STM32 dual-core applications with Keil MDK

AN 338, Summer 2021, V 1.0

keil-feedback@arm.com

Abstract

STM32CubeMX and Keil MDK work together seamlessly. This manual explains how to create projects that utilize STM32CubeMX together with Arm Keil MDK, which provides Device Family Packs (DFP) for the STM32 device series. This works out-of-the-box for single-core applications. However, for dual-core applications additional steps are required to create two µVision projects, one for each core.

This application note shows these steps and introduces a Bash script that can be used to create the µVision projects from the output of STM32CubeMX.

Contents

Introduction

This application note explains how to create an STM32 dual-core application using STM32CubeMX and Keil MDK. It is a step-by-step guide that shows an example using the STM32H745I-Discovery board from STMicroelectronics that features the STM32745XIHx with an Arm Cortex-M4 and an Arm Cortex-M7 core.

Prerequisites

To run through the application note, you need to install the following software:

- STM32CubeMX, v6.3.0 or above
- MDK v5.35 or above
- Git for Windows, 2.32.0 or above (to run the Bash script on your Windows machine)

Also, you need these two files (from the **apnt_338.zip** file) to generate the µVision projects:

- Project template file `DualCore_CMx.cprj`
- Project generation script `gen_BasicDualCorePproject.sh`

Initialization scripts help you to load the projects during debugging and to start a debug session:

- `Flash_CM4.ini`
- `Flash_CM7.ini`
- `Debug_CM4.ini`

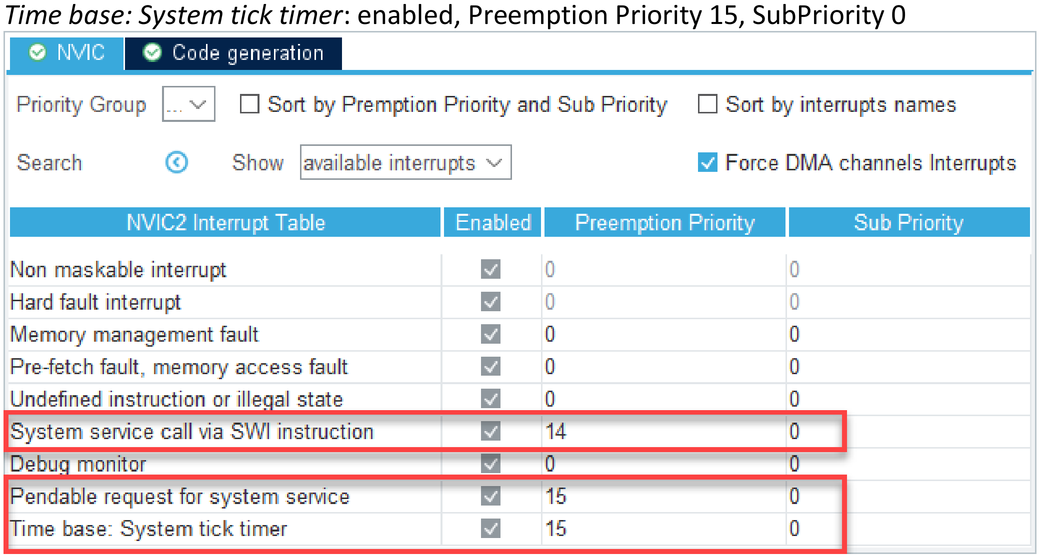
Step 1: Create a project in STM32CubeMX

Note:

The following NVIC settings are required if you want to use Keil RTX5 in your applications. If you do not require an RTOS, they can be omitted.

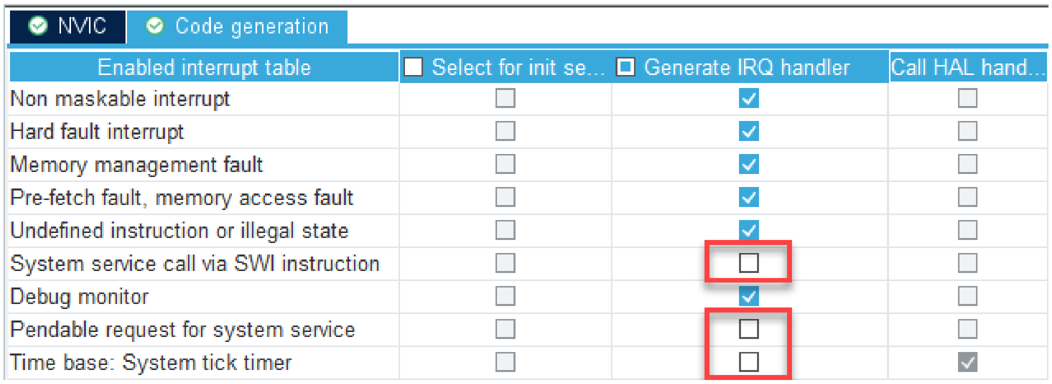
Start in STM32CubeMX

- Use either **Start My project from MCU** or **Start My project from ST Board**. This application note's screenshots are using the STM32H745XIHx device on the STM32H745XI-Disco board.
  - Configure used peripherals on the **Pinout & Configuration** tab:
  - **System Core**:
    Same settings for **NVIC1** and **NVIC2** on the **NVIC** tab:
    - *System service call via SWI instruction*: enabled, Preemption Priority 14, SubPriority 0
    - *Pendable request for system service*: enabled, Preemption Priority 15, SubPriority 0
    - *Time base: System tick timer*: enabled, Preemption Priority 15, SubPriority 0

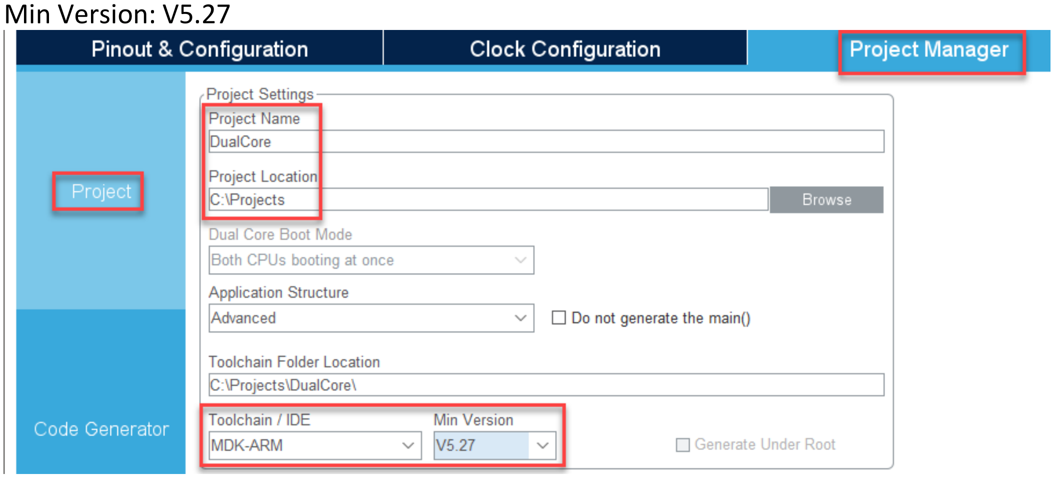


Same settings for **NVIC1** and **NVIC2** on the **Code generation** tab:

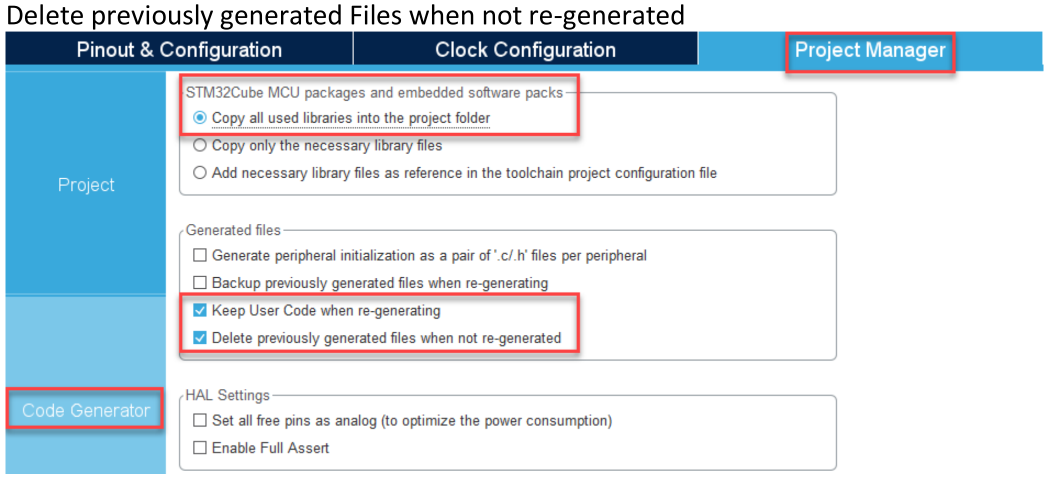
- *System service call via SWI instruction*: uncheck Generate IRQ handler
- *Pendable request for system service*: uncheck Generate IRQ handler
- *Time base:* uncheck Generate IRQ handler



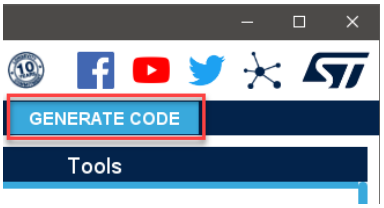
- Configure the clock settings on the **Clock Configuration** tab as required by your project.
- Configure project setting on the **Project Manager** tab:
  - **Project**:
    - Project Name: e.g.: DualCore
    - Toolchain / IDE: MDK-ARM
    - Min Version: V5.27



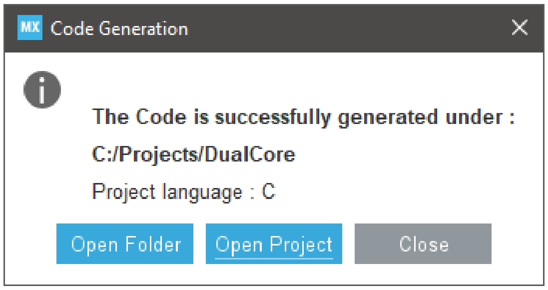
  - **Code Generator** (stay with default values)
    - Copy all used libraries into the project folder
    - Keep User Code when re-generating
    - Delete previously generated Files when not re-generated



- Click on **Generate Code**:



- Close this window:



- Close STM32CubeMX.

Result

When finished, you should have a folder named **DualCore** that contains:

- CM4: Cortex-M4 core specific code
- CM7: Cortex-M7 core specific code
- Common: CM4/CM7 common code
- Drivers: CMSIS (Device header files), HAL drivers
- MDK-ARM: µVision Project with two targets: 'DualCore_CM7' for CM7, 'DualCore_CM4' for CM4
- `.mxproject`: CubeMX project file
- `DualCore.ioc`: CubeMX device configuration file

Step 2: Create basic dual-core projects in µVision

In this step, a Bash script is used to generate µVision projects from the output of STM32CubeMX.

Notes:

- For the use of the shell script a Bash environment is required (refer to Prerequisites).
- The generation script assumes that your Keil MDK installation directory is "C:/Keil_v5". If your path is different, please change line 7 of the script.

Step 2a: Prepare the generation script and project template

- Copy the script `gen_BasicDualCorePproject.sh` and the project template `DualCore_CMx.cprj` to the parent folder of the **DualCore** folder.
- Open `gen_BasicDualCorePproject.sh`, `DualCore_CMx.cprj`, and `DualCore.ioc` (from the **DualCore** folder) in a text editor.
- In `DualCore.ioc`, search for *Mcu.Name=* and copy the name of the MCU (here "**STM32H745XIHx**").
- In `gen_BasicDualCorePproject.sh`, paste the MCU name at line 4 (*devicename=""*).
- In `DualCore_CMx.cprj`, paste the MCU name at line 13 *(Dname="")*.

Step 2b: Run the script

- Open a Bash console in the folder with the `gen_BasicDualCorePproject.sh` script.
- Execute the Bash script: `./gen_BasicDualCorePproject.sh`.
- Close the Bash console.

Note:

Full RTE support/benefit is only possible with single µVision project for a certain core! Run-time environment component configurations are tailored for a certain core.

Result

Two basic Arm Cortex-M4 and Cortex-M7 applications using Keil RTX5 are created (startup only). The folders **CM4** and **CM7** are at the same folder level as **DualCore**.

- The folder **CM7** contains:
  - CM7/Core/Inc: `main.h, stm32h7xx_hal_conf.h, stm32h7xx_it.h`
  - CM7/Core/Src: `main.c, stm32h7xx_hal_msp.c, stm32h7xx_it.c`
  - MDK-ARM/DebugConfig: `*.dbgconf`
  - MDK-ARM/Out: *empty*
  - MDK-ARM/RTE/Device/STM32H745ZITx_CM7: `startup_stm32h745xx.s, system_stm32h7xx.c`
  - MDK-ARM: `DualCore_cm7.uvoptx, DualCore_cm7.uvprojx, stm32h745xx_flash_CM7.sct, stm32h745xx_sram1_CM7.sct`
- The folder **CM4** contains:
  - CM4/Core/Inc: `main.h, stm32h7xx_hal_conf.h, stm32h7xx_it.h`
  - CM4/Core/Src: `main.c, stm32h7xx_hal_msp.c, stm32h7xx_it.c`
  - MDK-ARM/DebugConfig: `*.dbgconf`
  - MDK-ARM/Out: *empty*
  - MDK-ARM/RTE/Device/STM32H745ZITx_CM4: `startup_stm32h745xx.s, system_stm32h7xx.c`
  - MDK-ARM: `DualCore_cm4.uvoptx, DualCore_cm4.uvprojx, stm32h745xx_flash_CM4.sct, stm32h745xx_sram1_CM4.sct`

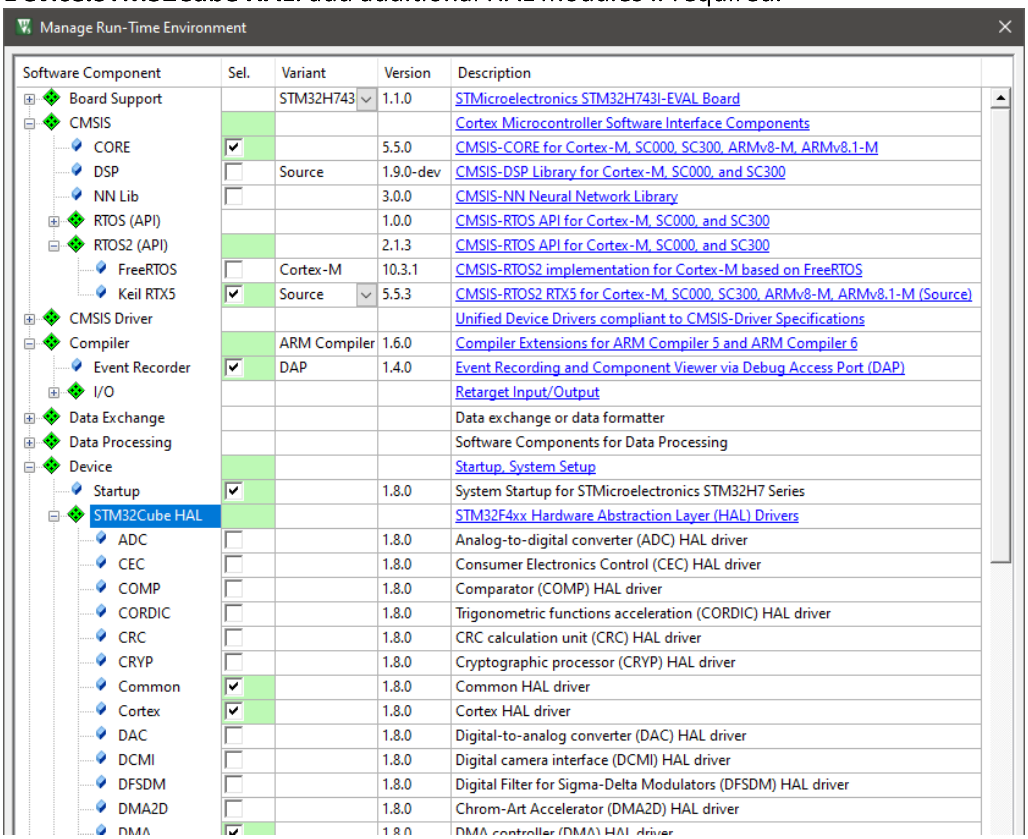
Step 3: Configure the basic µVision projects

Once the basic projects have been generated, it's now time to configure them separately.

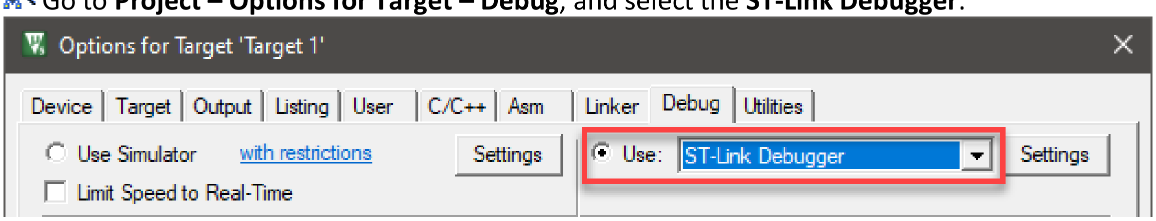
Step 3a: Configure the basic Cortex-M7 µVision project

Go to the **CM7** folder.

- Double-click the DualCore_cm7.uvprojx file in the MDK-ARM folder:
  - Go to **Project – Manage – Run-Time Environment**:
    - Check settings under **CMSIS:CMIS RTOS2 (API)**: used RTOS and used RTOS variant.
    - Check if **Compiler:Event Recorder** is enabled.
    - **Device:STM32Cube HAL**: add additional HAL modules if required.



- Click **OK** to close RTE window.
  - Go to **Project – Options for Target – Debug**, and select the **ST-Link Debugger**:



Save the project and exit µVision.

Step 3b: Configure the basic Cortex-M4 µVision project

Configure the Arm Cortex-M4 project in a similar way as Cortex-M7 in Step 3a: Configure the basic Cortex-M7 µVision project.

Step 4: Develop the applications for each core

Once configuration is done, the actual applications can be developed.

Step 4a: Develop the Cortex-M7 µVision project

- Adapt `main.c` in folder CM7/CM7/Core/Src
- Adapt `main.h` in folder CM7/CM7/Core/Inc
- Check `stm32h7xx_hal_conf.h` in folder CM7/CM7/Core/Inc
- Add Arm Cortex-M7 application code
- Adapt the µVision project in CM7/MDK-ARM/DualCore_cm7.uvprojx. For example, enable Event Recorder Global Initialization in the `RTX_Config.h` file and configure Event Recorder settings in `EventRecorderConf.h`.

Step 4b: Develop the Cortex-M4 µVision project

Develop the Arm Cortex-M4 application in a similar way as Cortex-M7 in Step 4a: Develop the Cortex-M7 µVision project.

Step 5: Flash the applications

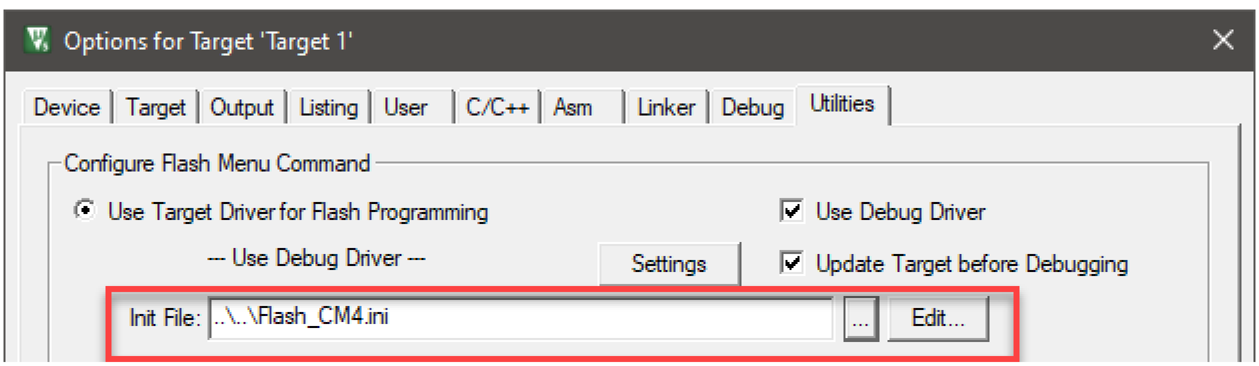
Each core can be flashed with the corresponding µVision project:

- Flash the Arm Cortex-M4 image with the CM4/MDK-ARM/DualCore_cm4.uvprojx µVision project.
- Flash the Arm Cortex-M7 image with the CM7/MDK-ARM/DualCore_cm7.uvprojx µVision project.

Both cores can also be flashed at once in a single µVision project. In this case, an initialization file is required:

- `Flash_CM7.ini` for Arm Cortex-M7 µVision Project. Contains command to load Arm Cortex-M4 code.
- `Flash_CM4.ini` for Arm Cortex-M4 µVision Project. Contains command to load Arm Cortex-M7 code.

Go to **Project – Options for Target – Utilities** to add the **Init File**:



Notes:

- After flash programming, a reset is required.
- Both INI scripts are delivered as part of the application note's ZIP file.

Step 6: Debug the applications

You can either debug each application stand-alone or both applications at the same time using two µVision windows.

Notes:

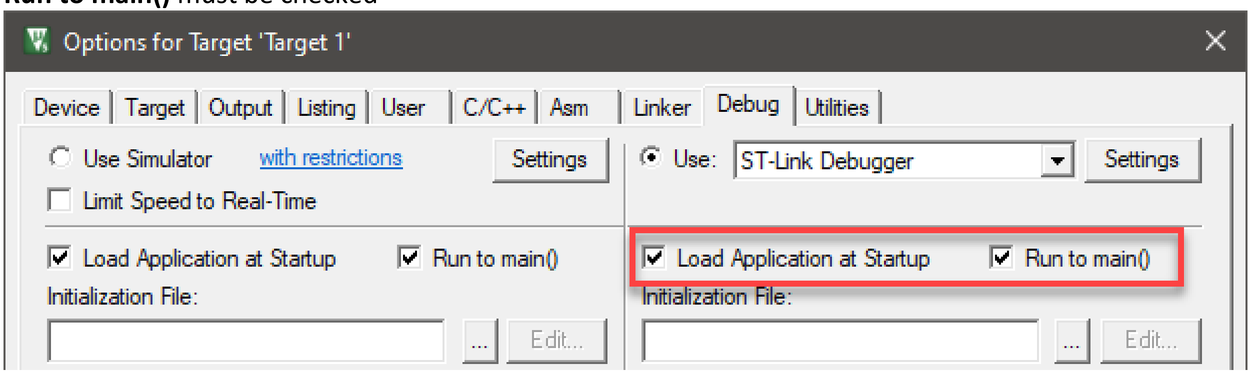
- You may disable 'Update Target before Debugging' if you like to differ between 'Flash' and 'Debug' steps.
- Copy the `Debug_CM4.ini` script from the application note's ZIP file to the folder containing the CM4 and CM7 folders.

Step 6a: Debug the Cortex-M7 application

In this scenario, the Arm Cortex-M4 is running freely, and an ST-Link is used for debugging. The prerequisite is that both applications are flashed (refer to Step 5: Flash the applications) and the project CM7/MDK-ARM/DualCore_cm7.uvprojx is opened in µVision.

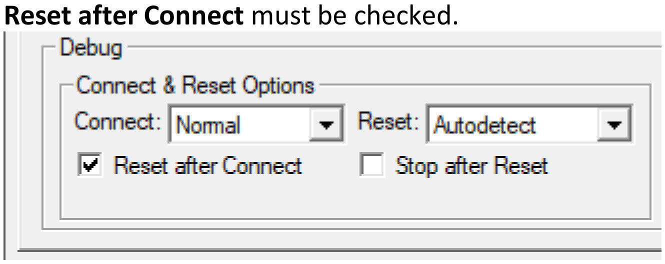
Go to **Project – Options for Target – Debug** to check the debug settings:

- **Load Application at Startup** must be checked
- **Run to main()** must be checked



Go to **Project – Options for Target – Debug – ST-Link Debugger Settings**:

- **Connect**: Normal
- **Reset**: Autodetect
- **Reset after Connect** must be checked.



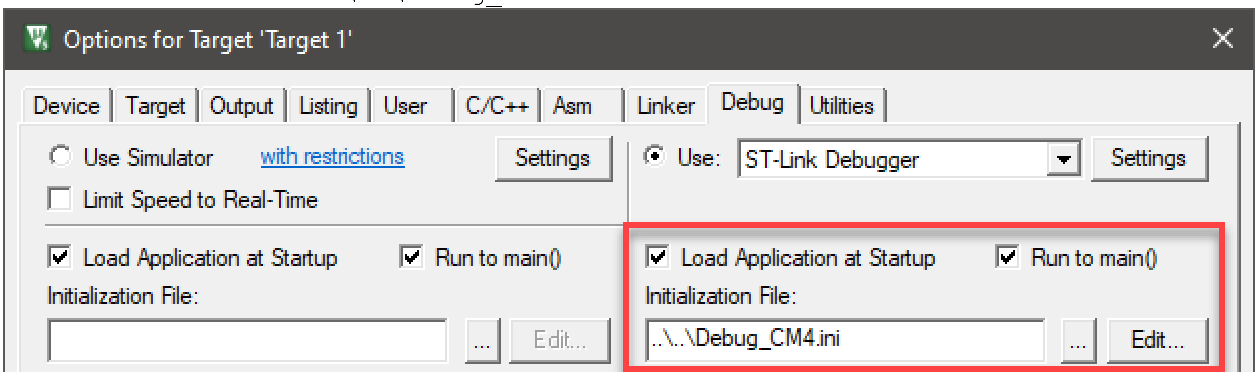
Go to **Debug – Start/Stop Debug Session** (Ctrl + F5). The Arm Cortex-M7 application is loaded and stops at `main()`. Continue debugging as normal.

Step 6b: Debug the Cortex-M4 application

In this scenario, the Arm Cortex-M7 is running freely, and an ST-Link is used for debugging. The prerequisite is that both applications are flashed (refer to Step 5: Flash the applications) and the project CM4/MDK-ARM/DualCore_cm4.uvprojx is opened in µVision.

Go to **Project – Options for Target – Debug** to check the debug settings:

- **Load Application at Startup** must be checked
- **Run to main()** must be checked
- **Initialisation File** is set to `..\..\Debug_CM4.ini`



Go to **Project – Options for Target – Debug – ST-Link Debugger Settings**:

- **Connect**: Normal
- **Reset**: Autodetect
- **Reset after Connect** must be checked.

Go to **Debug – Start/Stop Debug Session** (Ctrl + F5). The Arm Cortex-M4 application is loaded and stops at `main()`. Continue debugging as normal.

Note: If you want to debug the Arm Cortex-M4 application from start then you must add a synchronization point to the Arm Cortex-M4 startup, for example a busy loop, where the application loops until a debugger is connected. With the debugger you can manipulate the PC to continue running after the synchronization point.

Step 6c: Debug both applications

Start two µVision instances, one for each core (double-click CM7/MDK-ARM/DualCore_cm7.uvprojx and CM4/MDK-ARM/DualCore_cm4.uvprojx).

- Check in both instances that the ST-Link setting 'Shareable ST-Link' is checked.
- First, start the Arm Cortex-M7 Debug session (refer to Step 6a: Debug the Cortex-M7 application)
- Run until the Arm Cortex-M4 is released and runs
- Then, start the Arm Cortex-M4 Debug session (refer to Step 6b: Debug the Cortex-M4 application)

Step 7: Modify the STM32CubeMX project

If required, open the STM32CubeMX project (`DualCore.ioc` in the **DualCore** folder) to make changes to the STM32CubeMX code or configuration. Afterwards, merge the changes from the newly generated code to the two derived MDK projects.

Conclusion

This application note showed how you can create dual-core projects for STM32 targets using STMicroelectronics' STM32CubeMX and Keil MDK. A Bash script was introduced that creates the µVision projects from the generated output of STM32CubeMX.