This tutorial is an extract of my thesis report. The user' source and header files cited in this tutorial will be available in the Synology Drive folder, together with the µVision and Pycharm (for the python files) projects. This tutorial refers to the files used during the thesis, so that the results obtained can be replicated, but any other user file can be added to the project instead of my files.

9. Appendix

Tutorial

To run the NN inference, the Keil MDK (version 5.37) was used. A tutorial on the Keil tool and setup is presented. This thesis considered three different MCUs:

1. STM32H7A3ZI, from STMicroelectronics, based on the high-performance Arm Cortex-M7 32-bit RISC core operating at up to 280 MHz [62]. The board used is the NUCLEO-STM32H7A3ZI.
2. TM4C123GHPM, from Texas Instrument, based on the Arm Cortex-M4 32-bit RISC core operating at up to 80 MHz [63].
3. Cortex-M55 virtual-hardware target (VHT), Arm's most AI-capable Cortex-M processor and the first to feature Arm Helium vector processing technology [64].

The purpose of this tutorial is to provide a detailed explanation of the µVision setup to execute latency and energy measurements. Each chapter will focus on a different MCU, this division is necessary because the setup depends on the targeted hardware. The energy measurements won't be included for the Cortex-M55 since they cannot be performed on a VHT.

More information on the µVision's functionalities are available in the links at the end of the tutorial. The last chapter of this tutorial is reserved for a description of the Flash and RAM memory organization in embedded C.

1. STM32H7A3ZI latency and energy measures

In this chapter the setup for the NUCLEO-H7A3ZI board will be documented. µVision' functionalities that allow for latency and the energy consumption measurements will be discussed.

Hardware and Software requirements:

1. Any ST evaluation board with an on-board MIPI10 or ST-LINK/V2, it includes Discovery, NUCLEO and EVAL boards. This tutorial uses a NUCLEO-H7A3ZI-Q (Fig.17, left).
2. Keil MDK V 5.28 or later, and STM32CubeMX installed.
3. The Keil ULINK*plus*, a debug adapter that connects the PC's USB port to the target system (via JTAG or a similar debug interface) and allows to debug, trace, analyze and measure the consumption of the program running on the target hardware. (Fig.17, right).

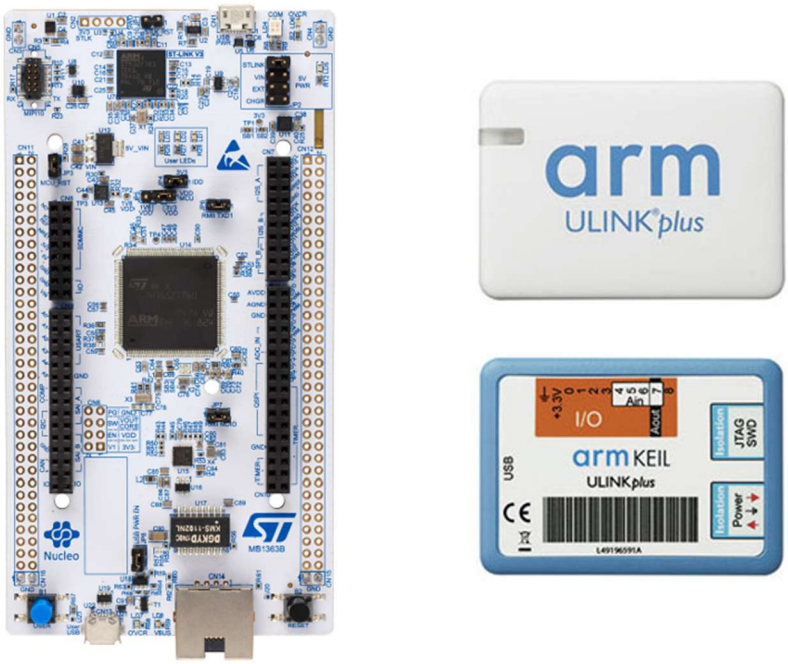
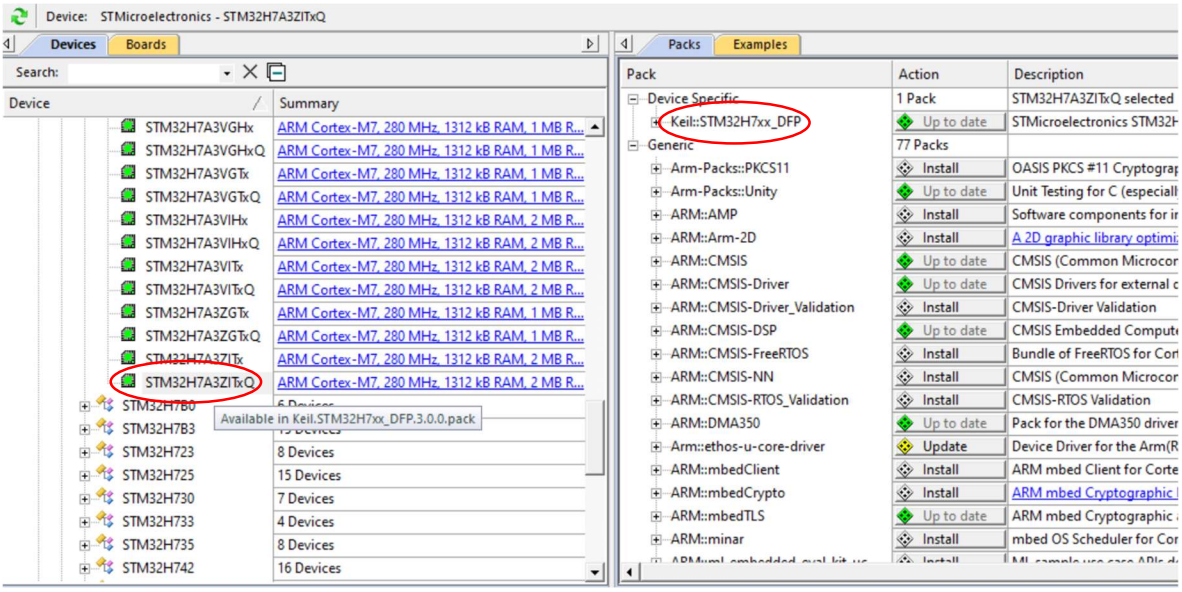


Figure 1 - NUCLEO-STM32H7A3ZI on the left, the debug adapter ULINKplus on the right.

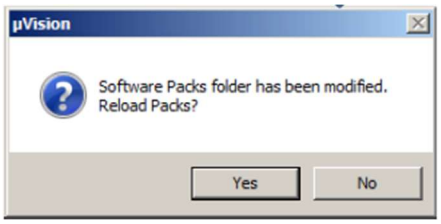
A - Start µVision and the pack installer

The Pack installer allows for the download of some components needed for this project. These instructions reference a NUCLEO-H7A3ZI-Q board. For a different board, install the appropriate Pack and example. For the download of the packages required, the computer must be connected to Internet. The Pack Installer is usually in the folder [MDK installation_path]\UV4\, the default installation path is C:\Keil_v5.

Open the Pack Installer by clicking on its icon: and select from the list of devices the correct MCU.



From the Packs tab, install the device specific pack that corresponds to the MCU selected, in this case Keil::STM32H7xx_DFP. Then install ARM::CMSIS (this project version is the 5.9.0), ARM::CMSIS-Driver, ARM::CMSIS-DSP (I haven't installed ARM::CMSIS-NN and I suggest not to install it), Keil::ARM_Compiler.
Close the Pack Installer. If a dialog box opens stating a Pack folder has been updated, click "Yes" to reload.

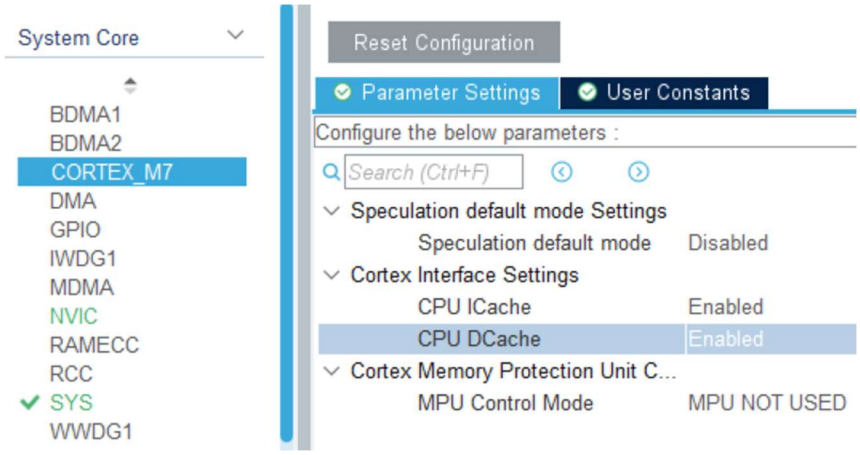


B - MCU setup with STM32CubeMX

Start STM32CubeMX, click on File → New Project, then on the "MCU/MPU Filters" tab, look for the correct MCU, as shown in the following picture.

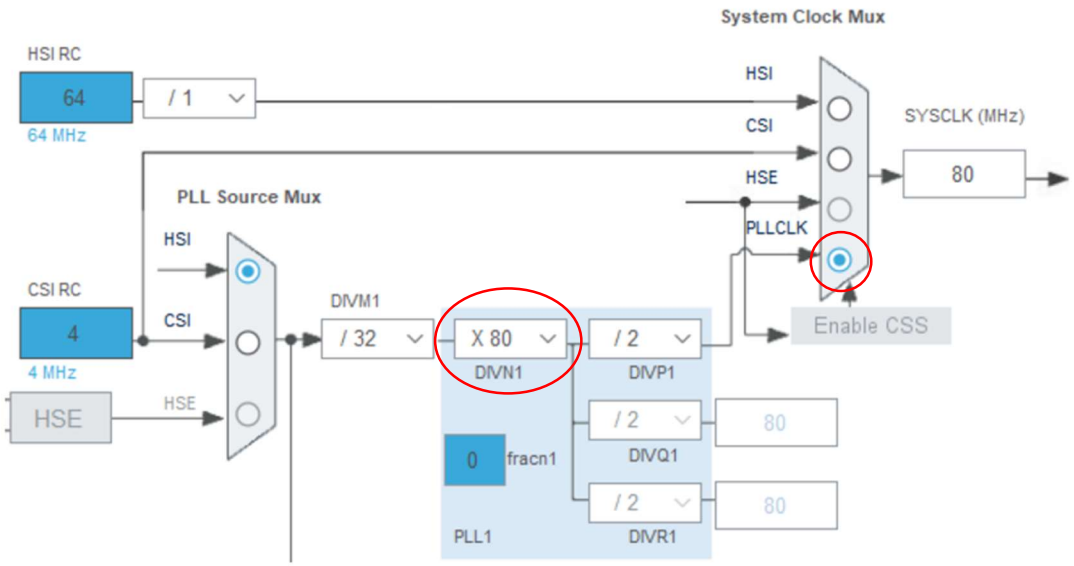
Once the correct MCU has been selected, a window will open asking if you want to apply the default configuration, which includes the Memory Protection Unit (MPU). The main purpose of using the MPU is to protect memory regions by defining different access permissions in privileged and unprivileged access levels for some embedded operating systems [66]. Since this project doesn't include an embedded OS, click on "No".

In Pinout & Configuration → System Core → CORTEX_M7 the two caches, D and I, must be enabled. The rest of Pinout & Configuration can be left as default.

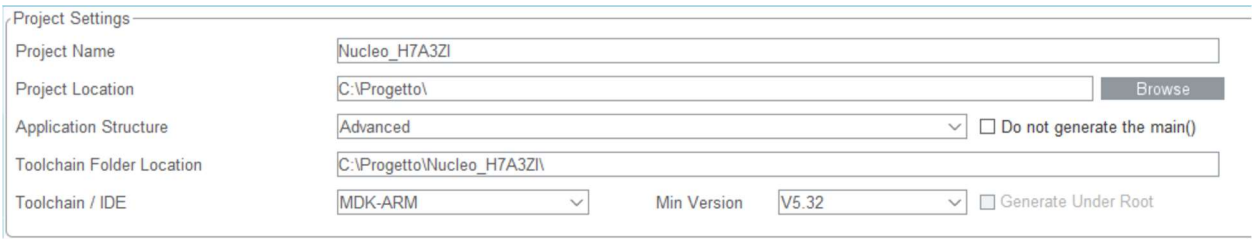


In clock configuration it's possible to change the clock frequency, we will select 80 MHz for this tutorial. STM32 microcontrollers have two internal RC oscillators, known as the HSI (high-speed internal) and LSI (low-speed internal) oscillators. In this tutorial, the clock source selected is the HSI. The LSI oscillator is a lower speed, low power clock source [67]. In this project, we selected a clock frequency (80 MHz) that is higher than the maximum clock frequency of the HSI (64 MHz), for this reason we use the Phase-locked loop (PLL). PLL is a clock generation engine in the MCU which is used to generate the clock speed higher than the internal HSI.

In order to have a system clock (SYSCLK) of 80 MHz the DIVN1 value must be changed to 80 and PLLCLK should be selected as system clock source in the multiplexer “System Clock Mux”.



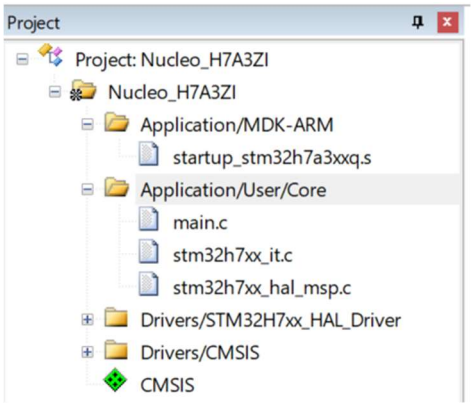
The last step before generating the code is to give a name to the project in the “Project Manager” tab, and select the correct IDE and version, close to Toolchain/IDE.



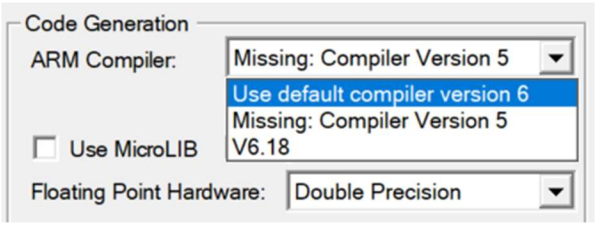
To generate the .c files, press the **GENERATE CODE** button. Then open the project and continue on the Keil MDK tool.

C - Keil MDK setup

In the Keil project created, there should be the following folders and files:



Click on Options for Target icon → Target, in the Code Generation tab, select the correct compiler (Use default compiler 6).

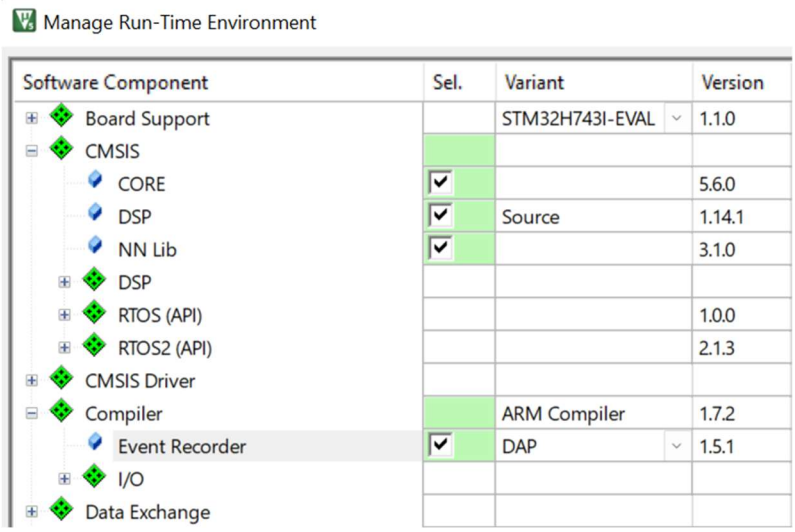


In the C/C++ tab, it's possible to choose the compiler optimization, for this project the O2 optimization was selected.

TIPS: Always click on "Ok" before closing a window, otherwise your changes won't be saved.

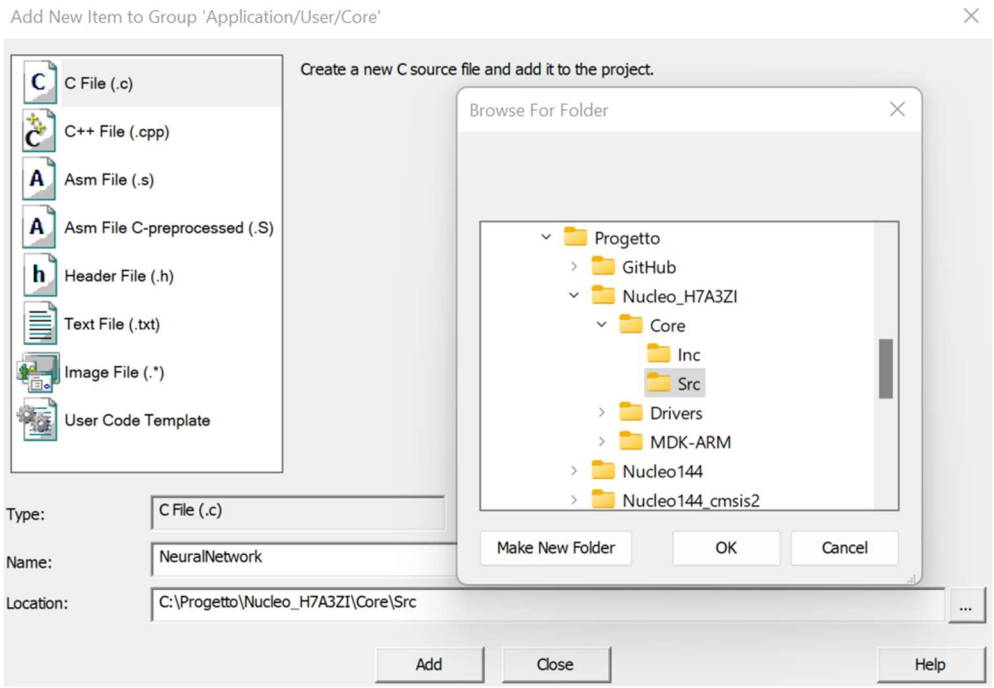
Include the CMSIS-NN code

In your project you need the CMSIS-NN library. Click the button Manage Run-Time Environment , expand CMSIS and select DSP and NN Lib, as in the following figure. Then expand "Compiler" and include Event Recorder, the way to use the Event Recorder will be explained later in this chapter (section D).



Click on "Ok" and close the window.

The .c file with the network functions (NeuralNetwork.c), its header file (NeuralNetwork.h) and the file with the parameters (parameters.h) should be added to the project. To add a source file, right click on the project folder Application/User/Core and then in "Add New Item". Then select a .c or .h type of file and name it. Select the right location for the file: if it's a source file, then it should be in C:/…/Project_name/Core/Src, if it's a header file, the correct location is C:/…/Project_name/Core/Inc.

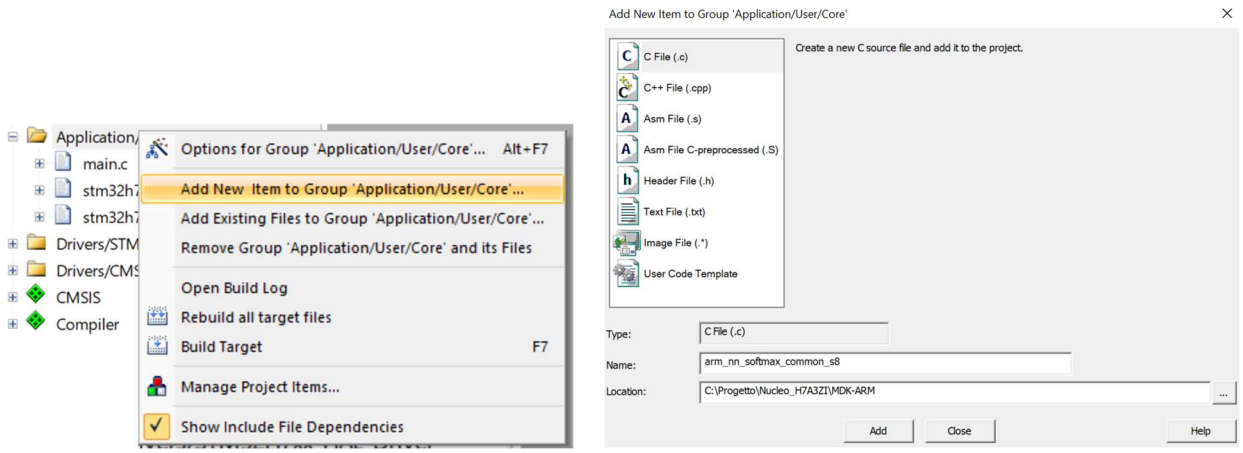


In the main.c file, if you want your code unchanged when you're regenerating the files with the STM32CubeMX tool, you should only write between the commented lines. For example, the private variables should be defined between the begin and end. If you're not interested in regenerating the files, ignore or delete those comments, just add your code.

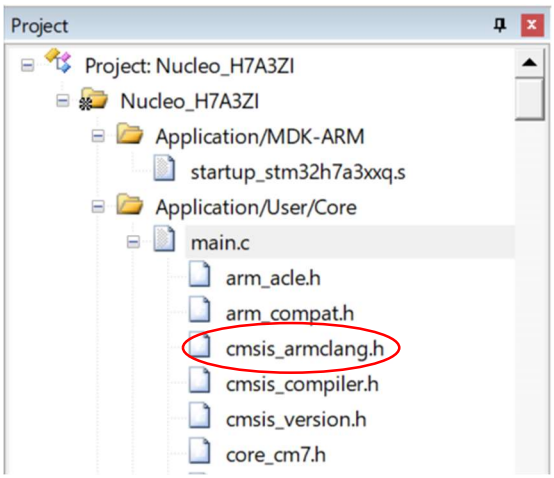
```
/* Private variables -*/
/* USER CODE BEGIN PV */

/* USER CODE END PV   */
```

Before compiling, you should pay attention to two issues:

1) Depending on your CMSIS Core version, it's possible that a file it's missing (arm_nn_softmax_common_s8.c). Check if this file is included in you project (open CMSIS in the project tab), if it's not, right click on the project folder Application/User/Core and then in "Add New Item". Then select a C type of file, name it "arm_nn_softmax_common_s8" and press "Add". Then click on this link [68], copy the code and paste it in the newly created file .c, in the µVision project. The version 5.9.0 of the CMSIS Core should not have this problem.



2) Open the header file cmsis_armclang.h, you can find this header file among the files included by the main.c file, in the Project tab, see the following figure.



In the header file, look for "`__SXTB16_RORn`" and "`__SXTAB16_RORn`", if these are not defined, then add the following lines to the file, after the list of "compiler specific intrinsics" (line 1358 circa):

```
#define __SXTB16_RORn(ARG1, ARG2)          __SXTB16(__ROR(ARG1, ARG2))

#define __SXTAB16_RORn(ARG1, ARG2, ARG3)  __SXTAB16(ARG1, __ROR(ARG2, ARG3))
```

```
1421  #define      __SMLSLDX                  __builtin_arm_smlsldx
1422  #define      __SEL                      __builtin_arm_sel
1423  #define      __QADD                     __builtin_arm_qadd
1424  #define      __QSUB                     __builtin_arm_qsub
1425
1426  #define __PKHBT(ARG1,ARG2,ARG3)          ( ((((uint32_t)(ARG1))          ) & 0x0000FFFFUL) |
1427                                             ((((uint32_t)(ARG2)) << (ARG3)) & 0xFFFF0000UL)  )
1428
1429  #define __PKHTB(ARG1,ARG2,ARG3)          ( ((((uint32_t)(ARG1))          ) & 0xFFFF0000UL) |
1430                                             ((((uint32_t)(ARG2)) >> (ARG3)) & 0x0000FFFFUL)  )
1431
1432  #define __SXTB16_RORn(ARG1, ARG2)        __SXTB16(__ROR(ARG1, ARG2))
1433
1434  #define __SXTAB16_RORn(ARG1, ARG2, ARG3) __SXTAB16(ARG1, __ROR(ARG2, ARG3))
1435
```

After these two steps, build the program clicking on Build  . If you want to flash the code and measure latency and energy consumption, skip the next section, and go to section D of this chapter.

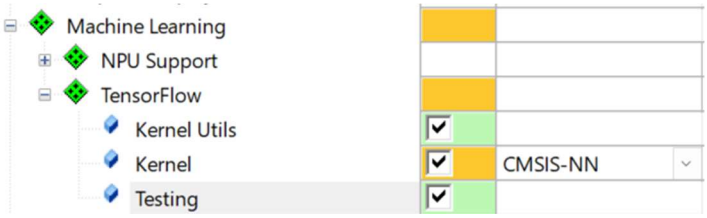
Include the TFLite code

To use the model generated by TensorFlow Lite, you must follow all the instructions shown in "Include the CMSIS-NN code" section. Instead of the files NeuralNetwork.c, NeuralNetwork.h, the files automatically generated by TensorFlow Lite: FFNN_model.cpp, FFNN_model.h.

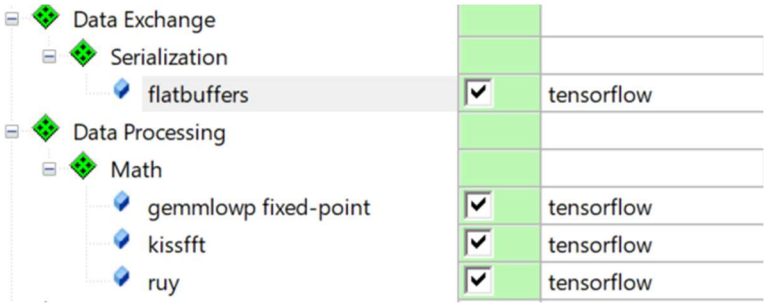
Compared to when we're only using CMSIS-NN, there are some more steps to include the TFLite library in the project.

Open the Pack Installer and install the five packages shown below: tensorflow::flatbuffers, tensorflow::gemmlowp, tensorflow::kissfft, tensorflow::ruy, tensorflow::tensorflow-lite-micro.

Click on Manage Run-Time Environment and expand Machine Learning → TensorFlow. Then include Kernel Utils, Kernel, and Testing, as shown below. In the drop-down menu, "CMSIS-NN" should be selected, in this way the TFLite implementation will use the CMSIS-NN optimized functions (it takes approximately 30 KB of Flash memory). If you select "Ethos", the implementation will be optimized for the Ethos processor, the "Reference" option, instead, makes the implementation target-independent [69].



The orange rectangle means that some other components are needed to use the ones already selected. In the image below, the requested components are being selected, following the instructions under "Validation output".



In Option for Target → C/C++ make sure that the language C++ selected is C++11.
In the file micro_time.cpp make sure that stm32h7xx_hal.h header file is included, if it's not, the following errors might appear. If not included, include the stm32h7xx_hal.h header file.

```
RTE/Machine_Learning/micro_time.cpp(45): error: use of undeclared identifier
'CoreDebug_DEMCR_TRCENA_Msk'
    CoreDebug->DEMCR |= CoreDebug_DEMCR_TRCENA_Msk;
                        ^
```

```
RTE/Machine_Learning/micro_time.cpp(48): error: use of undeclared identifier
'DWT'
    DWT->CYCCNT = 0;
    ^
RTE/Machine_Learning/micro_time.cpp(49): error: use of undeclared identifier
'DWT'
    DWT->CTRL |= 1UL;
    ^
RTE/Machine_Learning/micro_time.cpp(62): error: use of undeclared identifier
'DWT'
  return DWT->CYCCNT;
```

Add FFNN_model.cpp and FFNN_model.h files in the project as shown in "Include CMSIS-NN code".

Before compiling, you must change three .c source files in .cpp files, main.c, stm32h7xx_hal_msp.c and stm32h7xx_it.c, otherwise this error might pop up: `error "cstdio" file not found.`

TIPS: When you copy and paste the main.h code, or when you edit it, make sure that the C++ header files are included before these lines:

```
#ifdef __cplusplus
extern "C" {
#endif

# C header files/functions declaration

#ifdef __cplusplus
}
#endif
```

For example, FFNN_model.h is the header file of a .cpp file (FFNN_model.cpp), as you can see below, it's included before the `#ifdef`.

```
21  /* Define to prevent recursive inclusion -------------------------------------*/
22  #ifndef __MAIN_H
23  #define __MAIN_H
24
25  #include "FFNN_model.h"
26  #include "tensorflow/lite/micro/all_ops_resolver.h"
27  #include "tensorflow/lite/micro/micro_error_reporter.h"
28  #include "tensorflow/lite/micro/micro_interpreter.h"
29
30  #ifdef __cplusplus
31  extern "C" {
32  #endif
```

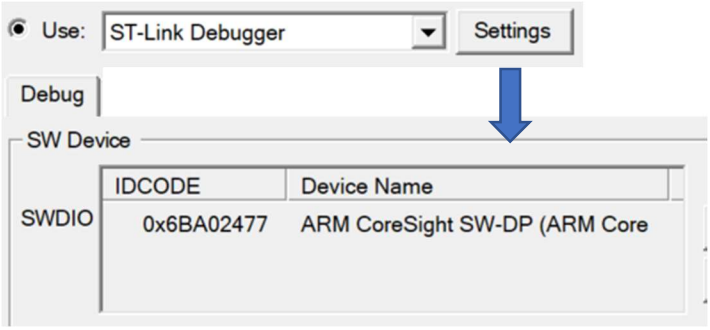
What the `#ifdef` accomplishes is that it allows you to use that C header file with your C++ code, because the macro `__cplusplus` will be defined. So, the C++ header files should be before the `#ifdef` or after the `#endif`.

TIPS: To restore the default µVision window configuration, click on Window→ Reset window to default → Restart.

D - Test the code and measure latency

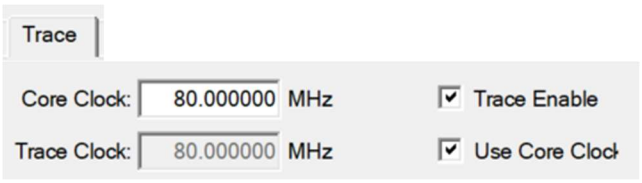
Connect your board to you PC with a USB cable, as shown in the figure.

In the Debug tab, make sure that the ST-Link Debugger is selected, the initialization File can remain unspecified if you're not interested in measuring the energy consumption.
If the board is connected, when clicking on the Settings button, you should have the following configuration, in the SW Device tab:

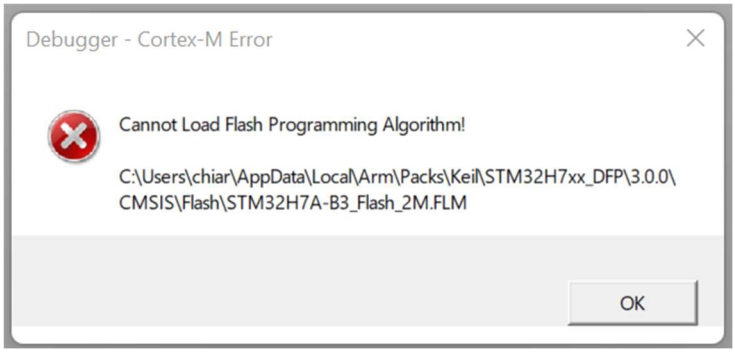


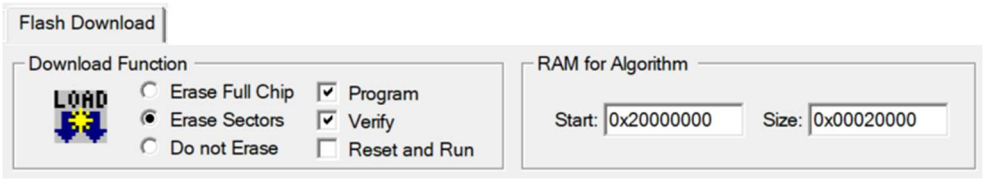
If an IDCODE doesn't show up, the device is not connected properly.

In the Trace tab, update the clock frequency (in this case, 80 MHz), and enable the Trace.

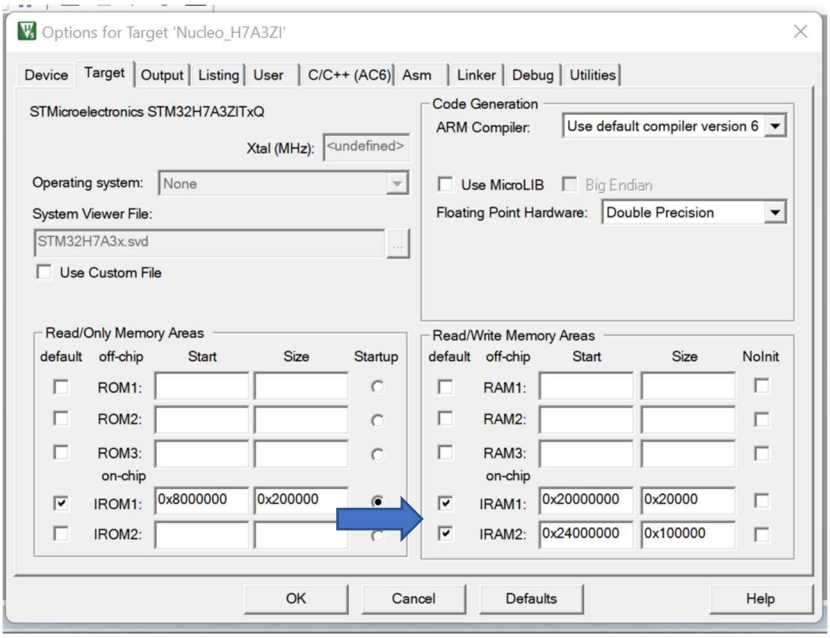


In the “Flash Download” tab, increase the size of the RAM for the Algorithm, you can also skip this step, but if you see the following error then you must increase the size.



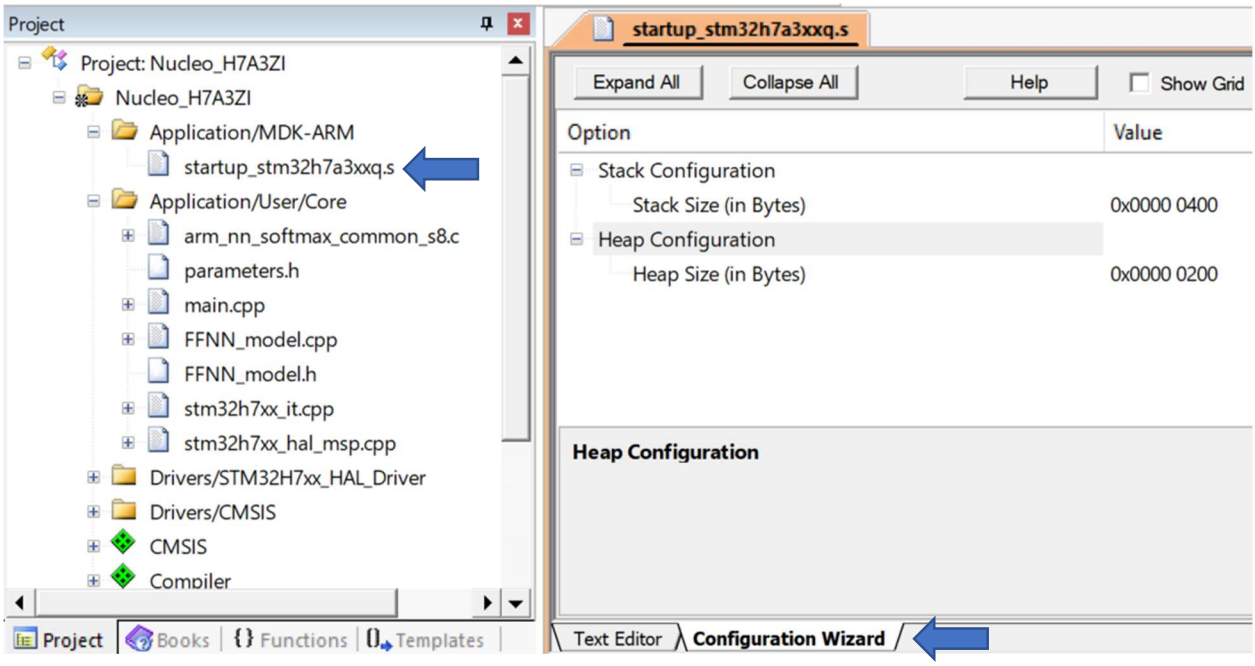


Choose a size that is not bigger than your RAM, you can find your RAM size in Option for Target → Target → Read/Write Memory Areas (see the following figure).



To enter in the debug mode, press the debug button , the Flash will be programmed. Flash programming progress will be indicated in the Output Window. The led LD4 will blink red and green indicating traffic over ST-Link V2. To run the program, press Run , to stop the program execution press Stop .

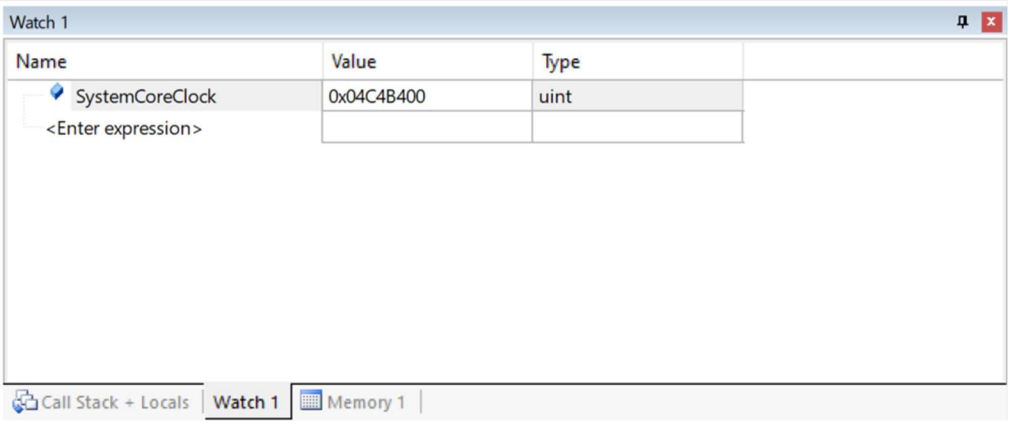
TIPS: If the code is not working properly, e.g., it gets stuck somewhere, there is the chance that the stack and heap values must be increased. To change the stack and heap value, open the startup_stm32h7a3xxq.s file, click on configuration wizard, and expand Stack Configuration and Heap Configuration.



If you want to know more about the memory organization, or to understand how much stack and heap you need, look at the chapter 4 in this tutorial.

Watch Window

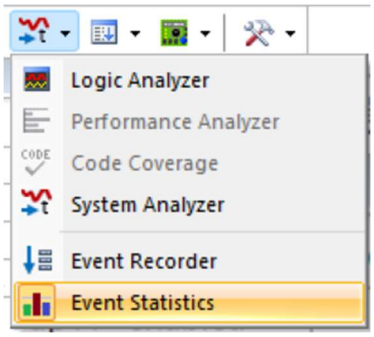
In Debug mode, if you're interested in the value of a variable in real-time, you can use the Watch Window. Watch Window will display variables in real-time as your program is running. You can display global and static variables and structures but not local variables since they are not always in focus. Memory and peripheral locations can also be displayed. To open a watch window, click on the watch window icon . Digiting "SystemCoreClock", for example, we're able to see the clock value (SystemCoreClock is a variable defined in the system_stm32h7xx.c file).



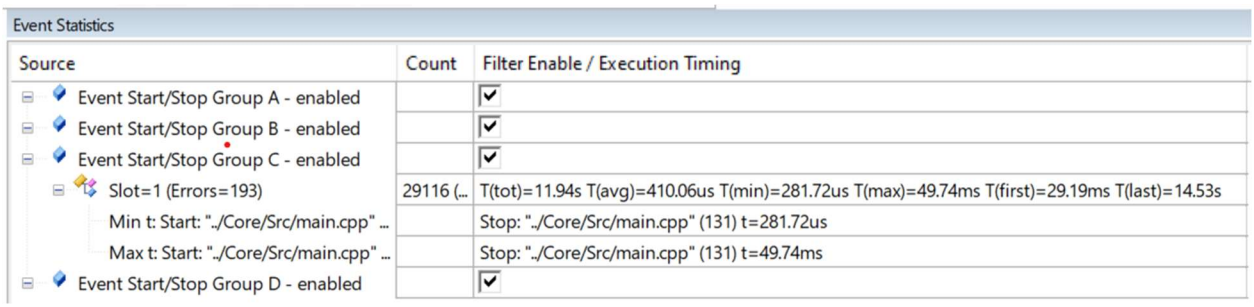
Event Statistics

You can add Start and Stop events to your source code. Information collected will include execution counts and execution time. If you are using ULINK*plus*, information will also include voltage, current and total charge (Q) consumed. Individual and aggregated times are provided. This information will be collected between the Start and Stop including the execution of any exception handlers or program branches. Data is collected from START to the corresponding STOP event. In this project code, a Start/Stop event is placed right before/after the inference execution.

To see the statistics of latency and energy consumption (if ULINK*plus* is connected) click on the System Analyzer icon and select Event Statistics.

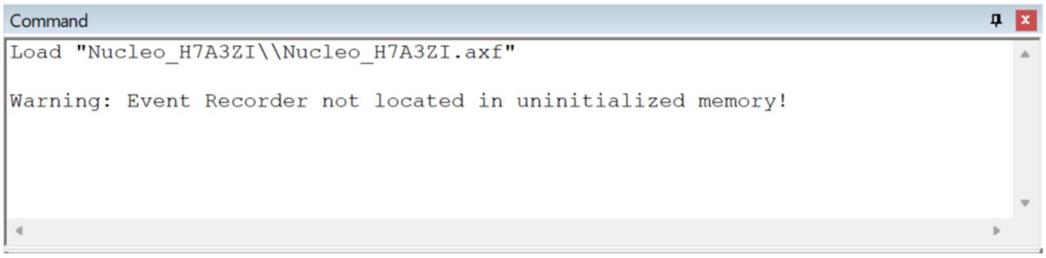


The Event Statistics window will show up, during the program execution the statistics will be calculated.



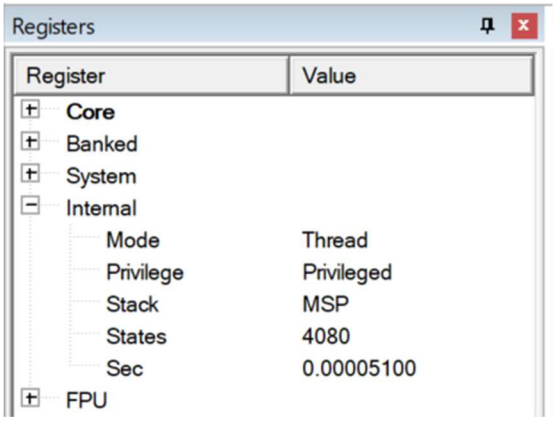
In the Event Statistics window, you can see the minimum, maximum and average time latency.

TIPS: It is possible that the following warning appears in the command window:



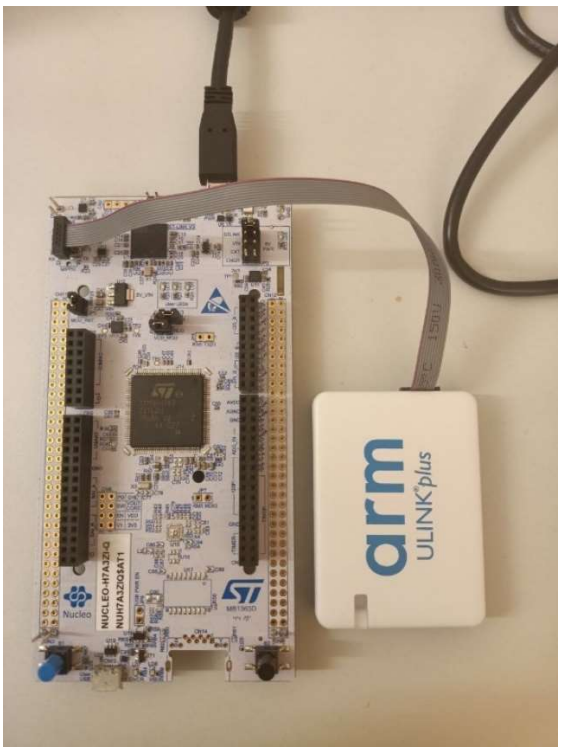
If this happens, follow the instructions at [70] in the "Locate Event Recorder in uninitialized memory" section.

An alternative to the Event Statistics to measure the latency is looking at the value of "States" in register tab, which shows the number of clock cycles.

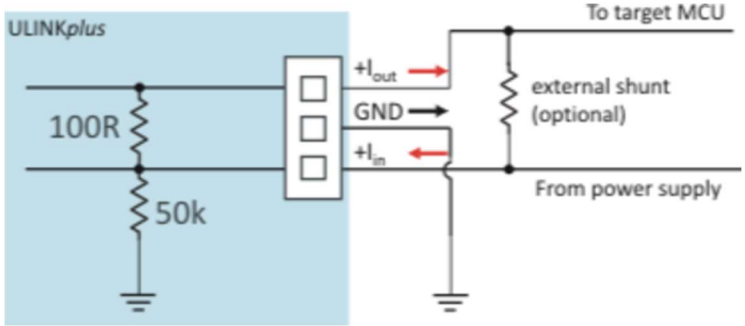


E – Measure the energy consumption

In this section we will show how to measure the energy consumption of the STM32H7A3ZI MCU. The first step is connecting the ULINK*plus* debugger to the board. Connect the 10-pin cable to the ULINK*plus* device and attach the other side of the cable to the 10-pin Cortex debug connector on the MIPI10 of the NUCLEO-H7A3ZI board, as shown in the picture.

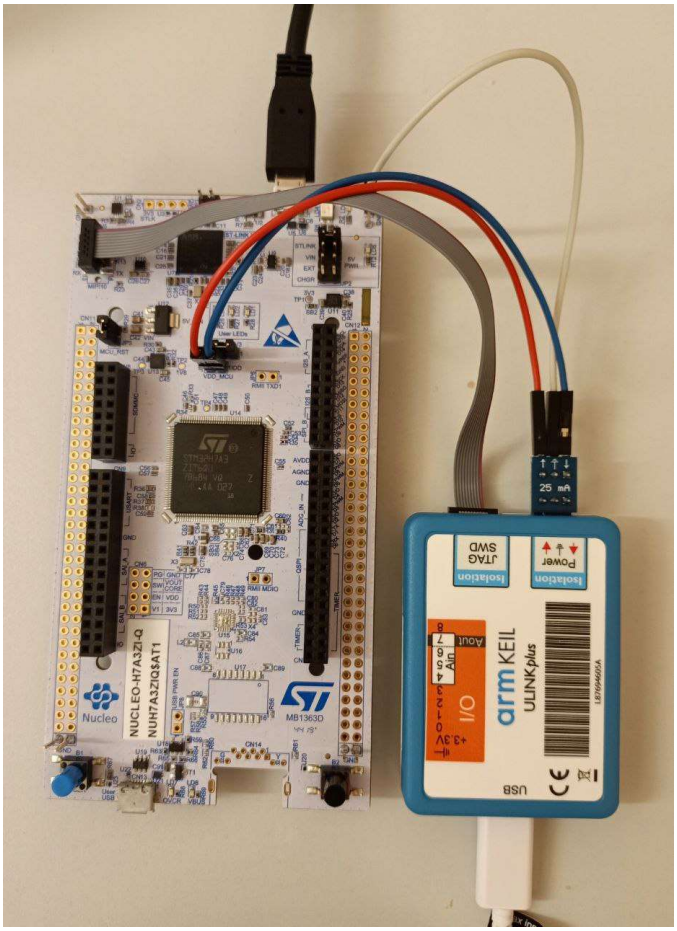


The 3-pin Power connector (with a 2.54 mm pitch) connects to the power supply circuit of the target system as shown on schematic figure. The current range with the ULINK*plus*-internal shunt resistor is +/- 2.5mA. This range can be extended using external shunt resistors. ULINK*plus* comes with six shunt boards (5 mA, 10 mA, 25 mA, 50 mA, 100 mA or 250 mA) that adjust the maximum current range to be measured [76].



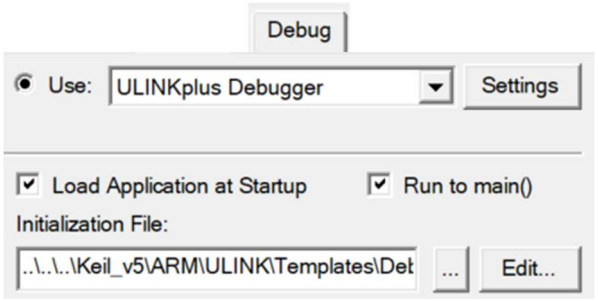
Select the shunt with a current higher that 5mA and attach it to the ULINK*plus*, following the arrows direction in the back of the device.

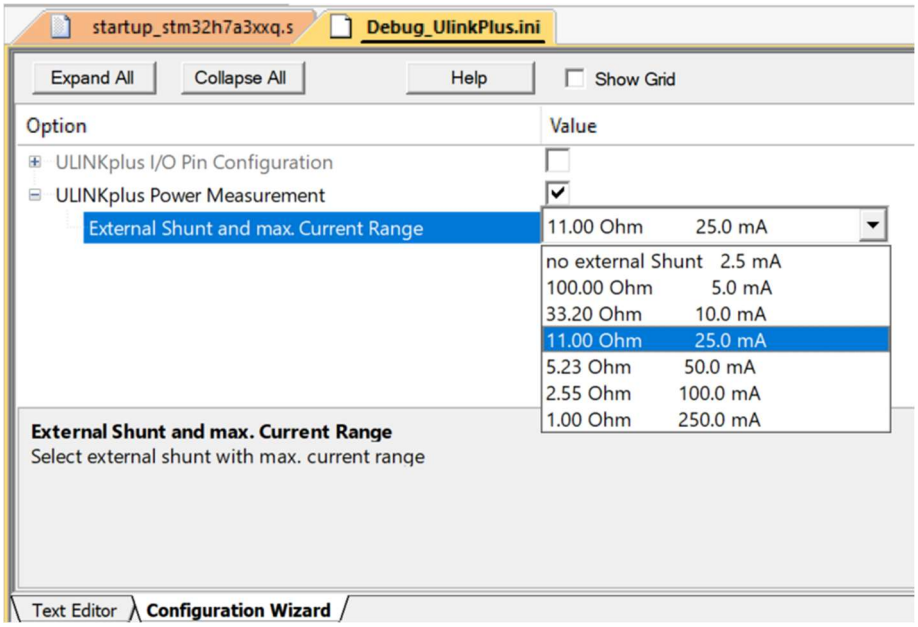
Remove the jumper JP4 and connect the shunt to the VDD, IDD and ground pins.



TIPS: If you reverse the shunt wires the current value will be negative.
TIPS: if you select the wrong shunt, a current value out of range will be displayed in the System analyzer window (see the figure).

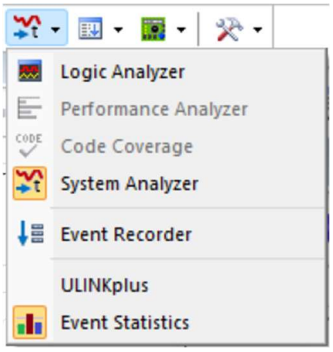
The configuration is now ready, connect the board and the debugger via USB to the PC. Click on Option for Target → Debug and select the ULINK*plus* debugger. The initialization file must be selected, the .ini file should be in the folder [MDK installation_path]\ARM\ULINK\Templates, the default [MDK installation_path] is C:\Keil_v5. The .ini file must be edited, click on "Edit" and the file will open. Click on "Configuration wizard" and expand "ULINK*plus* Power Measurement", then select the correct shunt value.

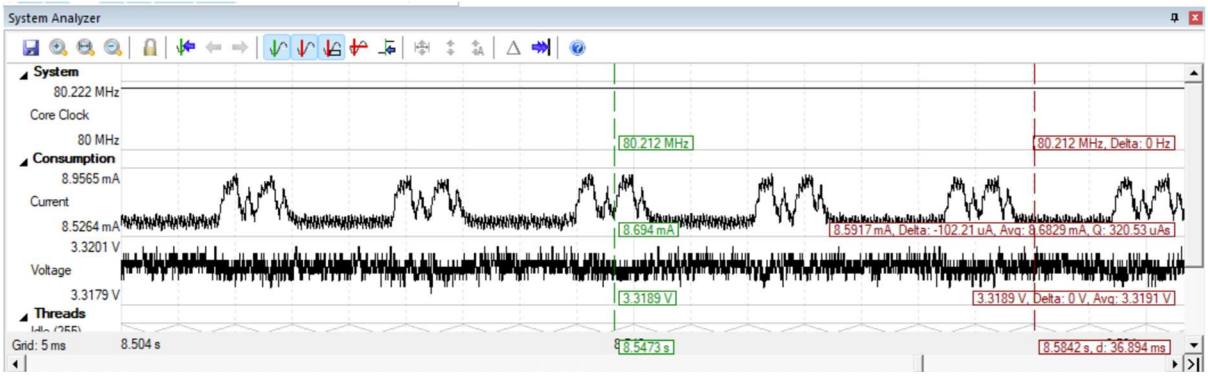




Then click on “Settings” in the Debug tab, the IDCODE should be displayed, otherwise there’s a connection error. In the Trace tab, update the Core clock value to 80MHz and enable the Trace. Then click on Rebuild .

The next step is starting a debug session and run the program. In the system analyzer window click on System Analyzer to see the current and the voltage signals. Click on the lock to stop the data collection. If you click on the Event Statistics, the minimum, maximum and average consumption will be displayed.





TIPS: If the current and the voltage values and signals are not displayed, follow the instructions at this link [74].

Event Statistics

Source	Count	Filter Enable / Execution Timing
Event Start/Stop Group A - ...		☑
Event Start/Stop Group B - ...		☑
Event Start/Stop Group C - ...		☑
Slot=1 (Errors=450)	43472 (+1)	T(tot)=13.12s Q(tot)=112.02mAs T(avg)=301.81us T(min)=269.64us T(max)=7.64ms T(first)=2.55ms T(last)=15.36s
Min t: Start: "../Core/Sr...		Stop: "../Core/Src/main.c" (108) t=269.64us Q=2.33uAs I=8.64mA U=3.32V
Max t: Start: "../Core/Sr...		Stop: "../Core/Src/main.c" (108) t=7.64ms Q=64.77uAs I=8.48mA U=3.32V
Min Q: Start: "../Core/Sr...		Stop: "../Core/Src/main.c" (108) t=269.82us Q=1.35uAs I=4.99mA U=3.32V
Max Q: Start: "../Core/S...		Stop: "../Core/Src/main.c" (108) t=7.64ms Q=64.77uAs I=8.48mA U=3.32V
Event Start/Stop Group D - ...		☑

WARNING: There are many tools that we haven’t cited in this tutorial, for a more complete tutorial see [65].

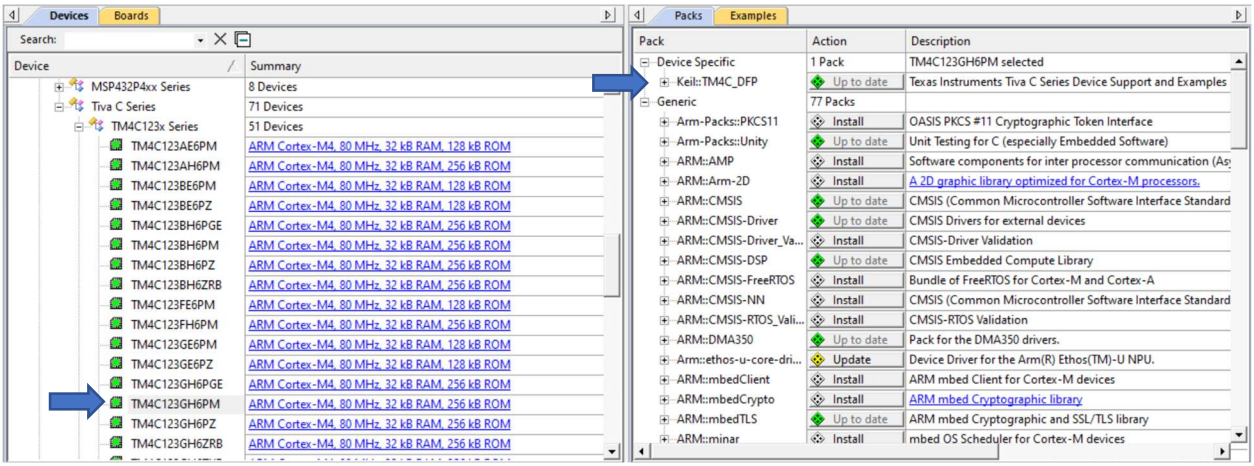
2. Tiva C latency and energy measures

This chapter describes the steps for the project creation, latency and energy measurements when using the TM4C123GH6PM MCU.

A. Create a project and include the code

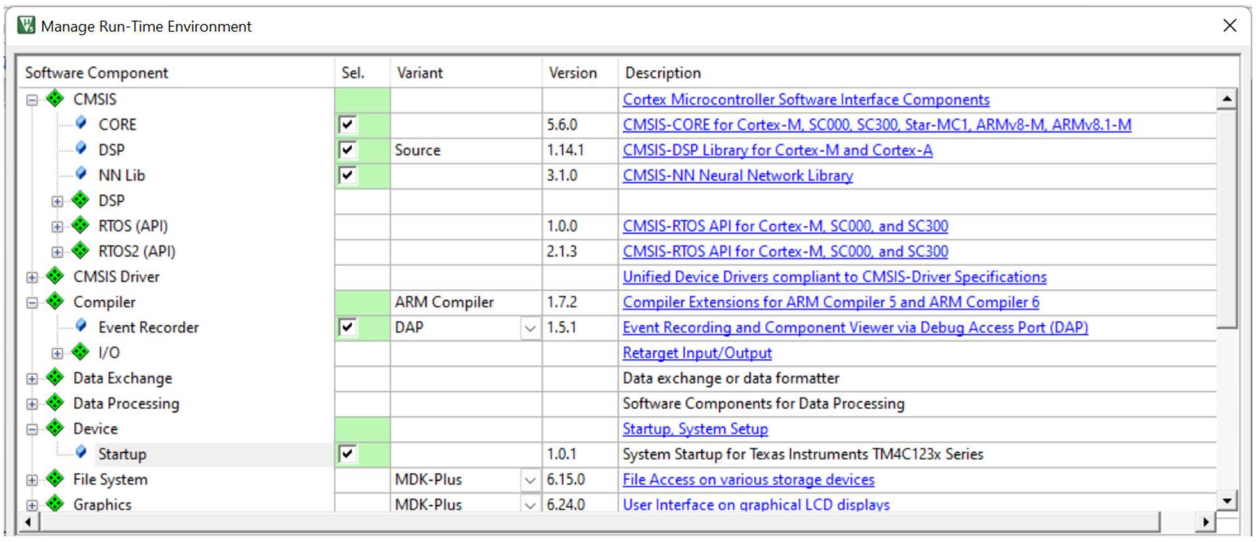
Before creating a project, you should open the Pack Installer and download the necessary packs. For the download the computer must be connected to Internet. The Pack Installer is usually in the folder [MDK installation_path]\UV4\, the default installation path is C:\Keil_v5.

Open the Pack Installer by clicking on its icon: . Search for the correct MCU and download the following packs: Keil::TM4C_DFP, ARM::CMSIS, ARM::CMSIS-Driver, ARM::CMSIS-DSP (I haven't installed ARM::CMSIS-NN and I suggest not to install it), Keil::ARM_Compiler. If you want to use the TFLite library, also install tensorflow::flatbuffers, tensorflow::gemmlowp, tensorflow::kissfft, tensorflow::ruy, tensorflow::tensorflow-lite-micro.



To create a project for the Tiva C board open µVision, click on Project → New µVision Project. Choose a folder for your project. After this step, you will be asked to select the device, digit the name of the MCU in "Search", click on the chosen MCU and then "Ok".

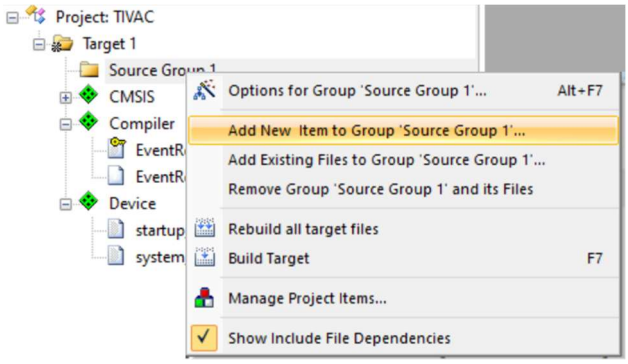
The Manage Run-Time Environment window will automatically open. Expand "CMSIS" and select CORE, DSP, NN Lib. Expand "Compiler" and select Event Recorder. Under "Device", select "Startup" (see the following figure). The latter allows you to have access to the system setup functions.



Only if you're using TFLite Library, expand "Data Exchange" and select flatbuffers. Then expand "Data Processing" and select the three packs under "Math". Also select Kernel Utils, Kernel, and Testing from Machine Learning → TensorFlow.

Software Component	Sel.	Variant	Version	Description
Data Exchange				Data exchange or data formatter
Serialization				Data Serializer Components
flatbuffers	☑	tensorflow	1.22.8	Flatbuffers
Data Processing				Software Components for Data Processing
Math				Math Components
gemmlowp fixed-point	☑	tensorflow	1.22.8	Jsmn
kissfft	☑	tensorflow	1.22.8	KISS FFT
ruy	☑	tensorflow	1.22.8	Flatbuffers
Device				Startup, System Setup
File System		MDK-Plus	6.15.0	File Access on various storage devices
Graphics		MDK-Plus	6.24.0	User Interface on graphical LCD displays
Machine Learning				Software Components for Machine Learning
NPU Support				
TensorFlow				
Kernel Utils	☑		1.22.8	TensorFlow Lite Micro Library
Kernel	☑	CMSIS-NN	1.22.8	TensorFlow Lite Micro Library
Testing	☑		1.22.8	TensorFlow Lite Micro Library

Right clicking on Source Group 1, you can add new source and header files.



As explained in chapter 1. section A, under "Include CMSIS-NN code", depending on your CMSIS Core version, it's possible that a file it's missing in the CMSIS-NN library (arm_nn_softmax_common_s8.c). Check if this file is included in you project (open CMSIS in the project tab), if it's not, right click on the project folder "Source Group 1" and then in "Add New Item". Then select a C type of file, name it "arm_nn_softmax_common_s8" and press "Add". Then click on this link [68], copy the code and paste it in the newly created file .c, in the µVision project. The version 5.9.0 of the CMSIS Core should not have this problem.

Then add the main.c, NeuralNetwork.c, NeuralNetwork.h and parameters.h files if you're only using the CMSIS-NN library, without TFLite. Add main.cpp, FFNN_model.cpp, FFNN_model.h and parameters.h if you're interested in using TFLite.

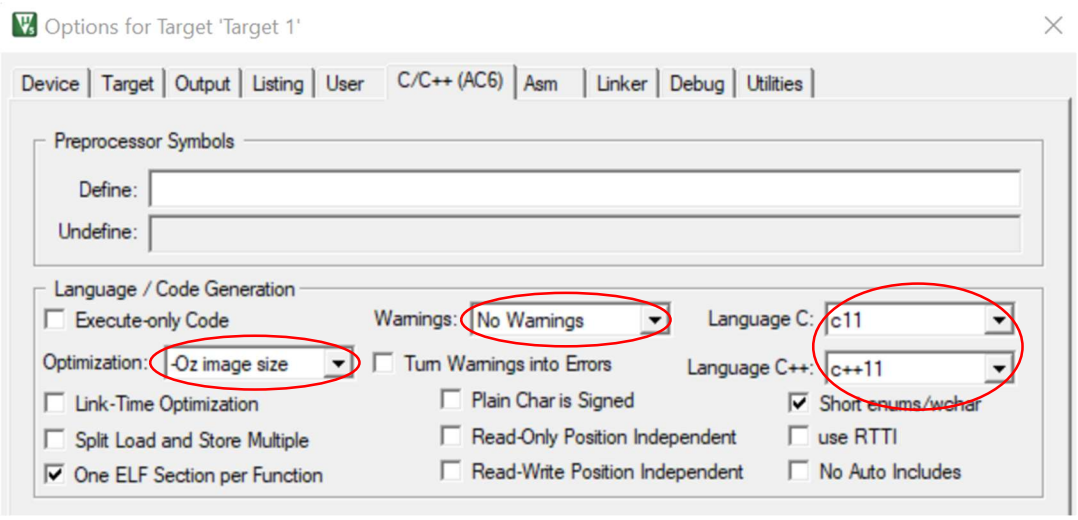
<u>Only when using TFLite</u>:
1) In the micro_time.cpp make sure that TM4C123.h header file is included, if it's not, include the header file.
2) The following error might pop up:

```
"RTE/Machine_Learning/micro_time.cpp(55): error: functions that differ only in
their return type cannot be overloaded
int32_t GetCurrentTimeTicks() {

C:/[…]/tensorflow/lite/micro/micro_time.h(27): note: previous declaration is
here
uint32_t GetCurrentTimeTicks();"
```

To solve this problem, open the micro_time.cpp file and check the returned type of ticks_per_second() and GetCurrentTimeTicks() functions. If the returned type doesn't not correspond to the one in micro_time.h, change it. In my case, I had to change the returned type from int32_t to uint32_t.

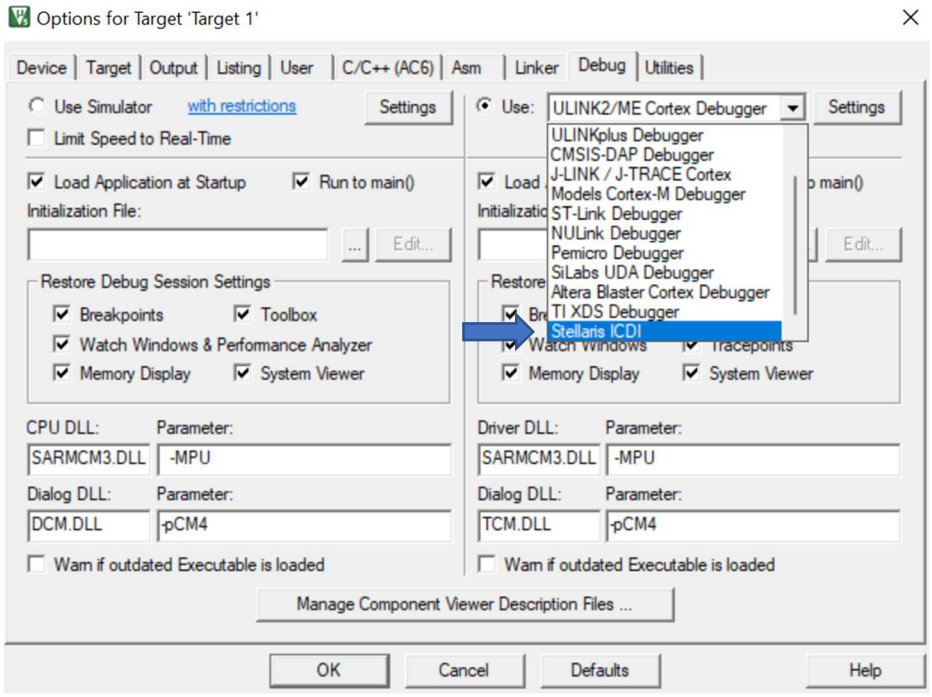
3) The TFLite library memory footprint is too high for the TM4C123 MCU, for this reason you must shrink the compiled code as much as possible, to obtain this select the Oz image size in Option for Target icon → C/C++(AC6). In the same tab, change to C11 and C++11 the language.

To build the code press the Build icon . When compiling, it's possible that a lot of warnings are displayed, if this is slowing down too much the compilation, click in Option for Target icon → C/C++(AC6), under Language/Code Generation select "no Warnings" in the Warnings drop down menu (see the picture).



B. Test the code and measure the latency

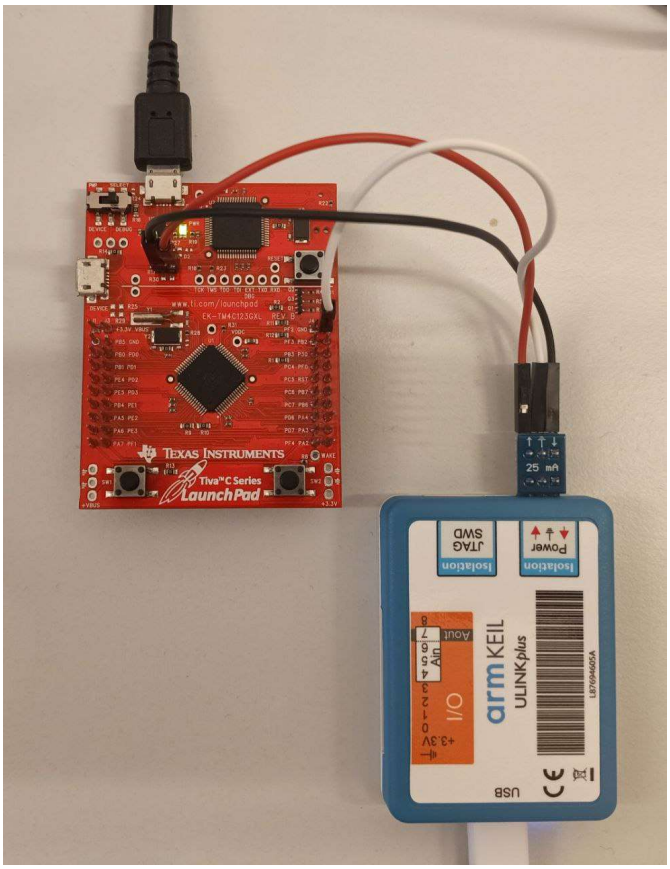
Before flashing the code into the MCU, you should download an add-on installer that supports the Stellaris ICDI connection. Download the add-on installer from [81]. When you're done with that, you should be able to see the Stellaris ICDI option in the drop-down menu in the Debug tab.



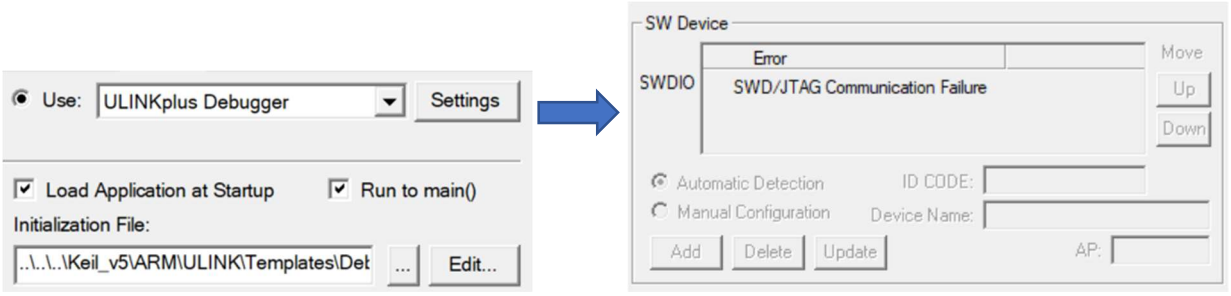
Select it and connect the Tiva C board to the PC via USB. Click on the Rebuild icon and enter the debug mode pressing the debug button . The Flash will be programmed, Flash programming progress will be indicated in the Output Window. To run the program, press Run , to stop the program execution press Stop . To gain more insight into the debug functionalities, read the chapter 1, section D of this tutorial, where the Watch Window and the Event Statistics functionalities are explained.
As said in the chapter 1, section D, you must pay attention to have enough space in the stack and heap, you can change the stack and heap size from the startup_TM4C123.s file, see section D for more details.

C – Measure the energy consumption

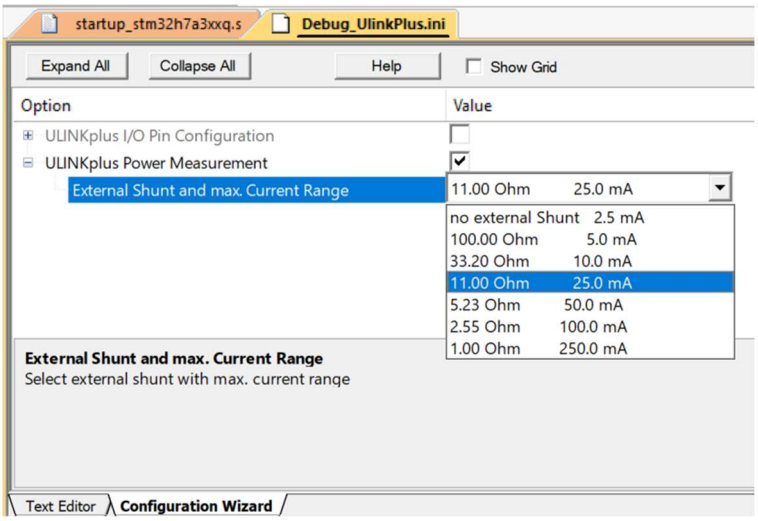
In this section we will show how to measure the energy consumption of the TM4C123 MCU. The first step is flashing the code into the MCU clicking on the Flash icon . When the flash is done, connect the ULINK*plus* debugger to the board. As for the NUCLEO board, section E, choose an appropriate shunt (>10 mA) and connect it to the ULINK*plus*, following the arrows direction in the back of the device. Connect the shunt to the VDD, IDD and ground pins with jumpers, as shown in the picture. If you reverse the jump wires, the current will be negative.



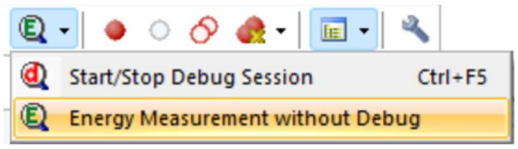
The configuration is now ready, connect the board and the debugger via USB to the PC. As seen for the NUCLEO board, click on Option for Target → Debug and select the ULINK*plus* debugger. The initialization file must be selected, this .ini file should be in the folder [MDK installation_path]\ARM\ULINK\Templates, the default [MDK installation_path] is C:\Keil_v5.



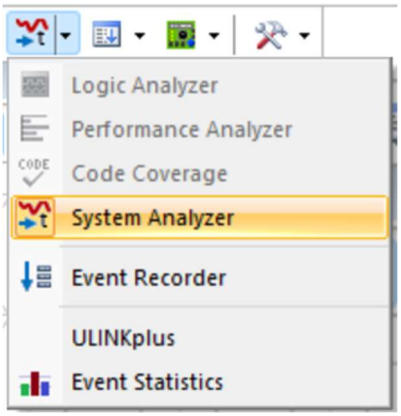
In Settings → Debug, under "SW Device", it will be displayed a communication failure, it should be like this, since we're measuring the energy consumption without a debug, i.e., we won't flash the code through ULINK*plus*, we'll only measure the energy consumption.
The .ini file must be edited, click on "Edit" and the file will open. Click on "Configuration wizard" and expand "ULINK*plus* Power Measurement", then select the correct shunt value.



Save the changes in the .ini file and rebuild the program.
Click on the options close to the debug button and select Energy Measurement without Debug:



Click on the “System Analyzer” button and select System Analyzer to be able to see the Current and Voltage signals and values.



Click on Run to start the data acquisition, click on the lock to stop the data collection.

Press the Energy measurement without button  to exit the measurement mode.

TIPS: If the current and the voltage values and signals are not displayed, follow the instructions at this link [74].

TIPS: If it's displayed that the current value is out of range, choose a shunt with a bigger current.

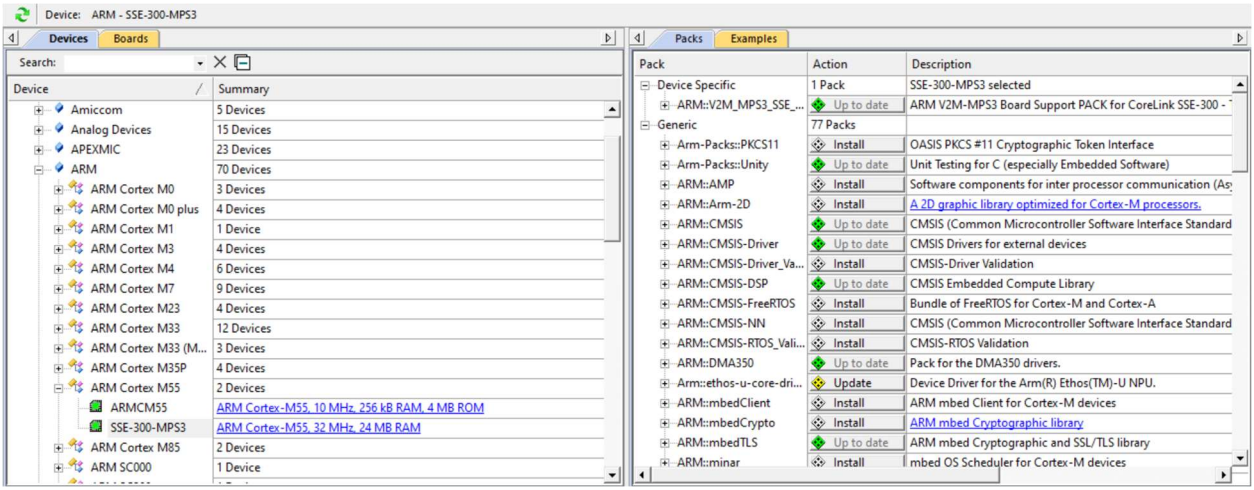
3. Virtual Hardware Target (VHT)

This chapter focuses on the setup of a project with a virtual hardware target. The VHT chosen is a system with a Cortex-M55 processor. This chapter will only show how to measure the clock cycles, it's not possible to measure the energy consumption when using a VHT.

A. Create a project and include the code

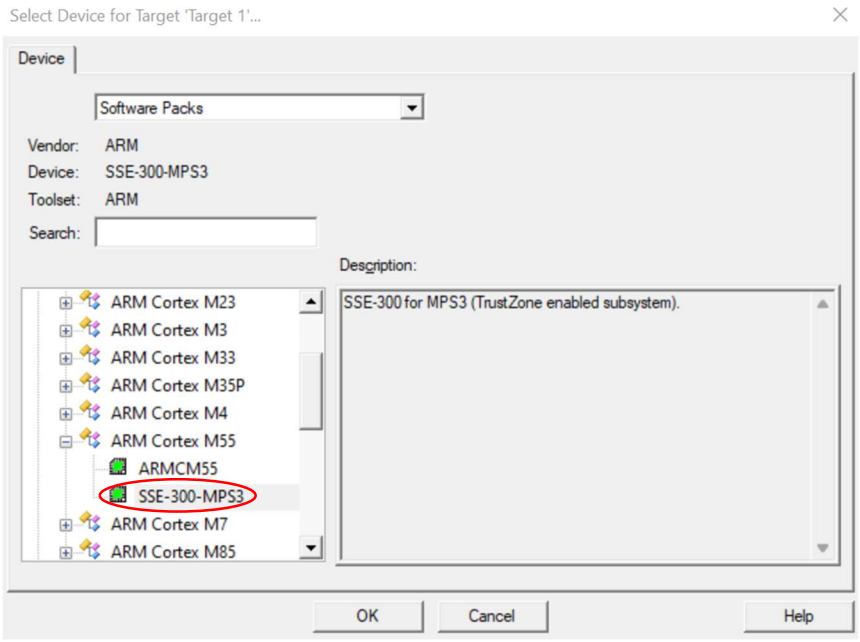
Before creating a project open the Pack Installer and download the necessary packs. For the download the computer must be connected to Internet. The Pack Installer is usually in the folder [MDK installation_path]\UV4\, the default installation path is C:\Keil_v5.

Open the Pack Installer by clicking on its icon:  . Search for the correct MCU (see the picture below) and download the following packs: ARM::V2M_MPS3_SSE_300_BSP, ARM::CMSIS, ARM::CMSIS-Driver, ARM::CMSIS-DSP (I haven't installed ARM::CMSIS-NN and I suggest not to install it), Keil::ARM_Compiler.
If you want to use the TFLite library, also install tensorflow::flatbuffers, tensorflow::gemmlowp, tensorflow::kissfft, tensorflow::ruy, tensorflow::tensorflow-lite-micro.

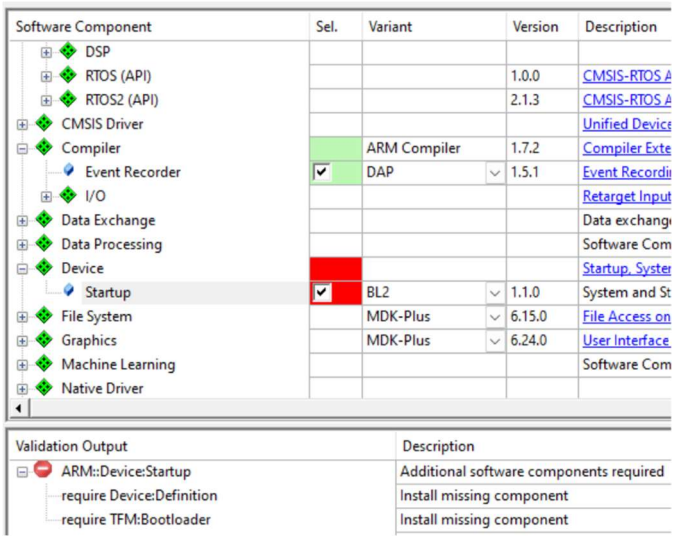


Note: If you don't install some necessary packages, in the window called "Manage Run-Time Environment" the rectangle of the corresponding package will become red as you try to include that component.

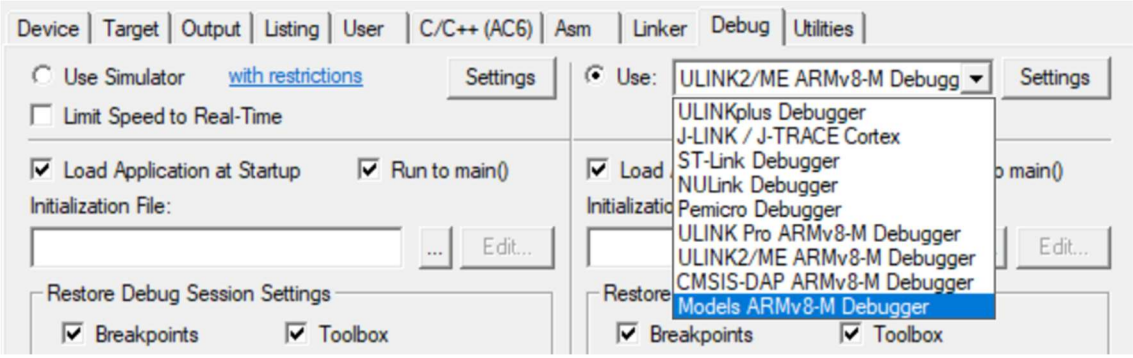
To create a project for the VHT open µVision, click on Project → New µVision Project. Choose a folder for your project. After this step, you will be asked to select the device, digit the name of the MCU in "Search", click on the chosen MCU and then "Ok".



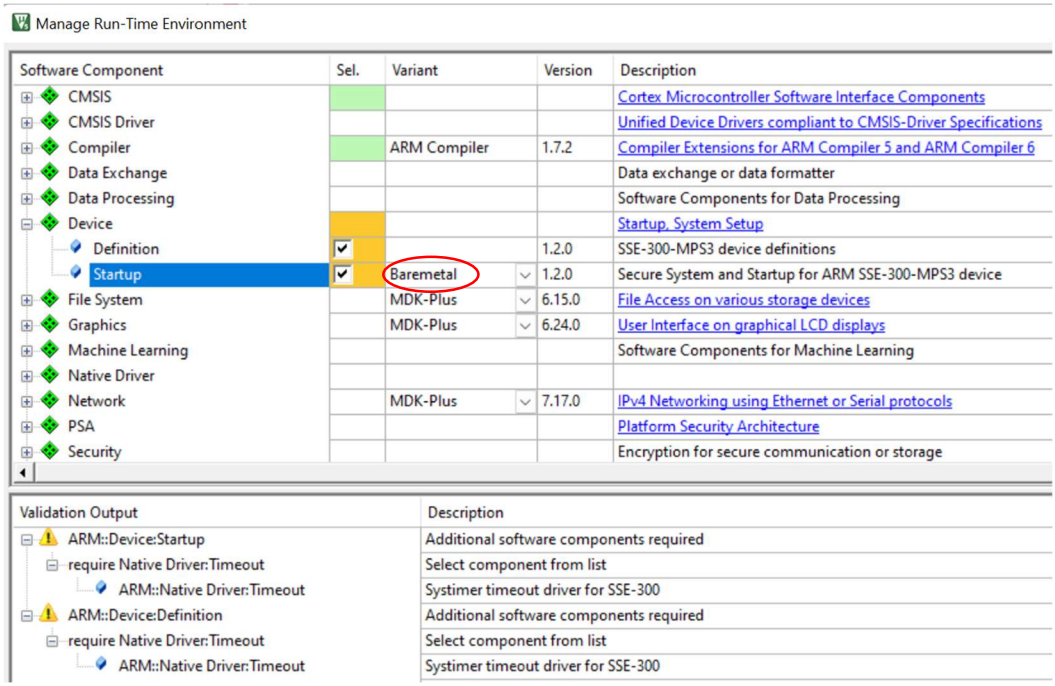
The "Manage Run-Time Environment" window will automatically open, from the window, expand "CMSIS" and select CORE, DSP, NN Lib. Expand "Compiler" and select Event Recorder. Under "Device", select "Startup". If, when selecting "Startup", the Validation Output displays that you need TFM:Bootloader and Device:Definition as in the picture below, click "Ok" and go in Option for Target → Debug.



In the Debug tab, choose Models ARMv8-M Debugger and click “Ok” to save your choice.

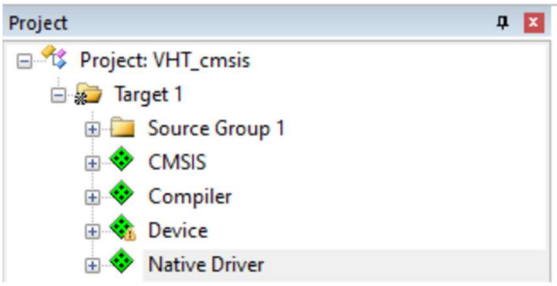


Now click on the Manage Run-Time Environment icon and expand “Device”. Select “Definition” and choose the baremetal option for “Startup” in the drop-down menu.
Then, add the components requested in the Validation Output and click “Ok”.



Right clicking on Source Group 1, you can add new source and header files.
As explained in chapter 1 section A, under “Include CMSIS-NN code”, depending on your CMSIS Core version, it’s possible that a file it’s missing in the CMSIS-NN library (arm_nn_softmax_common_s8.c). Check if this file is included in you project (open CMSIS in the project tab), if it’s not, right click on the project folder “Source Group 1” and then in “Add New Item”. Then select a C type of file, name it “arm_nn_softmax_common_s8” and press “Add”. Then click on this link [68], copy the code and paste it in the newly created file .c, in the µVision project.

The version 5.9.0 of the CMSIS Core should not have this problem.

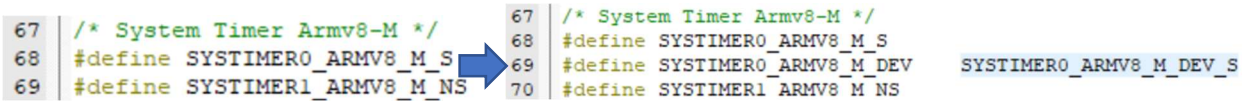


Then add the main.c, NeuralNetwork.c, NeuralNetwork.h and parameters.h files if you're only using the CMSIS-NN library, without TFLite. Add main.cpp, FFNN_model.cpp, FFNN_model.h and parameters.h if you're interested in using TFLite.

When compiling, some common errors might show up:

1) It's possible that the header file RTE_Device.h has the following warning: "comments are not allowed in this language". If this happens, it means that you're using C90. In Option for Target → C/C++ select another C language (e.g., C99). Build the program clicking on the Build icon .

2) A common error is "`use of undeclared identifier 'SYSTIMER0_ARMV8_M_DEV'`" in the file systimer_armv8-m_timeout.c., to solve this error open the header file device_cfg.h and add the definition "`#define SYSTIMER0_ARMV8_M_DEV SYSTIMER0_ARMV8_M_DEV_S`" below line 68, as in the picture:

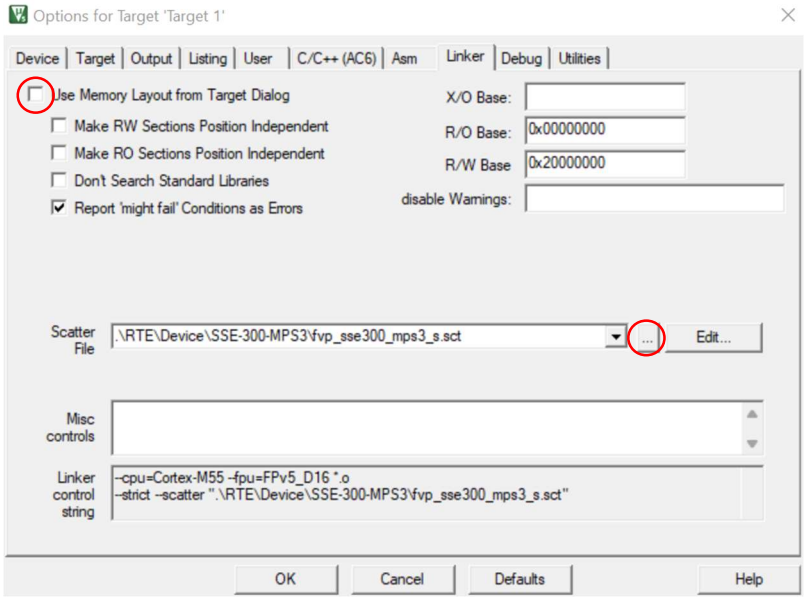


3) Another common error is a Linker Error, such as the following:

```
linking...
.\Objects\M55_Test.axf: Error: L6630E: Invalid token start expected number or ( but found n at position 5 on line 5
.\Objects\M55_Test.axf: Error: L6629E: Unmatched parentheses expecting ) but found n at position 5 on line 5
.\Objects\M55_Test.axf: Error: L6226E: Missing base address for region LR_.
.\Objects\M55_Test.sct(5): error: L6292E: Ignoring unknown attribute 'null' specified for region LR_.
.\Objects\M55_Test.sct(5): error: L6228E: Expected '{', found ')...'.
.\Objects\M55_Test.sct: Error: L6372E: Image needs at least one load region.
```

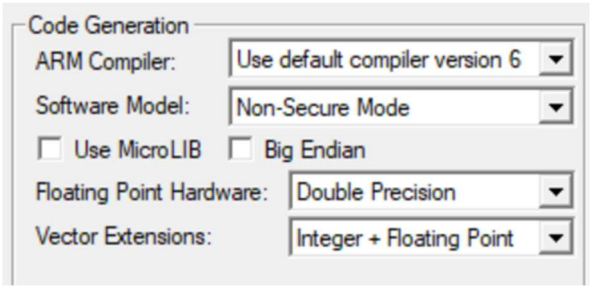
A possible workaround:

Click in Option for Target → Linker. Uncheck "Use memory layout from target dialog" (see the following figure). Then click on the browser button to select a proper scatter file. The path of the scatter file is [Project folder]\RTE\Device\ SSE-300-MPS3\. In this folder there should be a file that looks like this: fvp_sse300_mps3_s, select this file.



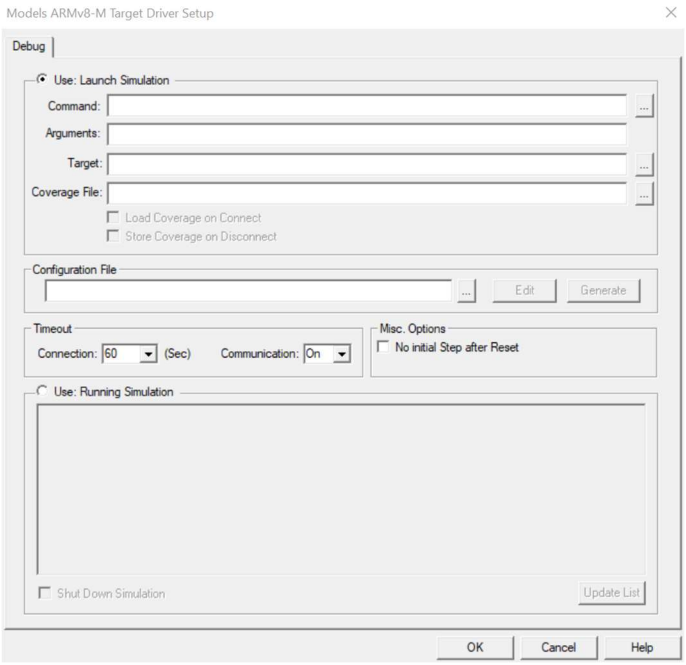
After this step, the building process shouldn't have errors. When compiling, it's possible that a lot of warnings are displayed, if this is slowing down too much the compilation, click in Option for Target icon → C/C++(AC6), under Language/Code Generation select "no Warnings" in the Warnings drop-down menu.

To enable Helium vector extension, in Option for Target → Target, select "Integer + Floating Point" in the Vector Extension drop-down menu.

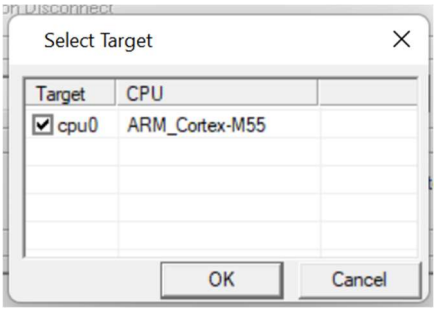


B. Test the code and measure the latency

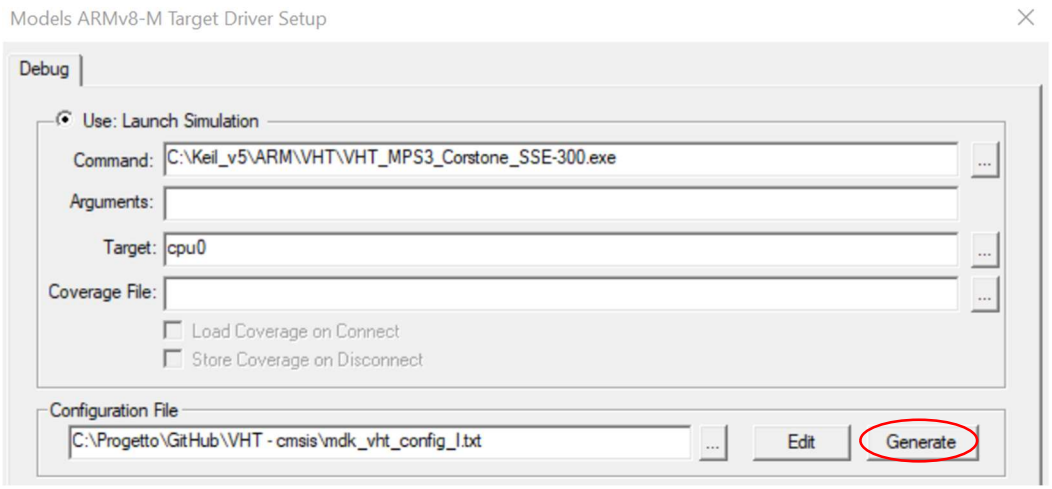
Click in Option for Target → Debug. Make sure that the Models ARM_v8-m Debugger is selected, then click on "Settings". The following window should appear:



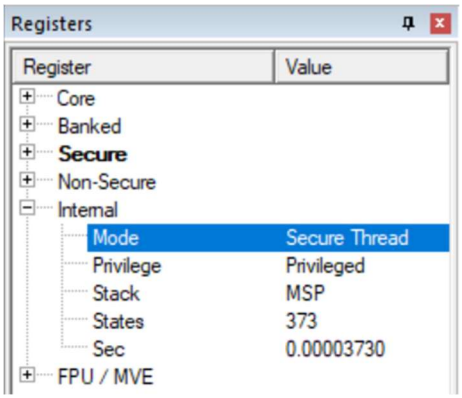
Click on the browser button close to “Command”, in the following path you should find the VHT_MPS3_Corstone_SSE-300 executable:
[MDK installation_path]\ARM\VHT\VHT_MPS3_Corstone_SSE-300.exe, the default MDK installation path is C:\Keil_v5.
Then click on the browser button close to “Target”, the following window should appear, click “Ok”.



In the project file, create an empty .txt file and select it in “Configuration file”, then click on “Generate”. Click “Ok” when it asks if you want to overwrite the file.



Enter the debug mode pressing the debug button . To run the program, press Run , to stop the program execution press Stop . "States" below Registers → Internal, shows the clock cycles number.

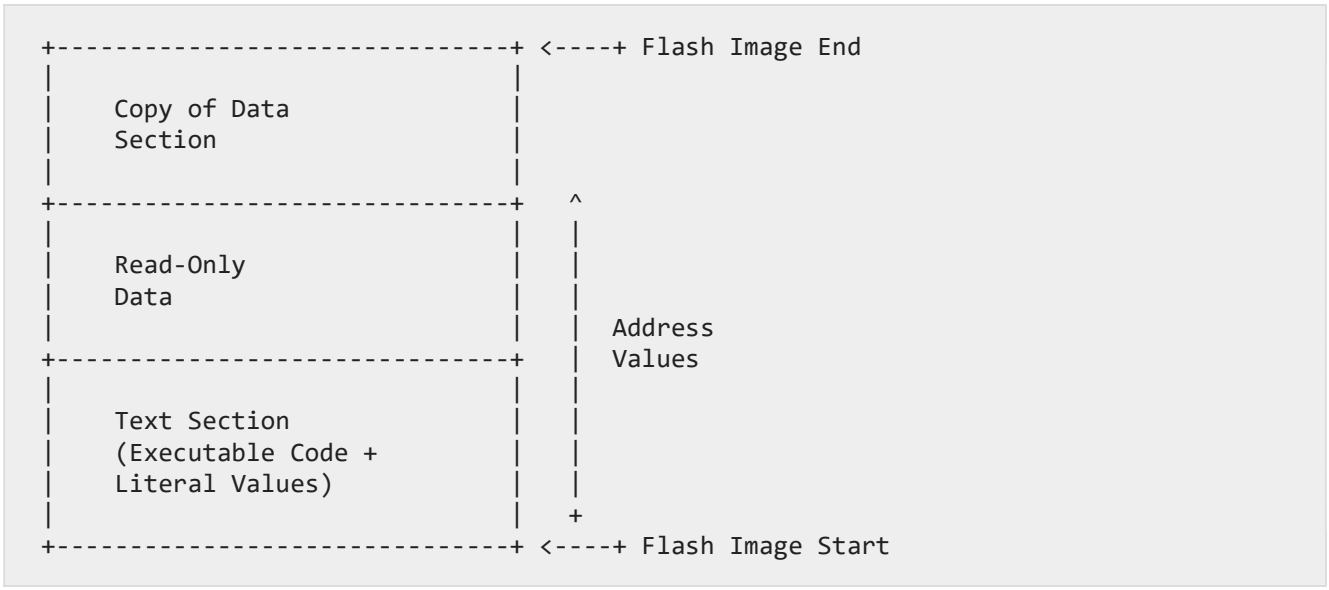


To gain more insight into the debug functionalities, read the chapter 1, section D of this tutorial, where the Watch Window and the Event Statistics functionalities are explained.

4. RAM/Flash usage in embedded C

The objective of this section is to clarify the memory organization in embedded C, regarding the ARM Cortex-M architecture. Information in this chapter were taken from [77]. The memory map is divided into a section for flash memory (code and read-only data) and a section for RAM (read-write data).

In the following diagram the map for the Flash memory is shown, code and read-only data are stored in flash memory.

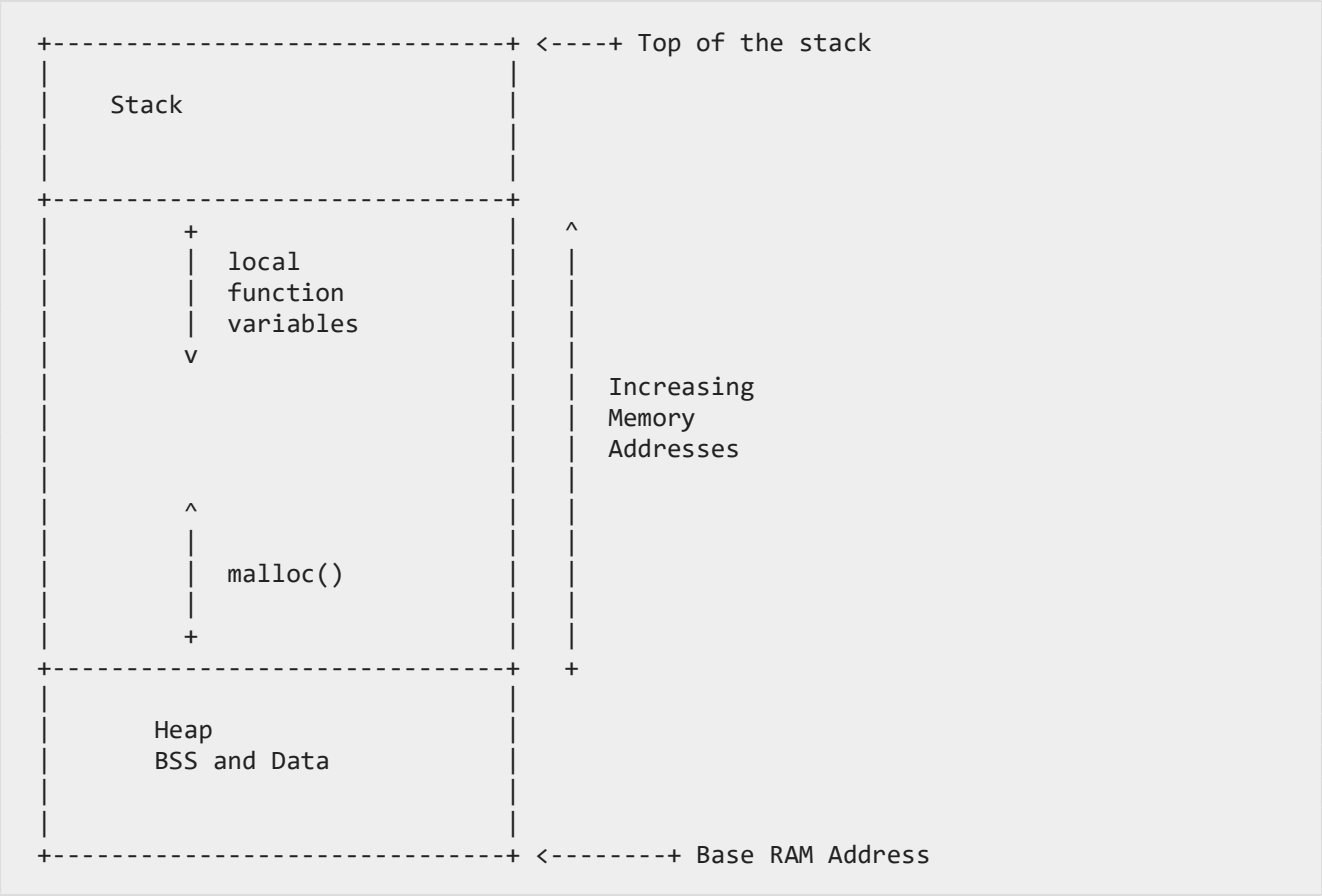


The beginning of the program (the lowest memory location at the bottom of the diagram) is the **text section** which includes executable code. the text segment is often read-only, to prevent a program from accidentally modifying its instructions [78]. This section also includes numerical values that are not assigned to any specific C variable called “literal values”.

The **read-only data section** follows the text section and is exclusively stored in flash memory (note this is only true for some embedded architectures, not all).

Then, there is a **copy of the “data” section**, which contains the initial values of statically allocated variables. This section is copied to RAM when the program starts up.

In the following diagram the map for the RAM memory is shown:



Data and BSS

Statically allocated memory means that the compiler determines the memory address of the variable at compile time. Static data is divided into two sections: data and bss (block started by symbol). The difference is that data is assigned an initial, non-zero value when the program starts while variables in the bss section are initialized to zero. For clarification, see the below example:

```
#include <stdio.h>

//these variables are globally allocated
int data_var = 500;
int bss_var0;
int bss_var1 = 0;

void my_function(void){
     int uninitialized_var;
     printf("data_var:%d, bss_var0:%d\n", data_var, bss_var0);
}
```

When the C program starts, the C runtime (CRT) start function loads the memory location assigned to data_var with 500. This is typically accomplished by copying the value from flash to RAM; this implies that each byte of data will occupy one byte of flash and one byte of RAM. The CRT (C runtime) start function then sets the memory locations for bss_var0 and bss_var1 to zero which does not require any space in flash memory.

Stack and heap

While the compiler determines the memory address of static memory at compile-time, the locations of dynamically allocated variables are determined while the program is running. The two dynamic memory constructs in C are the heap and the stack.
The stack grows down (from higher memory addresses to lower ones) and the heap grows up. If memory usage is ignored in the design, the stack and heap can collide causing one or both to become corrupted and result in a situation that can be difficult to debug. While the heap is managed by the programmer while the compiler takes care of the stack.
Heap: The C standard library contains two function families for managing the heap, malloc() and free(). Their usage with array elements can cause memory fragmentation [77].
Stack: Variables that are declared within a function (local variables), are either allocated on the stack or simply assigned a register value. Whether a variable is allocated on the stack or simply assigned to a register depends on many factors such as the compiler (including conventions associated with the architecture), the microcontroller architecture, as well as the number of variables already assigned to registers.

TIPS: If you declare an array in a function, make sure that the stack is big enough to store the array. The stack and heap values can be changed from the .s file in the project. For example, in the NeuralNetwork.c file, in the PredictFFNN() function, there is the array for intermediate results q15_t interVector[128]. To store this array the minimum stack size is 128*2 bytes.

USEFUL LINKS:
µVision User's guide: [80]
Managing Run-Time Environment: [79]
Guide for the event recorder: [71]
Guide for ULINK*plus*: [72][73]
Troubleshooting for energy measurements: [74]
Debugging with SWD: [75]
ARM VHT: [82][83]
ARM VHT Simulation: [84]

Bibliography

[62] https://www.st.com/en/microcontrollers-microprocessors/stm32h7a3zi.html
[63] https://www.ti.com/product/TM4C123GH6PM
[64] https://armkeil.blob.core.windows.net/developer/Files/pdf/white-paper/introduction-to-arm-cortex-m55-processor.pdf
[65] https://www.keil.com/appnotes/files/apnt_321.pdf
[66] **Ying Bai, 2016**. "ARM® Memory Protection Unit (MPU), " in *Practical Microcontroller Engineering with ARM Technology* , IEEE, 2016, pp.951-974, doi: 10.1002/9781119058397.ch12.
[67] https://www.st.com/resource/en/application_note/dm00425536-how-to-optimize-stm32-mcus-internal-rc-oscillator-accuracy-stmicroelectronics.pdf
[68] https://github.com/ARM-software/CMSIS-NN/blob/main/Source/SoftmaxFunctions/arm_nn_softmax_common_s8.c
[69] https://github.com/ARM-software/AVH-TFLmicrospeech
[70] https://www.keil.com/pack/doc/compiler/EventRecorder/html/er_use.html#place_uninit_memory
[71] https://www.keil.com/pack/doc/compiler/EventRecorder/html/er_use.html
[72] https://www2.keil.com/mdk5/ulink/ulinkplus/
[73] https://www.keil.com/support/man/docs/ulinkplus/default.htm
[74] https://developer.arm.com/documentation/101636/0100/Troubleshooting/Current-and-voltage-display-problems
[75] https://developer.arm.com/documentation/101636/0100/Debug-and-Trace?lang=en
[76] https://developer.arm.com/documentation/101636/0100/Power-Measurement/Power-Interface?lang=en
[77] https://blog.stratifylabs.dev/device/2013-10-18-RAM-Flash-Usage-in-Embedded-C-Programs/
[78] https://www.geeksforgeeks.org/memory-layout-of-c-program/
[79] https://developer.arm.com/documentation/101407/0538/Creating-Applications/Software-Components/Managing-Run-Time-Environment
[80] https://developer.arm.com/documentation/101407/0538?lang=en
[81] https://developer.arm.com/documentation/ka002280/latest
[82] https://arm-software.github.io/AVH/main/infrastructure/html/run_mdk_pro.html
[83] https://www.brighttalk.com/webcast/17792/527434?utm_source=ArmLtd&utm_medium=brighttalk&utm_campaign=527434
[84] https://arm-software.github.io/AVH/main/simulation/html/index.html