Tutorial Second Session

Lecturer :
Shahab Mahmoudi Sadaghiani

Email : shahab.mahmoudisadaghiani@mail.mcgill.ca

ECSE 444 - Microprocessor -Winter 2023

McGill

Agenda:

1. CMSIS-DSP Library
2. Add assembly code
3. Single Wire Output (SWO) interface
4. Lab 1 Introduction

Purpose of the Lab : Comparing the performance of a function which implemented in three different way.

Filter C code

Filter Assembly code

Filter C code by CMSIS-DSP

Agenda:

1. CMSIS-DSP Library
2. Add assembly code
3. Single Wire Output (SWO) interface
4. Lab 1 Introduction

CMSIS:

- “The **Common Microcontroller Software Interface Standard (CMSIS)** is a vendor-independent **abstraction layer** for microcontrollers that are based on Arm Cortex processors.” Ref.[1]

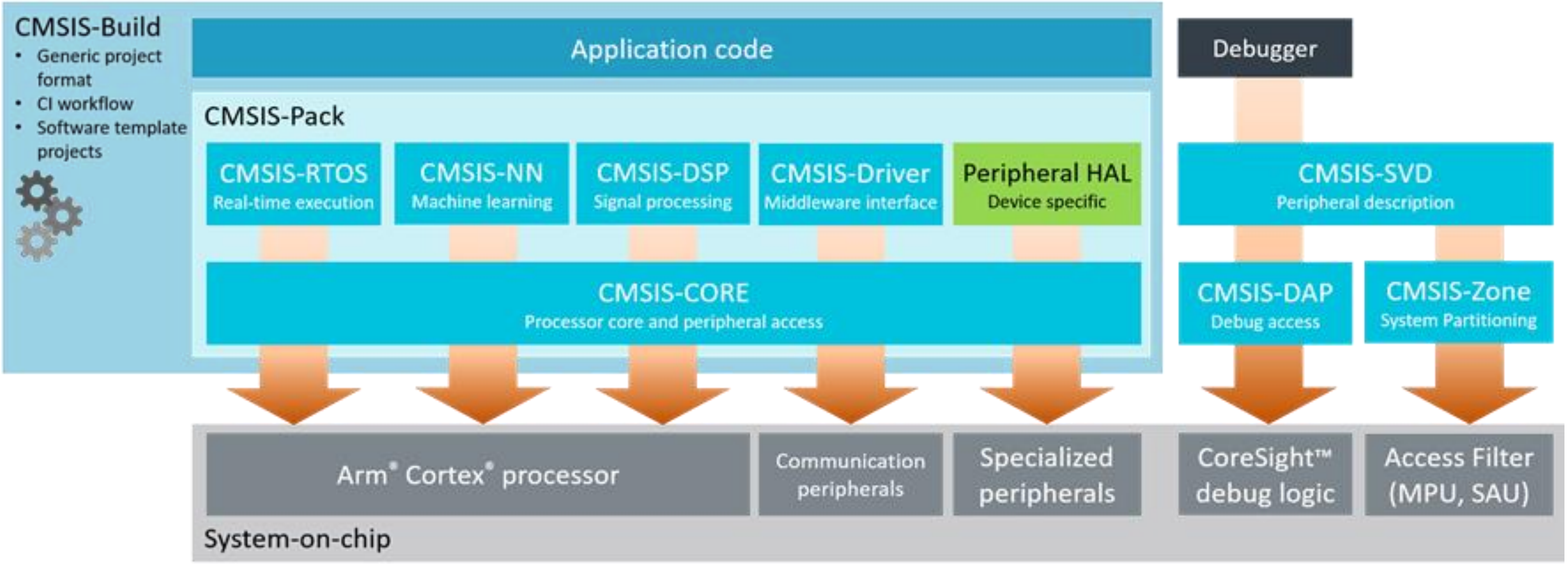
- Cortex **Microcontroller Software Interface Standard** (CMSIS) Ref. [2]

- The **CMSIS** is a set of tools, APIs, frameworks, and workflows that help to simplify software re-use.

CMSIS

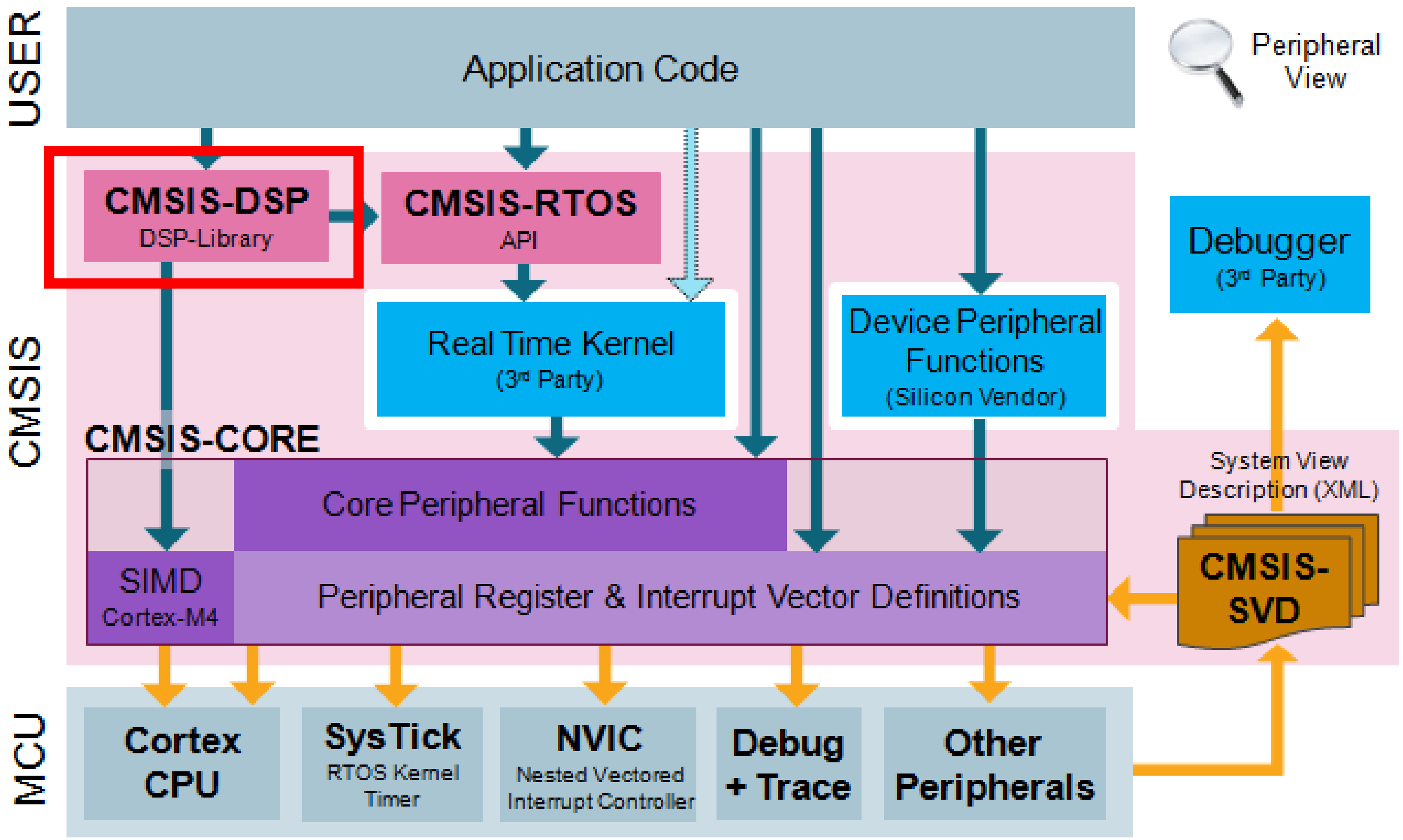
- Core(M)
- Core(A)
- Driver
- DSP
- NN
- RTOSv1
- RTOSv2
- Pack
- SVD
- DAP
- Zone

CMSIS Structure



Ref. [2]

CMSIS Structure for CortexM series:



Ref. [3]

- **CMSIS DSP Software Library**
  - Basic math functions
  - Fast math functions
  - Complex math functions
  - Filters
  - Matrix functions
  - Transforms
  - …

- **Pre-processor Macros :**

Each library project have different pre-processor macros.
UNALIGNED_SUPPORT_DISABLE:
Define macro UNALIGNED_SUPPORT_DISABLE, If the silicon does not support unaligned memory access
ARM_MATH_BIG_ENDIAN:
Define macro ARM_MATH_BIG_ENDIAN to build the library for big endian targets. By default library builds for little endian targets.
ARM_MATH_MATRIX_CHECK:
Define macro ARM_MATH_MATRIX_CHECK for checking on the input and output sizes of matrices
ARM_MATH_ROUNDING:
Define macro ARM_MATH_ROUNDING for rounding on support functions
ARM_MATH_CMx:
Define macro ARM_MATH_CM4 for building the library on Cortex-M4 target, ARM_MATH_CM3 for building library on Cortex-M3 target and ARM_MATH_CM0 for building library on cortex-M0 target.
__FPU_PRESENT:
Initialize macro __FPU_PRESENT = 1 when building on FPU supported Targets. Enable this macro for M4bf and M4lf libraries

Here is a list all modules:

- Basic Math Functions
  - Vector Absolute Value
  - Vector Addition
  - Vector Dot Product
  - Vector Multiplication
  - Vector Negate
  - Vector Offset
  - Vector Scale
  - Vector Shift
  - Vector Subtraction
- Fast Math Functions
  - Cosine
  - Sine
  - Square Root
- Complex Math Functions
  - Complex Conjugate
  - Complex Dot Product
  - Complex Magnitude
  - Complex Magnitude Squared
  - Complex-by-Complex Multiplication
  - Complex-by-Real Multiplication
- Filtering Functions
  - High Precision Q31 Biquad Cascade Filter
  - Biquad Cascade IIR Filters Using Direct Form I Structure
  - Biquad Cascade IIR Filters Using a Direct Form II Transposed Structure
  - Convolution
  - Partial Convolution
  - Correlation
  - Finite Impulse Response (FIR) Decimator
  - Finite Impulse Response (FIR) Filters
  - Finite Impulse Response (FIR) Lattice Filters
  - Finite Impulse Response (FIR) Sparse Filters
  - Infinite Impulse Response (IIR) Lattice Filters
  - Least Mean Square (LMS) Filters
  - Normalized LMS Filters
  - Finite Impulse Response (FIR) Interpolator
- Matrix Functions
  - Matrix Addition
  - Matrix Initialization
  - Matrix Inverse
  - Matrix Multiplication
  - Matrix Scale
  - Matrix Subtraction
  - Matrix Transpose
- Transform Functions
  - Radix-2 Complex FFT Functions
  - Radix-4 Complex FFT Functions
  - DCT Type IV Functions
  - Real FFT Functions
  - Complex FFT Tables
- Controller Functions
  - Sine Cosine
  - PID Motor Control
  - Vector Clarke Transform
  - Vector Inverse Clarke Transform
  - Vector Park Transform
  - Vector Inverse Park transform
- Statistics Functions
  - Maximum
  - Mean
  - Minimum
  - Power
  - Root mean square (RMS)
  - Standard deviation
  - Variance
- Support Functions
  - Vector Copy
  - Vector Fill
  - Convert 32-bit floating point value
  - Convert 16-bit Integer value
  - Convert 32-bit Integer value
  - Convert 8-bit Integer value
- Interpolation Functions
  - Linear Interpolation
  - Bilinear Interpolation
- Examples
  - Class Marks Example
  - Convolution Example
  - Dot Product Example
  - Frequency Bin Example
  - FIR Lowpass Filter Example
  - Graphic Audio Equalizer Example
  - Linear Interpolate Example
  - Matrix Example
  - Signal Convergence Example
  - SineCosine Example
  - Variance Example

http://www.disca.upv.es/aperles/arm_cortex_m3/curset/CMSIS/Documentation/DSP/html/modules.html

Using the Library

The library installer contains prebuilt versions of the libraries in the Lib folder.

arm_cortexM4lf_math.lib (Little endian and Floating Point Unit on Cortex-M4)
arm_cortexM4bf_math.lib (Big endian and Floating Point Unit on Cortex-M4)
arm_cortexM4l_math.lib (Little endian on Cortex-M4)
arm_cortexM4b_math.lib (Big endian on Cortex-M4)
arm_cortexM3l_math.lib (Little endian on Cortex-M3)
arm_cortexM3b_math.lib (Big endian on Cortex-M3)
arm_cortexM0l_math.lib (Little endian on Cortex-M0)
arm_cortexM0b_math.lib (Big endian on Cortex-M3)

- The library functions are declared in the public file **arm_math.h** which is placed in the Include folder.
  http://www.disca.upv.es/aperles/arm_cortex_m3/curset/CMSIS/Documentation/DSP/html/arm__math_8h.html
- Same header file will be used for floating point unit(FPU) variants.
- Define the appropriate pre processor MACRO ARM_MATH_CM4 or ARM_MATH_CM3 or ARM_MATH_CM0 depending on the target processor in the application.

Steps of Adding

- 1.We need to add two files to our project in STMCubeIDE for calling the CMSIS functions:

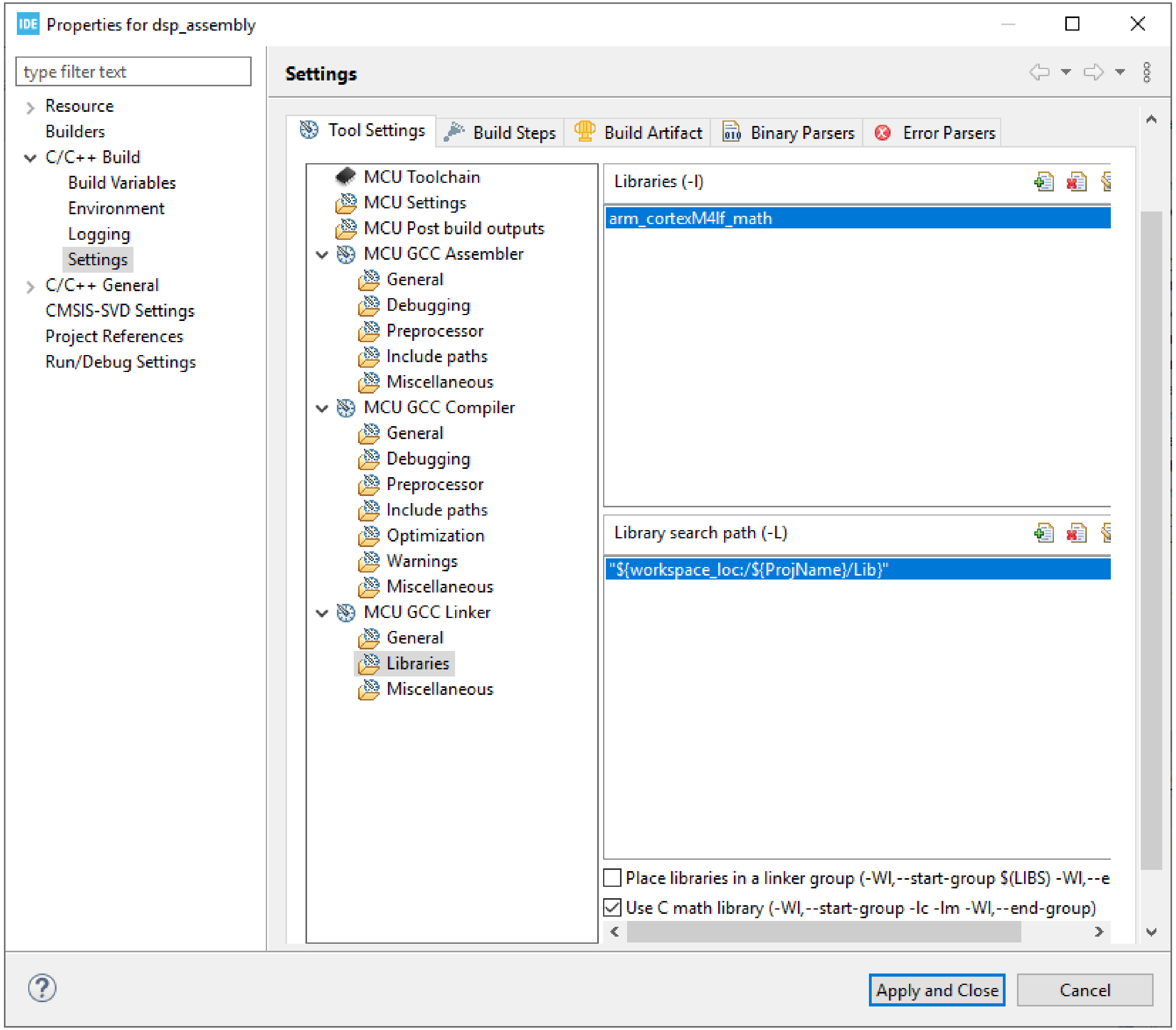
  C:\Users\<user_name>\STM32Cube\Repository\STM32Cube_FW_L4_V1.17.1\Drivers\CMSIS\DSP\Include\arm_math.h

  C:\Users\<user_name>\STM32Cube\Repository\STM32Cube_FW_L4_V1.17.1\Drivers\CMSIS\DSP\Lib\GCC\libarm_cortexM4lf_math.a

- 2. Copy “arm_math.h” to the “Inc” folder of project.
- 3. Create “lib” folder.
- 4. Add “libarm_cortexM4lf_math.a” to “lib” folder.
- 5. Link the header to prebuilt binary file. (next slide)

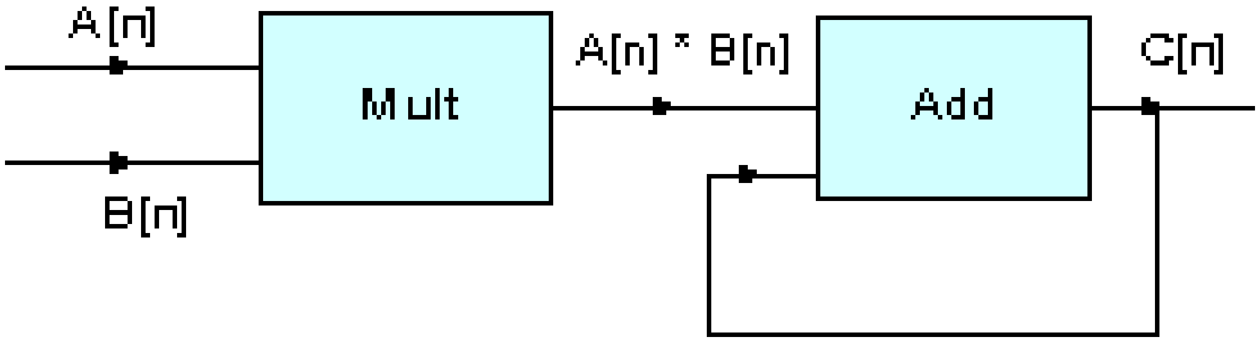
- Add these line to “main.c”

```
#define ARM_MATH_CM4
#include “arm_math.h”
```


Properties for dsp_assembly
type filter text
Resource
Builders
C/C++ Build
Build Variables
Environment
Logging
Settings
C/C++ General
CMSIS-SVD Settings
Project References
Run/Debug Settings
Settings
Tool Settings
Build Steps
Build Artifact
Binary Parsers
Error Parsers
MCU Toolchain
MCU Settings
MCU Post build outputs
MCU GCC Assembler
General
Debugging
Preprocessor
Include paths
Miscellaneous
MCU GCC Compiler
General
Debugging
Preprocessor
Include paths
Optimization
Warnings
Miscellaneous
MCU GCC Linker
General
Libraries
Miscellaneous
Libraries (-l)
arm_cortexM4lf_math
Library search path (-L)
"${workspace_loc:/${ProjName}/Lib}"
Place libraries in a linker group (-Wl,--start-group $(LIBS) -Wl,--e
Use C math library (-Wl,--start-group -lc -lm -Wl,--end-group)
Apply and Close
Cancel

dotProduct

dotProduct = A[0] * B[0] + A[1] * B[1] + ... + A[n-1] * B[n-1]



```c
#include <math.h>
#include "arm_math.h"

/* ----------------------------------------------------------------------
* Defines each of the tests performed
* ------------------------------------------------------------------- */
#define MAX_BLOCKSIZE   32
#define DELTA           (0.000001f)

/* ----------------------------------------------------------------------
* Test input data for Floating point Dot Product example for 32-blockSize
* Generated by the MATLAB randn() function
* ------------------------------------------------------------------- */
/* ----------------------------------------------------------------------
** Test input data of srcA for blockSize 32
** ------------------------------------------------------------------- */
float32_t srcA_buf_f32[MAX_BLOCKSIZE] =
{
-0.4325648115282207,   -1.6655843782380970,   0.1253323064748307,
 0.2876764203585489,   -1.1464713506814637,   1.1909154656429988,
 1.1891642016521031,   -0.0376332765933176,   0.3272923614086541,
 0.1746391428209245,   -0.1867085776814394,   0.7257905482933027,
-0.5883165430141887,    2.1831858181971011,  -0.1363958830865957,
 0.1139313135208096,    1.0667682113591888,   0.0592814605236053,
-0.0956484054836690,   -0.8323494636500225,   0.2944108163926404,
-1.3361818579378040,    0.7143245518189522,   1.6235620644462707,
-0.6917757017022868,    0.8579966728282626,   1.2540014216025324,
-1.5937295764474768,   -1.4409644319010200,   0.5711476236581780,
-0.3998855777153632,    0.6899973754643451
};
```

```c
/* ----------------------------------------------------------------------
** Test input data of srcB for blockSize 32
** ------------------------------------------------------------------- */
float32_t srcB_buf_f32[MAX_BLOCKSIZE] =
{
 1.7491401329284098,    0.1325982188803279,    0.3252281811989881,
-0.7938091410349637,    0.3149236145048914,   -0.5272704888029532,
 0.9322666565031119,    1.1646643544607362,   -2.0456694357357357,
-0.6443728590041911,    1.7410657940825480,    0.4867684246821860,
 1.0488288293660140,    1.4885752747099299,    1.2705014969484090,
-1.8561241921210170,    2.1343209047321410,  1.4358467535865909,
-0.9173023332875400,   -1.1060770780029008,    0.8105708062681296,
 0.6985430696369063,   -0.4015827425012831,    1.2687512030669628,
-0.7836083053674872,    0.2132664971465569,    0.7878984786088954,
 0.8966819356782295,   -0.1869172943544062,    1.0131816724341454,
 0.2484350696132857,    0.0596083377937976
};

/* Reference dot product output */
float32_t  refDotProdOut = 5.9273644806352142;

/* ----------------------------------------------------------------------
* Declare Global variables
* ------------------------------------------------------------------- */
float32_t multOutput[MAX_BLOCKSIZE];  /* Intermediate output */
float32_t testOutput;  /* Final ouput */

arm_status status;       /* Status of the example */
```

```c
int32_t main(void)
{
	uint32_t i;			/* Loop counter */
	float32_t diff;		/* Difference between reference and test outputs */

	/* Multiplication of two input buffers */
	arm_mult_f32(srcA_buf_f32, srcB_buf_f32, multOutput, MAX_BLOCKSIZE);

	/* Accumulate the multiplication output values to
	   get the dot product of the two inputs */
	for(i=0; i< MAX_BLOCKSIZE; i++)
    {
		arm_add_f32(&testOutput, &multOutput[i], &testOutput, 1);
    }

	/* absolute value of difference between ref and test */
	diff = fabsf(refDotProdOut - testOutput);

}
```

```c
/* Comparison of dot product value with reference */
	if(diff > DELTA)
	{
		status = ARM_MATH_TEST_FAILURE;
	}

	if( status == ARM_MATH_TEST_FAILURE)
	{
	  while(1);
	}

  while(1);				/* main function does not return */
```

Agenda:

1. CMSIS-DSP Library
2. Add assembly code
3. Single Wire Output (SWO) interface
4. Lab 1 Introduction

Cortex-M4 Proc Tech Ref Manual (2.1,7)

pm0214-stm32-cortexm4-mcus-and-mpus-programming-manual-stmicroelectronics (3.10)

The GNU Assembler (from stmCubeIDE)

Thumb®-2 Instruction Set

Vector Floating Point Instruction Set

Add *assembly* code to your project:

1. Create assembly file : <assembly_codeName>.s

   Ex. : func.s

2. (Optional : coding style)Add the function prototype to <your header file>.h OR put into <main.h>

```
extern void <function name>(arguments);//comments
```

```
Ex. extern void function(float *array, uint32_t size);
```

3. Define your function by assembly

   next slide

4. Call your function in your "main.c" :

```
Ex. func(&array, 8);
```

Assembler Directives:

as.pdf : // you can access through stmCubeIDE

```
/*
* func.s
*/
//.section .data

.syntax unified //as.pdf : p141
.align  16 //as.pdf :p71
.section .text, "x" //as.pdf :p96
//.rodata
.global func //as.pdf : p254
/**
Comments for code readability
*/
//Your assembly code
func: //label

//PUSH current state to stack
//setup registers
//your function
//POP stack



BX LR //end & return
```

Document your assembly code.
For demo and report.

C code: if (R0>R1) then R2=R0

```
CMP R0 R1// CMP compares two numeric data fields
BLE  else //Branch on Less than or Equal
MOV R2 R0   //set value
MOV R2, R0  //set value
B endif  //branch
else: MOV R2, R1
endif:
```

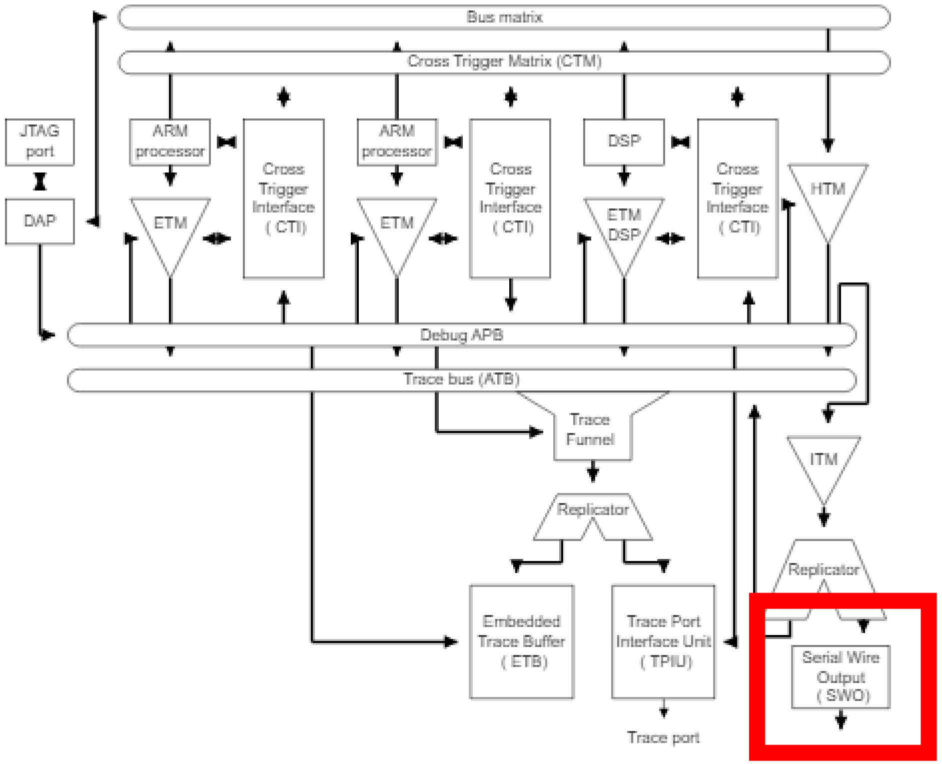
McGill

[Procedure Call Standard for the Arm® Architecture]

Agenda:

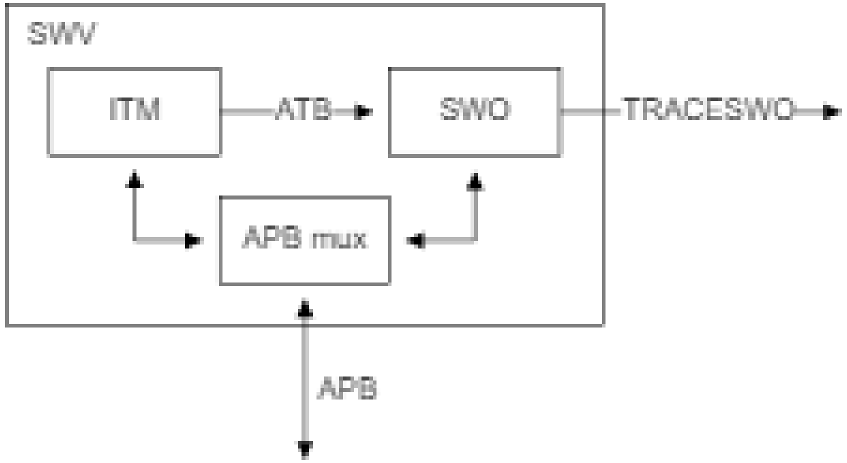
1. CMSIS-DSP Library
2. Add assembly code
3. Single Wire Output (SWO) interface
4. Lab 1 Introduction

Typical CoreSight Design Kit debugging environment



***Instrumentation Trace Macrocell* (ITM) and *Serial Wire Output* (SWO)**

As a Serial Wire Viewer(SWV)



https://developer.arm.com/documentation/ddi0314/h/Serial-Wire-Viewer/About-the-Serial-Wire-Viewer?lang=en

Single Wire Output (SWO) Debugging interface

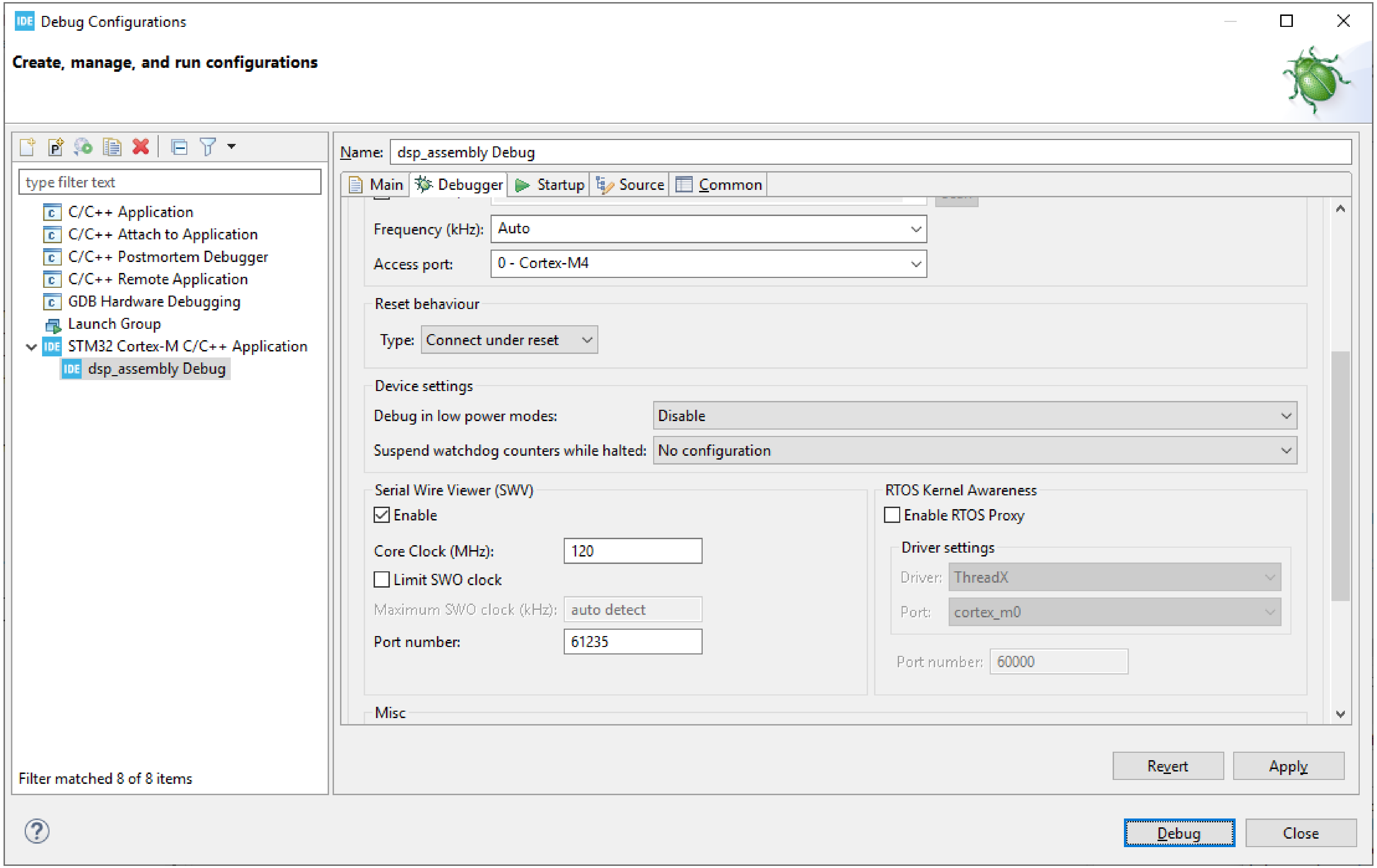
- **STMCubeMX**

  Set PB3 to SYS_JTDO-SWO, and
  Set PA13 to JTMS-SWDIO.

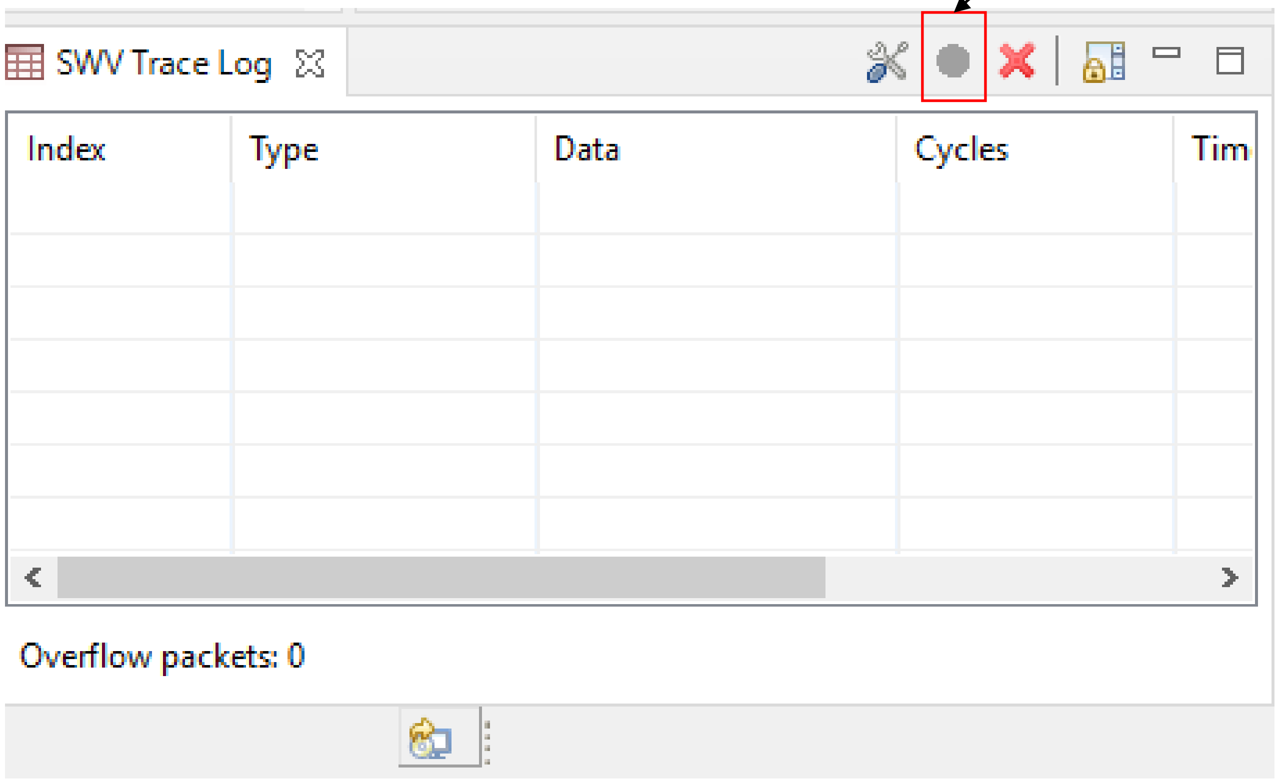
- **STMCubeIDE**

  RUN menu --> Debug Configurations :
  Under ***Serial Wire Viewer (SWV)*****,** tick the "***Enable***" box, and set the Core Clock to 120.0 MHz. Apply the changes and "**Debug.**"

  In the debugging mode Window menu --> Show View --> Other --> *SWV --> SWV Trace Log*


Debug Configurations
Create, manage, and run configurations
type filter text
C/C++ Application
C/C++ Attach to Application
C/C++ Postmortem Debugger
C/C++ Remote Application
GDB Hardware Debugging
Launch Group
STM32 Cortex-M C/C++ Application
dsp_assembly Debug
Filter matched 8 of 8 items
Name: dsp_assembly Debug
Main
Debugger
Startup
Source
Common
Frequency (kHz): Auto
Access port: 0 - Cortex-M4
Reset behaviour
Type: Connect under reset
Device settings
Debug in low power modes: Disable
Suspend watchdog counters while halted: No configuration
Serial Wire Viewer (SWV)
Enable
Core Clock (MHz): 120
Limit SWO clock
Maximum SWO clock (kHz): auto detect
Port number: 61235
RTOS Kernel Awareness
Enable RTOS Proxy
Driver settings
Driver: ThreadX
Port: cortex_m0
Port number: 60000
Misc
Revert
Apply
Debug
Close

After running in debug mode it will be activated.



- **Add these codes to your code for SVW debugging:**

```
/* Private define
------------------------------------------------------------*/
/* USER CODE BEGIN PD */
#define ITM_Port32(n) (*((volatile unsigned long *) (0xE0000000+4*n)))
/* USER CODE END PD */
```

`ITM_Port32(n)` is a location in memory; setting it to a value will generate a trace packet with that value as the data.

```
/* USER CODE BEGIN 3 */
ITM_Port32(31) = 1;
//put your code in here for monitoring execution time
ITM_Port32(31) = 2;
}
/* USER CODE END 3 */
```

McGill

Agenda:

1. CMSIS-DSP Library
2. Add assembly code
3. Single Wire Output (SWO) interface
4. Lab 1 Introduction

Purpose of the Lab : Comparing the performance of a function which implemented in three different way.

Filter C code

C code(you write)
Compiler
Assembler,
Binary

Filter Assembly code

Your Assembly code(you write)
Assembler
Binary

Filter C code by CMSIS-DSP

C code by DSP Lib(you write)
Compiler
Assembler
Binary

References:

1.https://developer.arm.com/tools-and software/embedded/cmsis

2.https://arm-software.github.io/CMSIS_5/General/html/index.html

3.http://www.disca.upv.es/aperles/arm_cortex_m3/curset/CMSIS/Documentation/General/html/index.html


GOODBYE
Image source : www.mtlblog.com/