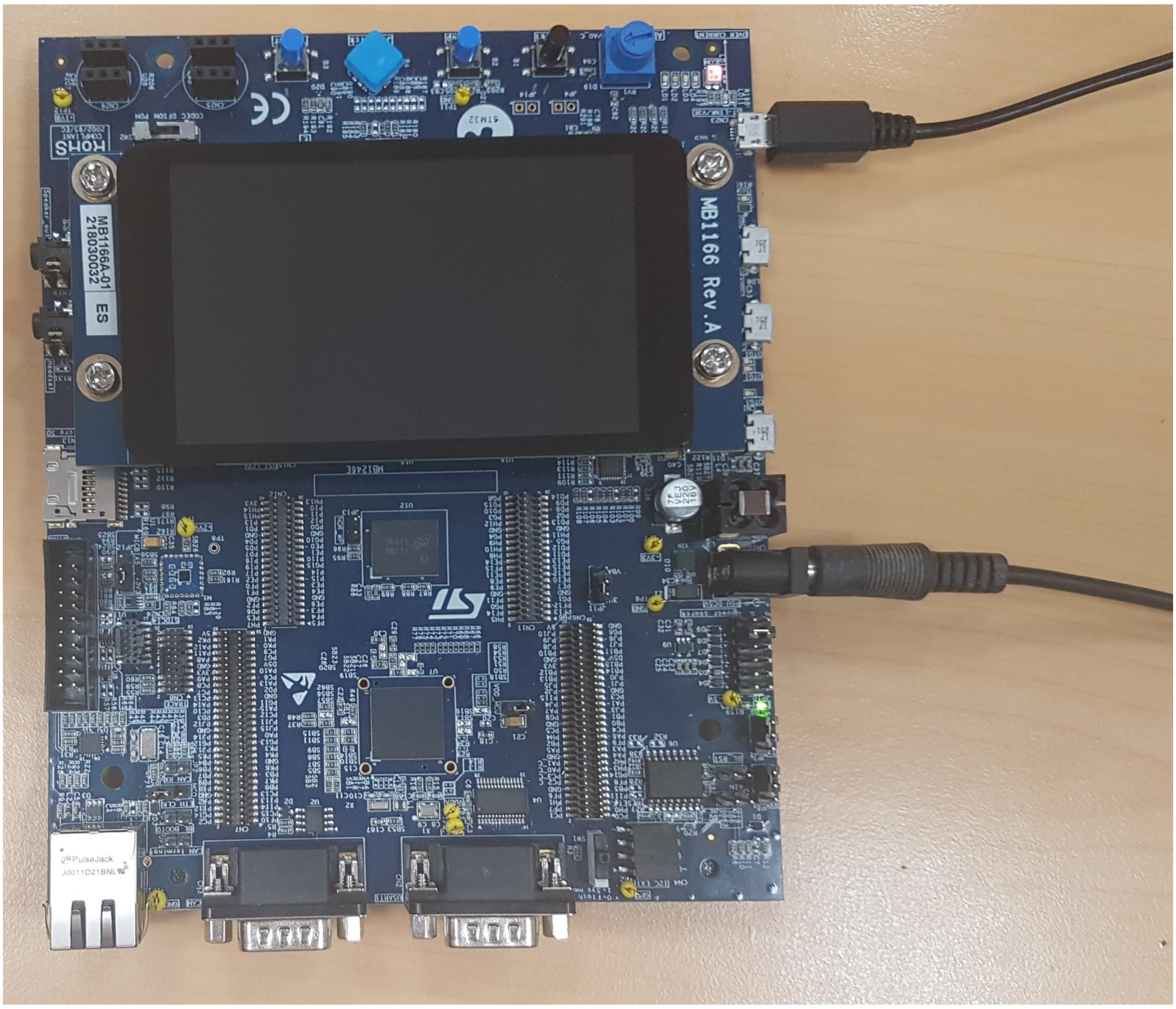
Figure 42. **Build the project**



f. Before downloading the projects, establish a connection with the STM32H7xxI-EVAL board as follows:
   i. Power the Board up using an external power supply and ensure that the jumper(jp20) connect the PSU pins.
   ii. Connect the ST-LINK/V2-1 programming and debugging tool on the STM32H7xxI-EVAL Board. Plug the USB cable to the CN23 ST-LINK/V2-1 USB connector of the board: LD12 lights up red when the ST-Link is connected.

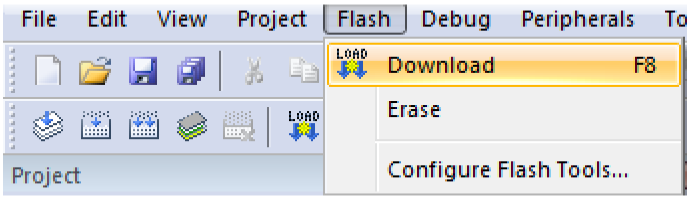
   Figure 43 illustrates the board set up.

Figure 43. **Hardware environment**



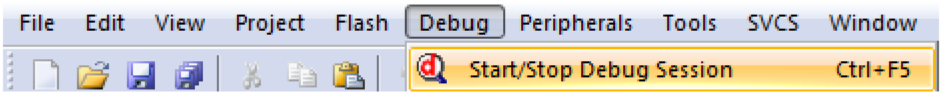
g. Download the applications from **Flash** →**Download** (Figure 44).

Figure 44. **Download the application**



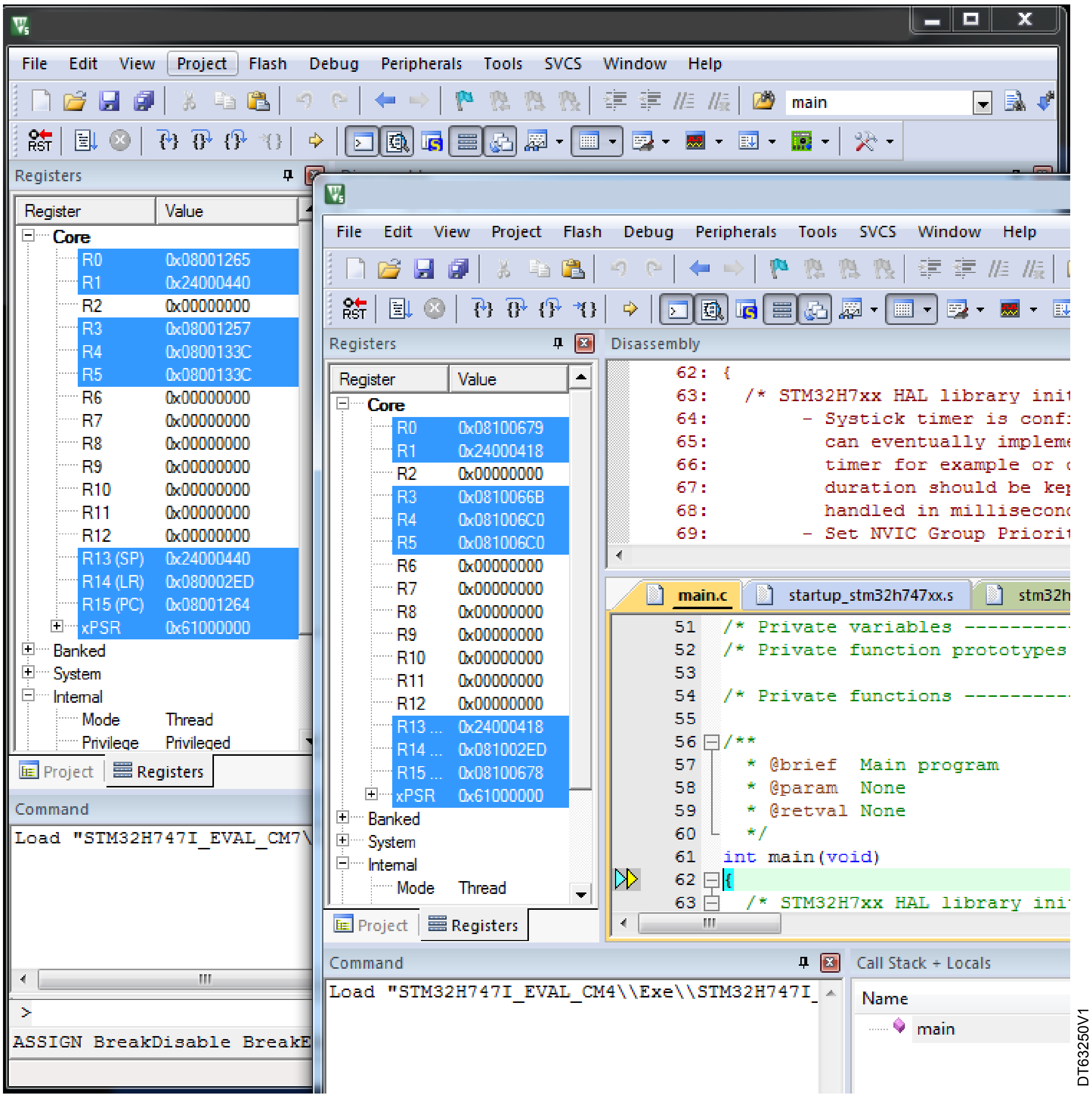
h. Start a Debug session from **Debug** → **Start/Stop Debug Session** (Figure 45):

Figure 45. **Start/Stop Debug Session**



Both cores are run individually or simultaneously from each of the GUI instances. Each instance of the IDE displays debug information for the connected core (Figure 46).

Figure 46. Debug session



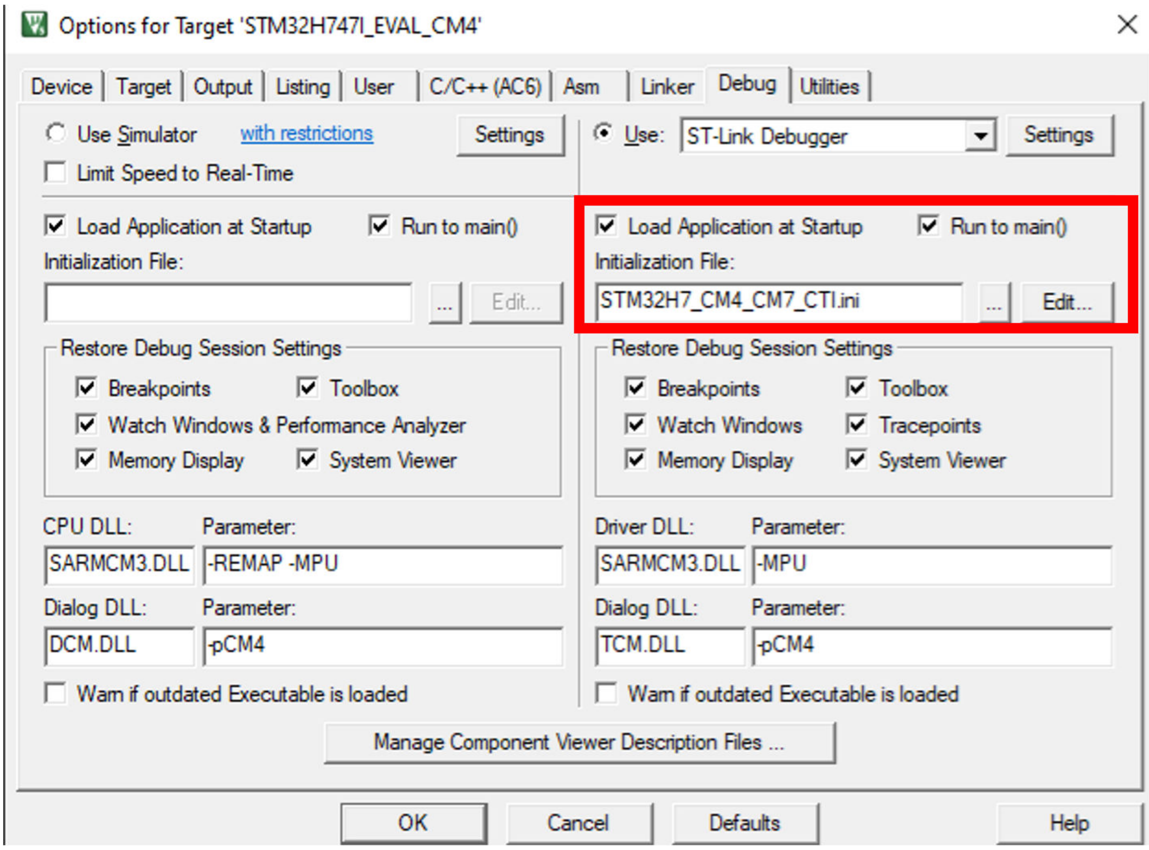
Note: *Same steps described above for ST-LINK are applicable when using ULINK probe.*

5.3 Embedded cross trigger (ECT) on MDK-ARM

To configure the CM7-project and CM4-project follow the same steps mentioned in Section 5.2.

Create an initialization file from **Project** → **Options** for **Target** → **Debug** that contains the CTI configuration for each core as shown in Figure 47.

Figure 47. **Options for Target → Debug tab**



DT63251V1

The initialization file STM32H7_CM4_CM7_CTI.ini must be located at MDK-ARM project level.

Refer to the following link to understand how to create an MDK-ARM debugger script .ini files and understood the syntax:

https://developer.arm.com/documentation/ka002991/latest

The file STM32H7_CM4_CM7_CTI.ini contains the following functions:

- Functions CTI_ConfigM4() and CTI_ConfigM7() for each core and allow to:
  - Enable cross trigger inputs bit and set GLBEN in CTI_CONTROL.
  - Enable and connect CTITRIGIN0 (HALTED) to one of CTM channels1.
  - Enable and connect CTITRIGOUT0 (EDBGRQ) to CTM channel0.
  - Enable and connect CTITRIGOUT7 (DBGRESTART) to CTM channel1.
- Functions ClearFlags() used to perform a trigger acknowledge flags to all CTITRIGOUT into CTI_INTACK register.

To run and halt both cores simultaneously the debugger must use the APPPULSE register in either of the CTIs. This allows the debugger to generate a pulse on any of the four CTM channels. To generate a stop request, create a pulse on channel 0 by writing 0x01 to the APPULSE register in either CTI.

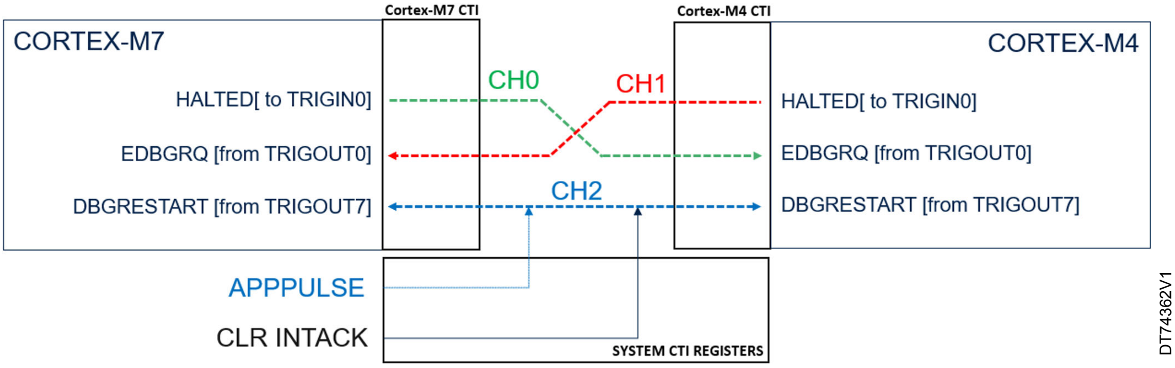
- Functions RunBothCores() used to generate a trigger acknowledge and a pulse on APPPULSE on channel 2.
- Function HaltBothCores to generate a trigger acknowledge and a pulse on APPPULSE channel 0 and channel 1.

Note:

```
FUNC void CTI_ConfigM4(void)
{
_WDWORD(0xe0043000,0x1); // CTI_M4 Enable
_WDWORD(0xe0043020,0x2); // Enable TRIGIN0 (HALTED) to channel1
_WDWORD(0xe00430a0,0x1); // Enable channel0 to TRIGOUT0(EDBGRQ)
_WDWORD(0xe00430BC,0x4); // Enable channel1 to TRIGOUT7(DBGRESTART)

printf("CTI_M4 configuration is done!\n");
}

FUNC void CTI_ConfigM7(void)
{
_WDWORD(0xe0043000,0x1); // CTI_M7 Enable
_WDWORD(0xe0043020,0x1); // Enable TRIGIN0 path (HALTED) to channel0
_WDWORD(0xe00430a0,0x2); // Enable channel0 to TRIGOUT0 (EDBGRQ)
_WDWORD(0xe00430BC,0x4); // Enable channel1 to TRIGOUT7 (DBGRESTART)

printf("CTI_M7 configuration is done!\n");
}

FUNC void ClearFlags(void)
{
if( _RDWORD(0xe0043000) == 1)
{
printf("Clear Flags.\n");
_WDWORD(0xe0043010,0xFF); // Trigger acknowledge clear high
_WDWORD(0xe0043010,0x00); // Trigger acknowledge clear low
}
else
printf("Error: CTI_M7 or CTI_M4 is not enabled!\n");
}

FUNC void RunBothCores(void)
{
if( _RDWORD(0xe0043000) == 1)
{
printf("Run both cores using CTI APPPulse on CH2.\n");
_WDWORD(0xe0043010,0xFF); // Trigger acknowledge clear high
_WDWORD(0xe0043010,0x00); // Trigger acknowledge clear low
_WDWORD(0xe004301C,0x4); // APPPulse on CH2
}
else
printf("Error: CTI_M7 or CTI_M4 is not enabled!\n");
}

FUNC void HaltBothCores(void)
{
if( _RDWORD(0xe0043000) == 1)
{
printf("Halt both cores using CTI APPPulse on CH0 & CH1.\n");
_WDWORD(0xe0043010,0xFF); // Trigger acknowledge clear high
_WDWORD(0xe0043010,0x00); // Trigger acknowledge clear low
_WDWORD(0xe004301C,0x3); // APPPulse on CH0 and CH1
}
else
printf("Error: CTI_M7 or CTI_M4 is not enabled!\n");
}
```

The following figure shows how are interconnected CTI internal signals between them:

Figure 48. **CTI internal connections**



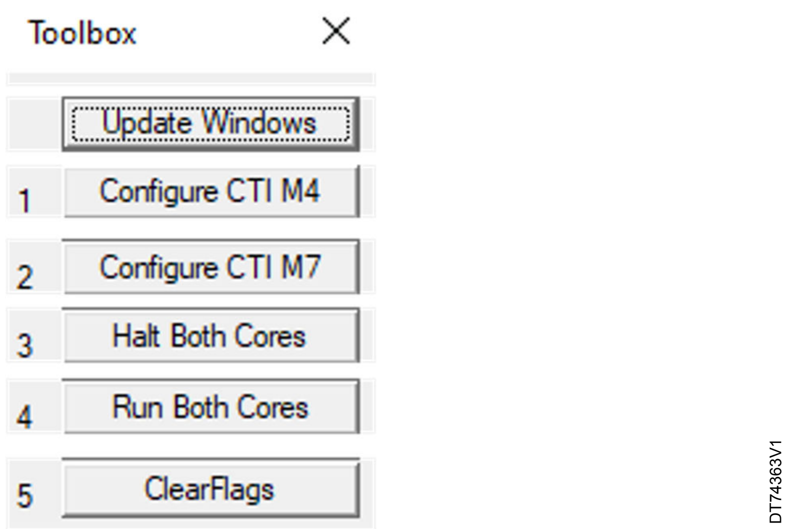
Finally, at the end of STM32H7_CM4_CM7_CTI.ini, add the code below to define graphical buttons to configure CTI for both cores, run and halt both cores to the MDK-ARM toolbar menu.

```
DEFINE BUTTON "Configure CTI M4", "CTI_ConfigM4()"
DEFINE BUTTON "Configure CTI M7", "CTI_ConfigM7()"
DEFINE BUTTON "Halt Both Cores", "HaltBothCores()"
DEFINE BUTTON "Run Both Cores", "RunBothCores()"
DEFINE BUTTON "ClearFlags", "ClearFlags()"
```

To configure the CM4-project, follow the same steps mentioned above for CM7_project:

1. Download the applications from **Flash** → **Download**.
2. Start a Debug session from **Debug** → **Start/Stop Debug Session**.
3. From the debug session, the following **Toolbox** window opens and displays the dedicated button and UI.

Figure 49. **Debug session** → **Toolbox**

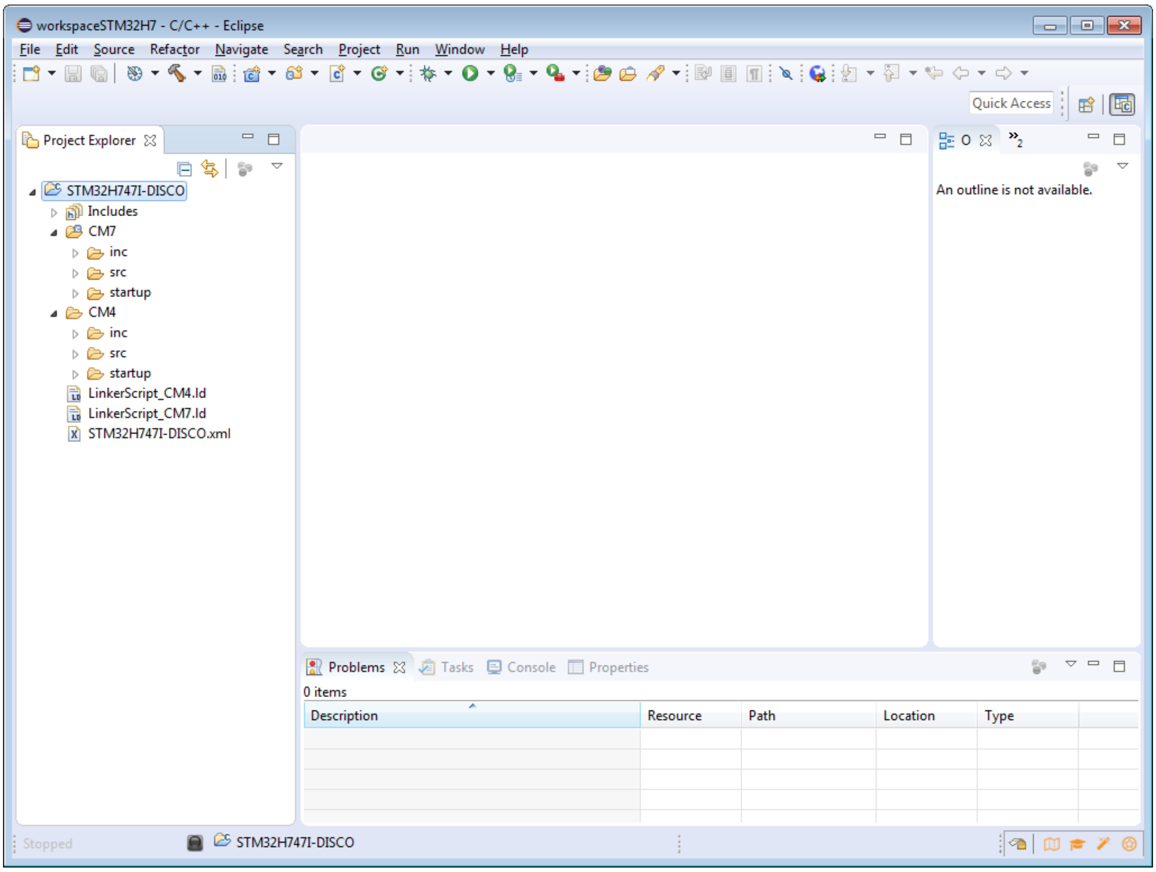


6 Using the STM32 system workbench

The latest version of system workbench for STM32 products (SW4STM32) is available to download from the official AC6 web site www.openstm32.org.

When creating a new project, both cores are included in a single project. A set of sources are dedicated to the CM4, another to the CM7 and where needed the user adds common sources built for both cores. The build generates two binaries, one for each core as shown in Figure 50.

Figure 50. Multicore project



6.1 Dual debugging on SW4STM32

System workbench for STM32 products offers the possibility to simultaneously debug two or more identical cores using either the symmetric multicore processing (SMP), or two cores with different architectures also known as asymmetric multicore processing (AMP). The debug is done either by using the same project for the multicore case (one board for X cores) or separate projects for different core configurations, each core/project using the shareable ST-Link to address each target.

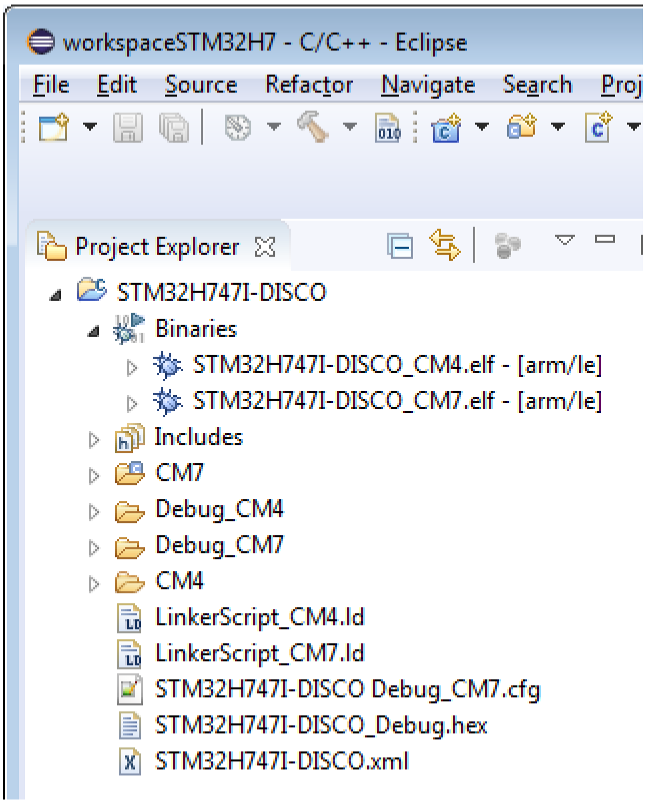
6.2 Steps for dual debug with ST-LINK V2/V3

This section provides step by step instructions to the STM32H7x5/x7 microcontrollers debug using SW4STM32 v2.9.0 and ST-LINK as a debug probe.

For this example, one project is created for both cores:

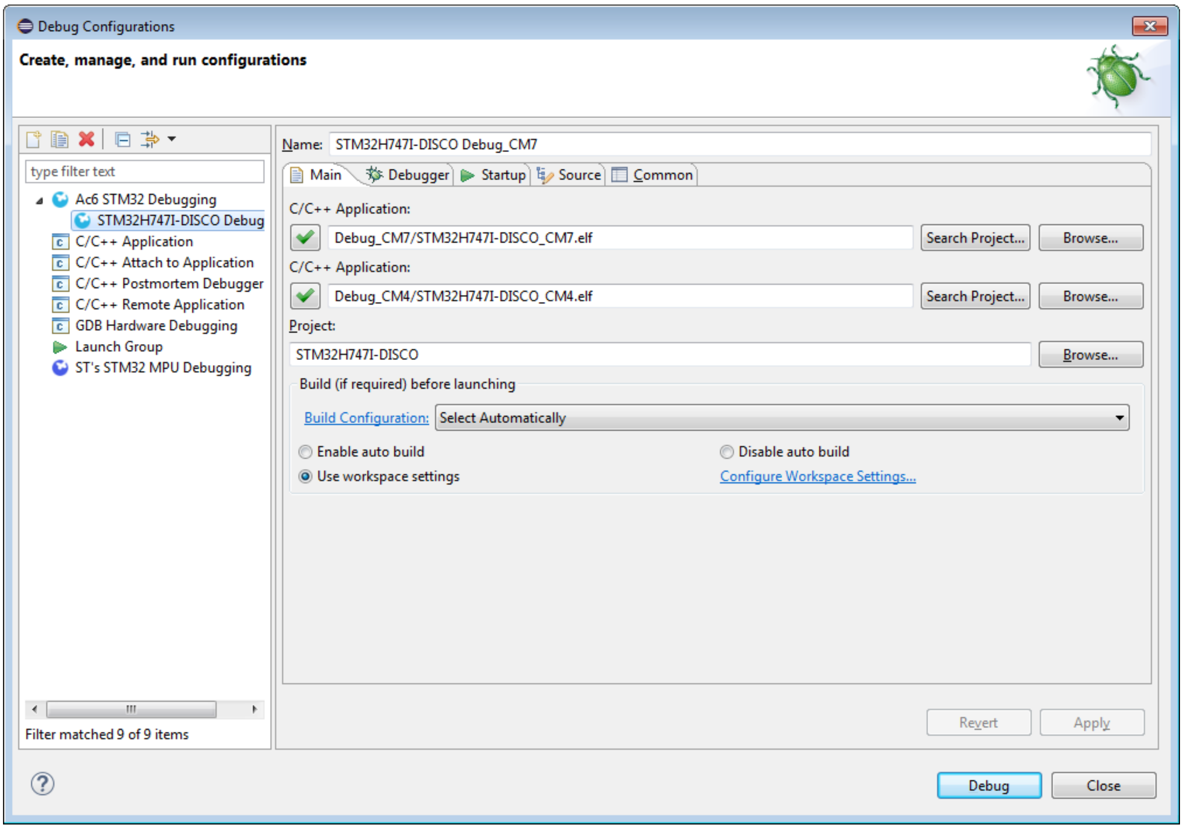
1. Open the STM32 system workbench and create one project for both cores.
2. Two binaries are generated by the compiler (Figure 51).

Figure 51. Multicore project: two binaries



3. Go to **Run** → **Debug Configurations** and double-click on Ac6 STM32 Debugging to create a new debug configuration (Figure 52):

Figure 52. Multicore project: debug configuration

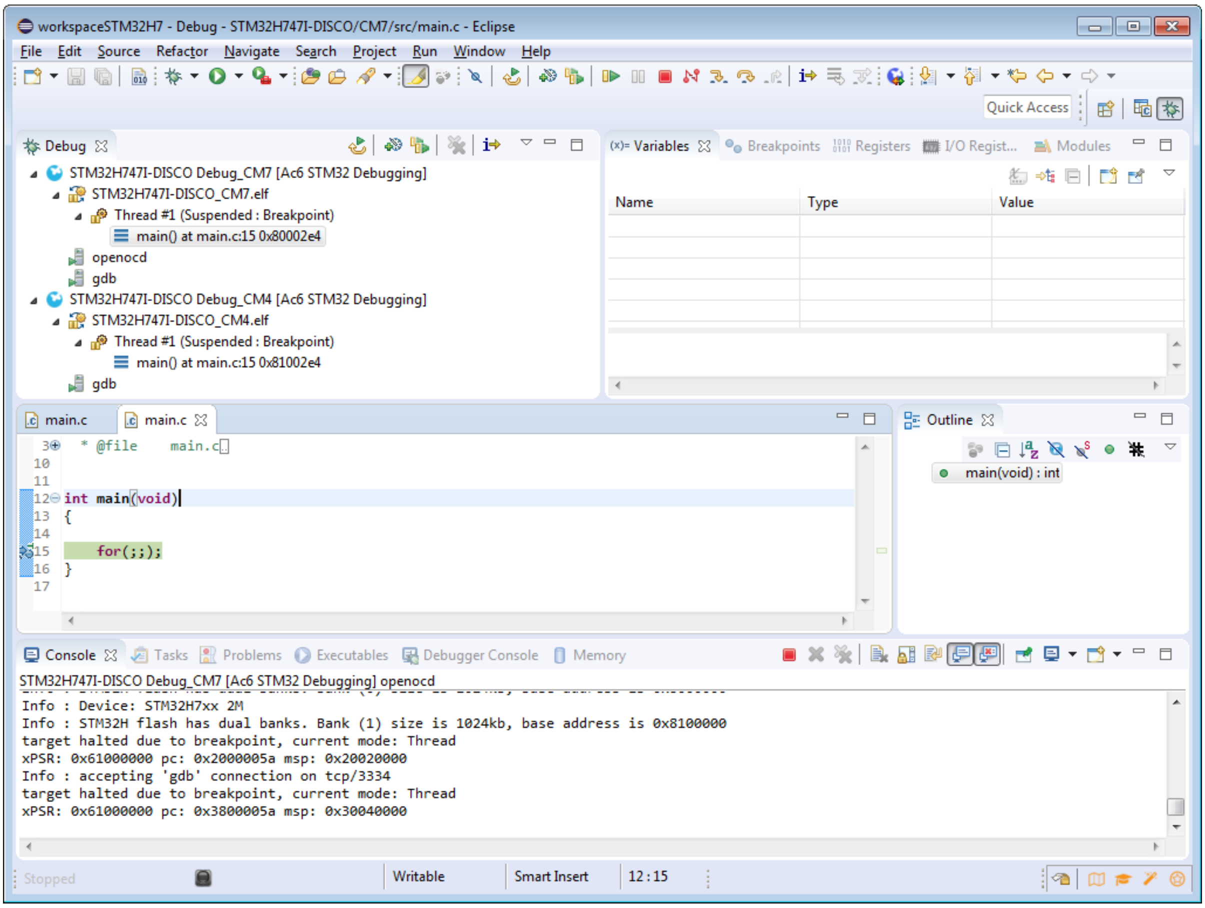


a. *Ensure both C/C++ Application fields are filled in.*

b. *Click the*  *button to debug one or several cores at the same time.*

4. Launch a debug session by clicking the Debug button in the Debug Configuration dialog box.
   In this example, both cores are launched at the same time and stop or break at the “main” instruction (Figure 53).

Figure 53. **Multicore project: debug session**



Break points are applied to both cores so they run either individually or simultaneously using the cross trigger interface (CTI). By default the CTI is not configured, therefore no core is able to break the other.

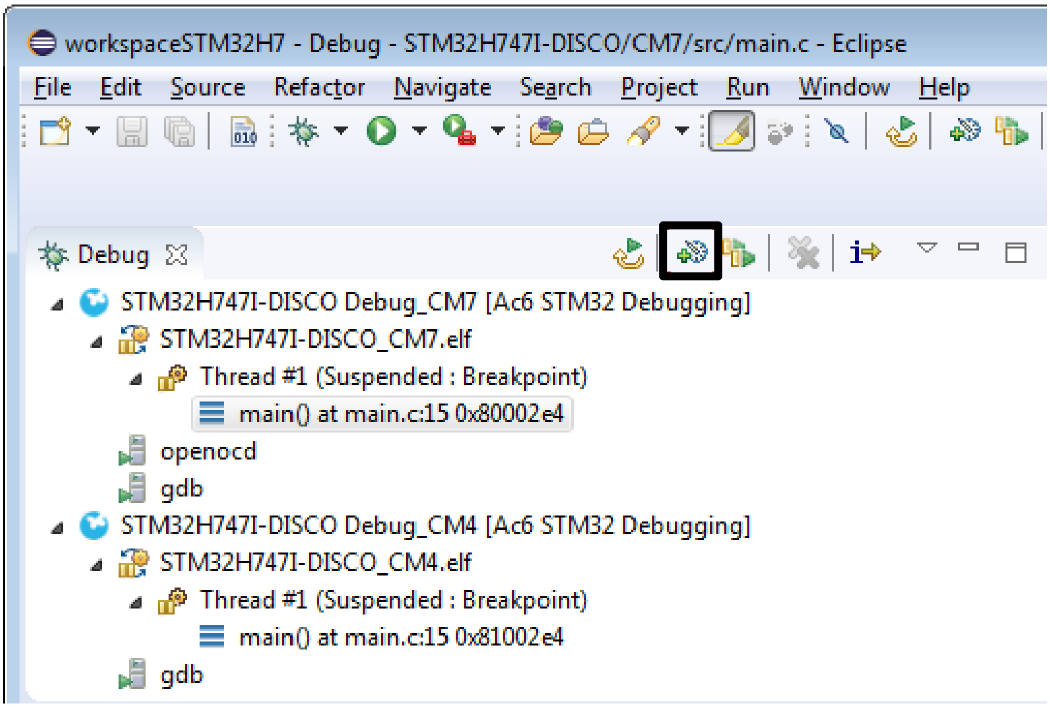
6.3 Embedded cross trigger (ECT) on SW4STM32

By default and as stated in Section 6.2, the CTI is not activated.

To configure the CTI and stop both cores simultaneously, open the debug configuration that has already been created. In the debugger tab, go to the break cross-trigger matrix and select how the core breaks are configured, one breaks the other or both break at the same time. In the current configuration (Figure 54), both cores break at the “main” instruction which means when one core breaks the other breaks simultaneously.

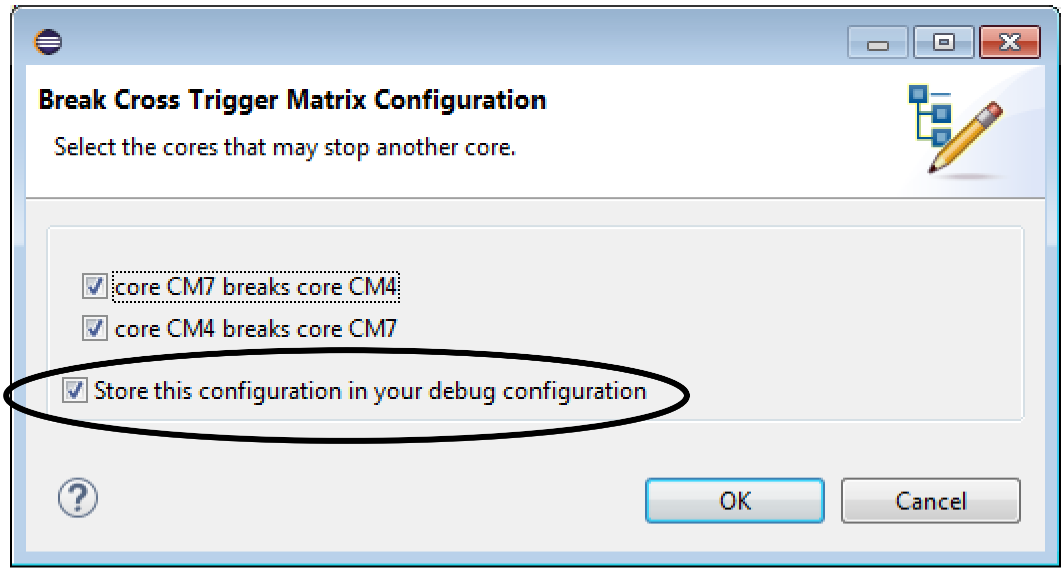
The CTI can be changed on the fly by clicking on Configure the BCTM button (Figure 54).

Figure 54. Multicore project: BCTM button



A new dialog box opens (Figure 55). It is possible to store the new BCTM configuration in the associated debug configuration.

Figure 55. Multicore project: BCTM dialog



Revision history

Table 2. Document revision history

Date	Version	Changes
01-Jul-2019	1	Initial release.
22-Aug-2024	2	Updated: • Section 5.3: Embedded cross trigger (ECT) on MDK-ARM • Section 5.2: Steps for dual debug with ST-LINK V2/V3 • Figure 12. ST-LINK → Multicore • Figure 24. Debugger → Multicore tab • Figure 32. Options → Debugger tab • Figure 37. Flash Download Tab • Figure 46. Debug session • Figure 47. Options for Target → Debug tab Added: • Figure 48. CTI internal connections • Figure 49. Debug session → Toolbox

Contents

List of figures