LAB-7 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)**

Problem Statement

In this lab, we'll use Assembly level code to calculate the reciprocal of a number's square root.

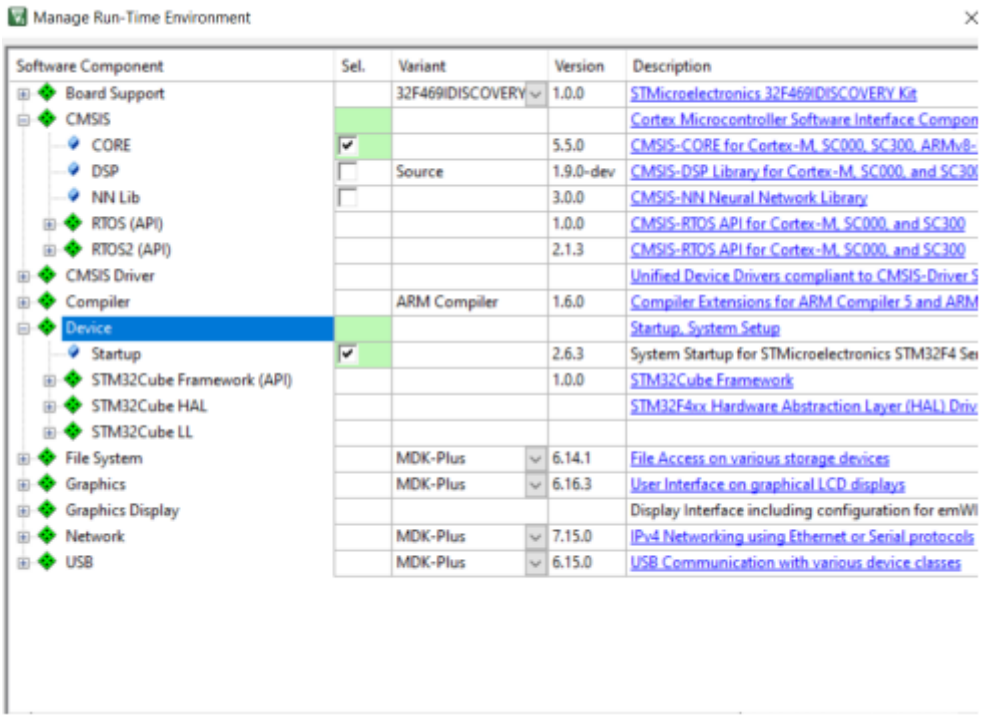
Environment Setup

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:



1.2. Create a new group under the project

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

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

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

1.4. 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 We must compile the files in the project folder in order to run the code (.s files). The build button in Keil may be used to do this.
2 After building the codes, it shows the errors that the user has made. The user needs to fix them first.
3 Start the debug session when you've fixed all of the issues and then run the code to observe the results.
4 If we need to make any changes to the code now, we must first pause the debugging session, then make the necessary changes, recompile our code, and execute it as instructed previously.

CODE:

```
; Code for reciprocal of square root
AREA RECIPROCAL_SQRT, CODE,READONLY
EXPORT __main

__main

    LDR R0, =0XE000ED88; READ VALUE AT COPROCESSOR ADDRESS
    LDR R1, [R0]
    ORR R1, R1, #(0XF<<20)
    STR R1, [R0]

    ADR R0, RECIPROCAL_SQRT_ODD_TABLE ; ADDRESS TO ODD TABLE
    ADR R1, RECIPROCAL_SQRT_EVEN_TABLE ; ADDRESS TO EVEN TABLE

    VLDR.F S0, inputValue
    VMOV.F R2, S0

    ;(1) PROCESS THE EXPO FIRST - FOR POSITIVE INPUT
    MOV R3, R2 ; EXPONENT IN R2, FRACTION IN R3
    LSR R2, #23 ; SHIFT EXPONENT FOR SUBTRACTION
    SUB R2, #127 ; SUBTRACT OUT THE BIAS
    AND R4, R2, #1 ; CAPTURE LSB TO R4
    TEQ R4, #1 ; CHECK FOR EXPONENT
    ;(2) ODD EXPO - ADD 1 BEFORE THE NEGATE AND SHIFT RIGHT
OPERATIONS
    ADDEQ R2, #1
```

```
    ;(3) ALL EXPONENTS
    LSR R2, R2, #1 ; SHIFT RIGHT BY 1 TO DIVIDE BY 2
    NEG R2, R2 ; NEGATE
    ADD R2, #127 ;
    LSL R2, #23 ;
    AND R3, 0x00780000
    LSR R3, #18 ; SHIFT SO THEY ARE *2 ;
    LDRHEQ R4, [R3, R0] ;
    LDRHNE R4, [R3, R1] ;
    VMOV.F S3, R4 ;
    VCVTB.F32.F16 S4, S3 ;
    VMOV.F S5, R2 ;
    VMUL.F S6, S5, S4 ;

    ;(4) EXTRACT UPPER 4 FRACTION BITS FOR TABLE LOOKUP
    AND R3, 0x00780000
    LSR R3, #18 ; SHIFT SO THEY ARE *2

    ;(5) SELECT TABLE AND TABLE ENTRY BASED ON THE UPPER FRACTION
BITS
    LDRHEQ R4, [R3, R0] ; INDEX INTO ODD TABLE
    LDRHNE R4, [R3, R1] ; INDEX INTO EVEN TABLE
    VMOV.F S3, R4 ; COPY THE SELECTED HALF PRECISION
    VCVTB.F32.F16 S4, S3 ; CONVERT THE ESTIMATE TO SP
    VMOV.F S5, R2 ; MOVE THE EXP MULTIPLIER TO S5
    VMUL.F S6, S5, S4 ; COMPUTE THE RECIPROCAL ESTIMATE

LOOP B LOOP
    ALIGN

inputValue
    DCD 0x42333333 ;
    ALIGN

RECIPROCAL_SQRT_ODD_TABLE
    DCW 0x3DA8 ; 0.5000 -> 1.4142
    DCW 0x3D7C ; 0.5322 -> 1.3707
    DCW 0x3D55 ; 0.5625 -> 1.3333
    DCW 0x3D31 ; 0.5938 -> 1.2978
    DCW 0x3D0F ; 0.6250 -> 1.2649
    DCW 0x3CF0 ; 0.6563 -> 1.2344
    DCW 0x3CD3 ; 0.6875 -> 1.2060
```

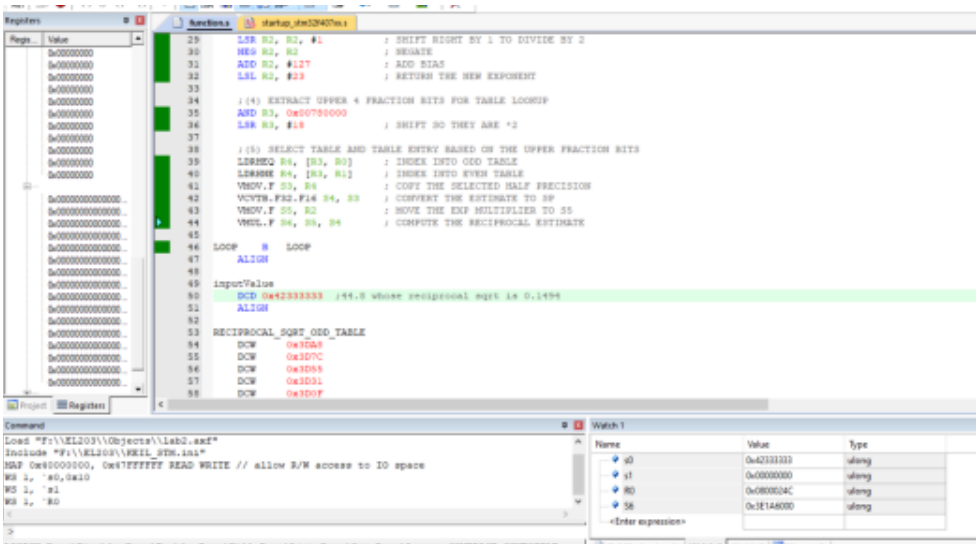
```
    DCW 0x3CB8 ; 0.7186 -> 1.1795
    DCW 0x3C9E ; 0.7500 -> 1.1547
    DCW 0x3C87 ; 0.7813 -> 1.1313
    DCW 0x3C70 ; 0.8125 -> 1.1094
    DCW 0x3C5B ; 0.8438 -> 1.0886
    DCW 0x3C47 ; 0.875
    ALIGN

RECIPROCAL_SQRT_EVEN_TABLE
    DCW 0x3C00 ; 1.0000 -> 1.0000
    DCW 0x3BC3 ; 1.0625 -> 0.9701
    DCW 0x3B8B ; 1.1250 -> 0.9428
    DCW 0x3A57 ; 1.1875 -> 0.9177
    DCW 0x3B28 ; 1.2500 -> 0.8944
    DCW 0x3AFC ; 1.3125 -> 0.8729
    DCW 0x3AD3 ; 1.3750 -> 0.8528
    DCW 0x3AAC ; 1.4375 -> 0.8340
    DCW 0x3A88 ; 1.5000 -> 0.8165
    DCW 0x3A66 ; 1.5625 -> 0.8000
    DCW 0x3A47 ; 1.6250 -> 0.7845
    DCW 0x3A29 ; 1.6875 -> 0.7698
    DCW 0x3A0C ; 1.7500 -> 0.7559
    DCW 0x39F1 ; 1.8125 -> 0.7428
    DCW 0x39D8 ; 1.8750 -> 0.7303
    DCW 0x39BF ; 1.9375 -> 0.7184
    ALIGN
END
```

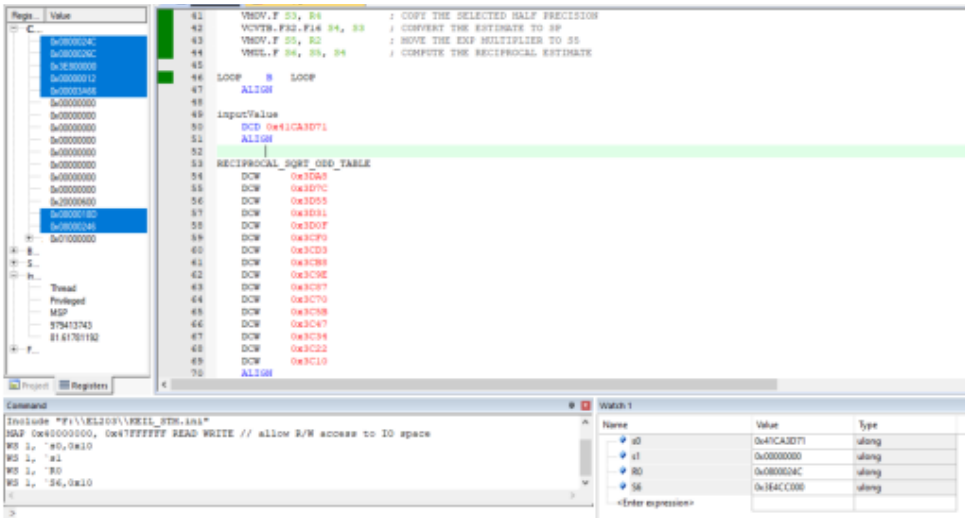
Simulation

Here we shall consider various cases of input (as the initial value of x) and later store the actual value in S6. This is clearly highlighted by the values present on the left portion of the Screenshots (FPU Section) as well as the watch values.

Input 1: The input here is 44.8



Input 2: The input here is 0.1507568359375.



Conclusions/Observations:

- We'll use 44.8 as the input in the first scenario. The predicted output is 0.1494, while the simulated observation is 0.1507568359375. The solution in this example can be validated because the two values are so similar.

- In the second scenario, we'll use 25.28 as the input. The simulated observation is 0.199951171875, while the predicted output is 0.19889. Because the two values are so comparable, the solution in this example may be validated.