# STM32 dual-core applications with Keil MDK

AN 338, Summer 2021, V 1.0



#### Abstract

STM32CubeMX and Keil MDK work together seamlessly. This <u>manual</u> 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  $\mu$ Vision projects from the output of STM32CubeMX.

#### Contents

Abstract1	
Introduction2	
Prerequisites	•
Step 1: Create a project in STM32CubeMX2	•
Start in STM32CubeMX	•
Result4	
Step 2: Create basic dual-core projects in μVision5	,
Step 2a: Prepare the generation script and project template5	,
Step 2b: Run the script5	)
Result5	,
Step 3: Configure the basic μVision projects6	j
Step 3a: Configure the basic Cortex-M7 $\mu$ Vision project6	)
Step 3b: Configure the basic Cortex-M4 μVision project6	)
Step 4: Develop the applications for each core7	,
Step 4a: Develop the Cortex-M7 μVision project7	,
Step 4b: Develop the Cortex-M4 μVision project7	,
Step 5: Flash the applications7	,
Step 6: Debug the applications	j
Step 6a: Debug the Cortex-M7 application8	,
Step 6b: Debug the Cortex-M4 application9	I
Step 6c: Debug both applications9	ł
Step 7: Modify the STM32CubeMX project9	ł
Conclusion9	I

### 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:

- <u>STM32CubeMX</u>, 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  $\mu$ Vision 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

Solution Solution Solution					
Priority Group Sort by Premption Priority and Sub Priority Sort by interrupts names					
Search C Show available interrupts V Force DMA channels Interru					
NVIC2 Interrupt Table	Enabled	Preemption Priority	Sub Priority		
Non maskable interrupt	$\checkmark$	0	0		
Hard fault interrupt	$\checkmark$	0	0		
Memory management fault	$\checkmark$	0	0		
Pre-fetch fault, memory access fault	$\checkmark$	0	0		
Undefined instruction or illegal state	$\checkmark$	0	0		
System service call via SWI instruction	$\checkmark$	14	0		
Debug monitor	$\checkmark$	0	0		
Pendable request for system service	$\checkmark$	15	0		
Time base: System tick timer	$\checkmark$	15	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

SNVIC Sold generation			
Enabled interrupt table	Select for init se	Generate IRQ handler	Call HAL hand
Non maskable interrupt		✓	
Hard fault interrupt		✓	
Memory management fault		✓	
Pre-fetch fault, memory access fault		<ul> <li>✓</li> </ul>	
Undefined instruction or illegal state		<u></u>	
System service call via SWI instruction			
Debug monitor			
Pendable request for system service			
Time base: System tick timer			$\checkmark$

- 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

1 111041 & C	onliguration	Сюск	Configuration	Project Manager
Project	Project Settings Project Name DualCore Project Location C:\Projects Dual Core Boot Mode Both CPUs booting at once Application Structure		]	Browse
Code Generator	Advanced Toolchain Folder Location C:\Projects\DualCore\ Toolchain / IDE	~ Min Version	Do not generate the main()	

- 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

Pinout & C	Configuration	Clock Configura	ation	Project M	anager
Project	STM32Cube MCU packag Copy all used libraries Copy only the necess Add necessary library Generated files	file			
	Generate peripheral initialization as a pair of '.c/.h' files per peripheral Backup previously generated files when re-generating				
	<ul> <li>Keep User Code when</li> <li>Delete previously gene</li> </ul>				
Code Generator	HAL Settings Set all free pins as an Enable Full Assert	alog (to optimize the power consum	ption)		

• Click on Generate Code:



• 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  $\mu$ Vision 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:
  - o CM7/Core/Inc:main.h, stm32h7xx\_hal\_conf.h, stm32h7xx\_it.h
  - o CM7/Core/Src:main.c, stm32h7xx\_hal\_msp.c, stm32h7xx\_it.c
  - MDK-ARM/DebugConfig: \*.dbgconf
  - MDK-ARM/Out: *empty*
  - o MDK-ARM/RTE/Device/STM32H745ZITx\_CM7:startup\_stm32h745xx.s, system\_stm32h7xx.c
  - o MDK-ARM:DualCore\_cm7.uvoptx, DualCore\_cm7.uvprojx, stm32h745xx\_flash\_CM7.sct, stm32h745xx\_sram1\_CM7.sct
- The folder CM4 contains:
  - o CM4/Core/Inc:main.h, stm32h7xx\_hal\_conf.h, stm32h7xx\_it.h
  - o CM4/Core/Src:main.c, stm32h7xx\_hal\_msp.c, stm32h7xx\_it.c
  - o MDK-ARM/DebugConfig: \*.dbgconf
  - MDK-ARM/Out: *empty*
  - o 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:
  - So 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.

Software Component	Sel.	Variant	Version	Description
🐵 🚸 Board Support		STM32H743 ~	1.1.0	STMicroelectronics STM32H743I-EVAL Board
CMSIS				Cortex Microcontroller Software Interface Components
CORE	<ul><li>✓</li></ul>		5.5.0	CMSIS-CORE for Cortex-M, SC000, SC300, ARMv8-M, ARMv8.1-M
DSP		Source	1.9.0-dev	CMSIS-DSP Library for Cortex-M, SC000, and SC300
NN Lib			3.0.0	CMSIS-NN Neural Network Library
🗉 🚸 RTOS (API)			1.0.0	CMSIS-RTOS API for Cortex-M, SC000, and SC300
🖃 🚸 RTOS2 (API)			2.1.3	CMSIS-RTOS API for Cortex-M, SC000, and SC300
FreeRTOS		Cortex-M	10.3.1	CMSIS-RTOS2 implementation for Cortex-M based on FreeRTOS
Keil RTX5	<b>v</b>	Source 🗸	5.5.3	CMSIS-RTOS2 RTX5 for Cortex-M, SC000, SC300, ARMv8-M, ARMv8.1-M (Source)
🗄 🚸 CMSIS Driver				Unified Device Drivers compliant to CMSIS-Driver Specifications
🖨 🚸 Compiler		ARM Compiler	1.6.0	Compiler Extensions for ARM Compiler 5 and ARM Compiler 6
Event Recorder	<b>v</b>	DAP	1.4.0	Event Recording and Component Viewer via Debug Access Port (DAP)
				Retarget Input/Output
🗄 🚸 Data Exchange				Data exchange or data formatter
🗉 🚸 Data Processing				Software Components for Data Processing
😑 🚸 Device				Startup, System Setup
Startup	<b>v</b>		1.8.0	System Startup for STMicroelectronics STM32H7 Series
🗉 🚸 STM32Cube HAL				STM32F4xx Hardware Abstraction Layer (HAL) Drivers
ADC			1.8.0	Analog-to-digital converter (ADC) HAL driver
CEC			1.8.0	Consumer Electronics Control (CEC) HAL driver
COMP			1.8.0	Comparator (COMP) HAL driver
CORDIC			1.8.0	Trigonometric functions acceleration (CORDIC) HAL driver
CRC			1.8.0	CRC calculation unit (CRC) HAL driver
CRYP			1.8.0	Cryptographic processor (CRYP) HAL driver
Common	<b>v</b>		1.8.0	Common HAL driver
Cortex	<b>v</b>		1.8.0	Cortex HAL driver
DAC			1.8.0	Digital-to-analog converter (DAC) HAL driver
DCMI			1.8.0	Digital camera interface (DCMI) HAL driver
DFSDM			1.8.0	Digital Filter for Sigma-Delta Modulators (DFSDM) HAL driver
DMA2D			1.8.0	Chrom-Art Accelerator (DMA2D) HAL driver
			1.8.0	DMA controller (DMA) HAL driver

Save the project and exit  $\mu$ Vision.

0

## 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  $\mu$ Vision 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  $\mu$ Vision project.

#### Step 5: Flash the applications

Each core can be flashed with the corresponding  $\mu\text{V}\textsc{ision}$  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.

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

🐺 Options for Target 'Target 1'		×
Device Target Output Listing User C/C++ Asm	Linker Debug Utilities	
Configure Flash Menu Command		٦
• Use Target Driver for Flash Programming	Vise Debug Driver	
Use Debug Driver	Settings   Update Target before Debugging	
Init File:\\Flash_CM4.ini	Edit	

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  $\mu$ Vision 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  $\mu$ Vision.

K 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

8 Options for Target 'Target 1'			
Device   Target   Output   Listing	User C/C++ Asm	Linker Debug Utilities	
C Use Simulator with restrict	ions Settings	• Use: ST-Link Debugger	✓ Settings
Load Application at Startup	Run to main()	Load Application at Startup	Run to main()
Initialization File:		Initialization File:	
	Edit		Edit

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

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

Debug	et Ontions —			
Connect: Nom	al 🔻	Reset:	Autodetect	-
Reset after	Connect	🗆 Sto	p after Reset	_

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  $\mu$ Vision.

K 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

😗 Options for Target 'Target 1'	×
Device Target Output Listing User C/C++ Asm	Linker Debug Utilities
© Use Simulator with restrictions Settings	• Use: ST-Link Debugger
✓ Load Application at Startup	I Load Application at Startup I Run to main()
Initialization File:	Initialization File:
Edit	\\Debug_CM4.ini Edit

K 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  $\mu$ Vision projects from the generated output of STM32CubeMX.