LAB-5 REPORT

Subject: **Embedded Hardware Design**

Subject Code: **EL203**

Members: 1) Shantanu Tyagi **(201801015)**
2) Nikhil Mehta **(201801030)**
3) Sudhanshu Mishra **(201801114)**
4) Bhavana Kolli **(201701082)**

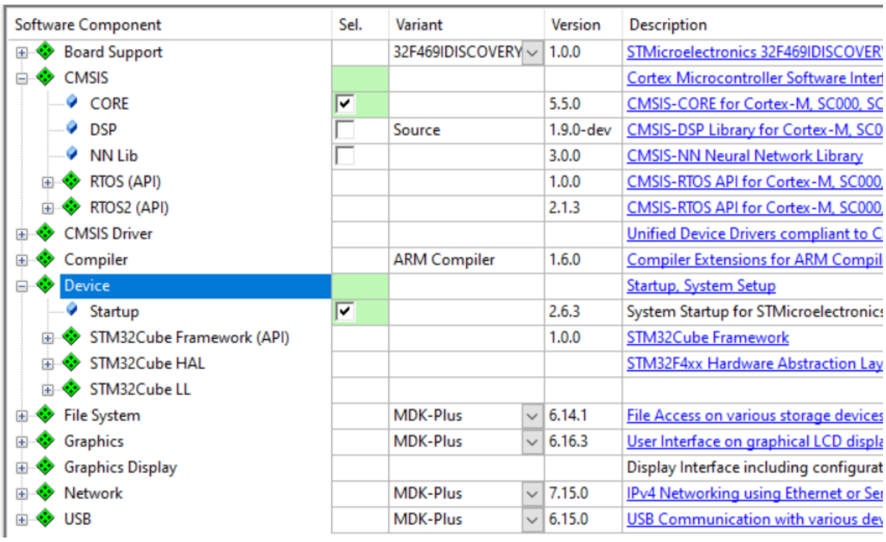
Environment Setup

In the IDE, install the Keil::STM32F4xx DFP Package from Pack Manager.

1) Create a new Project and rename it using the appropriate nomenclature.

1.1.1.1. Set STM32F407VGTx as the target device under the device header

1.1.1.2. Modify the options for the target device:

2. 

Software Component	Sel.	Variant	Version	Description
Board Support		32F469IDISCOVERY	1.0.0	STMicroelectronics 32F469IDISCOVER
CMSIS				Cortex Microcontroller Software Interf
CORE	[x]		5.5.0	CMSIS-CORE for Cortex-M, SC000, SC
DSP	[ ]	Source	1.9.0-dev	CMSIS-DSP Library for Cortex-M, SC0
NN Lib	[ ]		3.0.0	CMSIS-NN Neural Network Library
RTOS (API)			1.0.0	CMSIS-RTOS API for Cortex-M, SC000,
RTOS2 (API)			2.1.3	CMSIS-RTOS API for Cortex-M, SC000,
CMSIS Driver				Unified Device Drivers compliant to C
Compiler		ARM Compiler	1.6.0	Compiler Extensions for ARM Compil
Device				Startup, System Setup
Startup	[x]		2.6.3	System Startup for STMicroelectronics
STM32Cube Framework (API)			1.0.0	STM32Cube Framework
STM32Cube HAL				STM32F4xx Hardware Abstraction Lay
STM32Cube LL				
File System		MDK-Plus	6.14.1	File Access on various storage devices
Graphics		MDK-Plus	6.16.3	User Interface on graphical LCD displa
Graphics Display				Display Interface including configurat
Network		MDK-Plus	7.15.0	IPv4 Networking using Ethernet or Ser
USB		MDK-Plus	6.15.0	USB Communication with various dev

2.1. Create a new group under the project

2.2. Make modifications for the following additional options for the target device:

2.2.1. Set ARM compiler version 6 under the Target -> Code Generation header

2.2.2. Select the Simulator radio button and load the KEIL_STM.ini file as an initialization file under the Debug header

2.3. Add the main.c and sinewave.c file under the created group in the project with the relevant header file (#include "stm32f4xx_hal.h",#include "arm_math.h")

Running the Simulation

1 To run the code, first, we have to compile the main.c file. This is done by clicking on the build button in Keil IDE.

2 Once the build is complete, we are prompted with errors and we need to fix them.

3 After fixing the bugs we move to the debug section and run the code to get the results.

4 Now, if we need to make any changes to the code, we must first halt the debug session, then make the required modifications, recompile our code, and run it as described before.

Code Part 1:

```
//main.c
int Number(void);
int value;
int main() {
    while(1){
        value = Number();
    }
}
```

```
// function.s
      AREA |.text|, CODE, READONLY
      EXPORT Number
Number
      MOV R0, #50
      BX LR
      ALIGN
      END
```

Output:

Name	Value	Type
\\lab2\main.c\inputSample	<cannot evaluate>	uchar
value	50	int
R0	50	ulong
\\lab2\main.c\inputSample	<cannot evaluate>	uchar
<Enter expression>		

Code Part 2:

```
// function.c

int num;
int Adder(void)
{
      num = num + 50;
      return(num);
}
```

```
// main.s
; IMPORT C TO ASM

                  AREA | .text|, CODE, READONLY
                  IMPORT num
                  IMPORT Adder
                  EXPORT __main
                  ENTRY

__main
                  LDR R1, = num
                  MOV R0, #50
                  STR R0, [R1]
                  BL Adder ; RO = RO+R1

                  END
```

Output:

Name	Value	Type
num	100	int
R0	100	ulong
R1	50	ulong
<Enter expression>		

Code Part 3:

```
// main.c
__inline int getSum(int a, int b)
{
            int sum;
            __asm{

            ADD sum, a, b;
            }
            return sum;
}

int z;
int main(void)
{
      int x,y;
      x =50;
      y = 50;
      while(1)
```

```
    {
            // use add from instruction set
             z = getSum(x,y);
    }

}
```

Output:

Name	Value	Type
freq	<cannot evaluate>	uchar
count	<cannot evaluate>	uchar
z	100	int
<Enter expression>		

Conclusion/Observations:

1. In the first section, we imported Assembly code into C code. The file function.s (assembly-level code) returns a number, which is then read and examined by the C level code. The screenshots above can be used to compile these findings. The argument/input, in this case, is 50, thus the output is also 50.
2. In Part 2, we'll learn how to convert C code to assembly code. In C, we create an adder function that multiplies the input integer by 50 and returns it. The data is subsequently imported into assembly code and placed in the appropriate registers. The argument/input is 50 in this case, thus the total is 100.
3. In Part 3, we write inline assembly code in a .c file to blend Assembly level code with C level code. In this example, we build an adder function in assembly language and return the total of the parameters to the C driver code, which is then saved in a variable and examined. Both arguments/inputs are 50 in this case, thus the total is 100.