LABORATORY MANUAL

Microcontroller& Programming Lab Manual
(21EI43)
SEMESTER IV
(Autonomous Scheme2021)

Student Name:	
USN:	

DEPARTMENT OF ELECTRICAL & ELECTRONICS ENGINEERING

RV College of Engineering®,
(Autonomous Institution affiliated to VTU, Belgaum)

Bangalore - 560059

RV College of Engineering®, Bangalore - 59

(Autonomous Institution affiliated to VTU, Belgaum)

Department of Electrical & Electronics Engineering

<u>Laboratory Certificate</u>

This is to certify that Mr. / Ms ______________________________ ________________________has satisfactorily completed the course of Experiments in Practical_________________________prescribed by the Department during the year _________________

Name of the Candidate: _________________________________

USN No.:____________________Semester:_________________

Marks	
Max. Marks	Obtained
50	

Marks in Words	

Signature of the staff in-charge

Head of the Department

Date:

RV College of Engineering®, Bangalore - 59

(Autonomous Institution affiliated to VTU, Belgaum)

Department of Electronics and Communication Engineering

SCHEME OF CONDUCTION AND EVALUATION

CLASS: IV SEMESTER CIE MARKS: (Max.):50
YEAR: 2023 SEE MARKS: (Max): 50

Sl. No.	Date	EXPERIMENTS	Page No.	Marks Obtained (Max Marks:10)	Signature of Staff
1					
2					
3					
4					
5					
6					
7					
8					
9					
10					
Total					
TEST					

SEE: 03 Hrs

RV College of Engineering®, Bangalore - 59

(Autonomous Institution affiliated to VTU, Belgaum)

Department of Electronics and Communication Engineering

LIST OF EXPERIMENTS

Part I: Programming in ARM Assembly using Keil

1. Data Transfer Programs: Block Moves & Exchange (With & Without Overlap) with &without String Instructions.
2. Arithmetic Operations: Addition, Multiplication & Division on 32-Bit Data.
3. Search for a Key in an Array of Elements using Linear Search, Binary Search.

Part-II: Programming in Keil using embedded C in STMCubeMx

4. Program digital IOs control LEDs, seven segment interface, push buttons.
5. Program digital IOs to control stepper and motor drivers for given specifications.
6. Program ADC and show analog to digital conversion. Display digital value on suitable interface.
7. Program ADC and show interfacing of analog sensor for given specifications.
8. Program USART and serial data transfer.
9. Program SPI and show the configuration and data transfer between SPI slave device and master.
10. Program to configure NVIC and writing interrupt service routines.

Part III: Innovative Experiments

1. Program SPI and show the configuration and data transfer between SPI slave device and master.
2. Program ADC and show interfacing of analog sensor for given specifications.
3. Data transfer in polling, interrupt and DMA based modes.
4. Real time Audio applications: Flanging effect.

Introduction to Keil Embedded Development Tools

Keil software embedded development tools provide a powerful development environment for rapidly creating and testing embedded systems software. Integrated editor, compiler, assembler, and debugger enable you to build sophisticated, complex applications quickly and easily.The Keil µVision development supports three major microcontroller architectures which allow us to develop a wide range of applications.

- 8-bit 8x51 family of controllers
- 16-bit Infineon C16x/XC16x and ST10/Super 10
- 32-bit ARM based devices

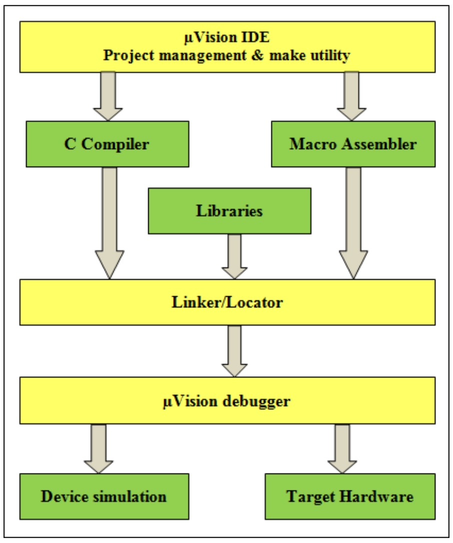
The supported controllers can be seen in target devices list. We are using AT89C51ED2 microcontroller by Atmel, the compiler tools are installed only for 8051 variants. Hence other devices are not seen.

Software Development Cycle

When you use the Keil µVision, the project development cycle is roughly the same as it is for any other software development project.

1. Create a project, select the target chip from the device database, and configure the tool settings.
2. Create source files in C or assembly.
3. Build your application with the project manager.
4. Correct errors in source files.
5. Test the linked application.

The following block diagram illustrates the complete µVision software development cycle. Each component is described below.



µVision IDE

The µVision IDE combines project management, a rich-featured editor with interactive error correction, option setup, make facility, and on-line help. Use µVision to create your source files and organize them into a project that defines your target application. µVision automatically compiles, assembles, and links your embedded application.

DATA TRANSFER PROGRAMS

Experiment 01

a) Write Thumb-2 Assembly Language Program to transfer block of data from ROM memory to RAM and show data exchange operation.

Algorithm:

1. Load the data on the ROM.
2. Load the length of both arrays together in register 1.
3. Load the address of the last element of the ROM array 2 to register 2.
4. Load the address of the last element of the RAM array 2 to register 3.
5. Move the contents present at the address pointed by register 2 to the address pointed by register 3 and decrement the length in register 1 to point to the previous address.
6. Loop till length is 0.
7. Load the length of a single array in register 1.
8. Load the address of the last element of RAM array 1 to register 2.
9. Load the address of the last element of RAM array 2 to register 3.
10. Load the data pointed by both registers 2 and 3 to registers 3 and 4 respectively.
11. Store the data from registers 3 and 4 to the address in registers 3 and 2 respectively.
12. Decrement the length in register 1 to point to the previous address.
13. Loop till length is 0.

Program:

```
        PRESERVE8                                  ; Used to specify 8-byte alignment of stack

Stack   EQU 0x00000100                             ;Define stack size

        AREA STACK, NOINIT, READWRITE,ALIGN=3 ; AREA directive is used to define a code or
                                           ;data segment in arm assembly program
StackMem SPACE Stack                       ; STACK directive is used to define stack segment
                                              ; SPACE directive is used to define
;specify that from label "StackMem" to next 256 byte have to be 0 initialized


        AREA RESET, DATA , READONLY                ; This is a data segment which defines vector
                                                   ;table which points to different interrupt handlers
        EXPORT__Vectors                            ; Export is used to specify that the block of code
                                                   ;following the label is global label

__Vectors
        DCD StackMem + Stack                       ; Starts with address of stack pointer
        DCD Reset_Handler


        AREA tempData, DATA, READONLY, ALIGN=4
ROMARRAY1   DCD         0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07, 0x08
                                                   ; Initializing Array1 in the ROM
ROMARRAY2   DCD         0x09, 0x10, 0x11, 0x12, 0x13, 0x14, 0x15, 0x16
                                                   ; Initializing Array2 in the ROM

        AREA   |.data|, DATA, READWRITE, ALIGN=4
LEN EQU             0x08                           ; Length of Array,8 words
ARRAY1      SPACE 32                               ; 32 bytes allocated for Array 1
```

```
ARRAY2          SPACE 32                                  ; 32 bytes allocated for Array 2

        AREA |.text|, CODE , READONLY , ALIGN=4
        ENTRY
        EXPORT Reset_Handler                               ; Defining reset handler
Reset_Handler                                             ; Reset handler starts
Reset_Handler_End                                         ; Reset handler ends

        MOV     R0, #0x10                                 ; HEX(16) = Length of 2 Arrays
        LDR     R1, =ROMARRAY2 + 28
                                                          ; Point to last address of Array 2 on ROM 28 = (LEN-1) * 4
        LDR      R2, =ARRAY2 + 28                         ; Point to last address of Array 2 on RAM28 = (LEN-1) * 4

MEMCPY_LOOP
        LDR     R3, [R1],#-4                              ; Load data from address pointed by R1, and
                                                          ;decrement the address in R1 by 0x04
        STR     R3,[R2],#-4                               ; Load data from address pointed by R2, and
                                                          ;decrement the address in R2 by 0x04
        SUBS    R0, R0, #1                                ; Decrement the length
        BNE             MEMCPY_LOOP                       ; Loop till length is equal to 0

        MOV             R0, #LEN                          ; Store the length of single array to R0
        LDR             R1,=ARRAY1+28                     ; Point to last address of Array 1 on RAM     28 = (LEN-1) * 4
        LDR             R2,=ARRAY2+28                     ; Point to last address of Array 2 on RAM     28 = (LEN-1) * 4

EXCH_LOOP
        LDR             R3, [R1]
        LDR             R4, [R2]
        STR             R4, [R1], #-4
        STR             R3, [R2], #-4
        SUBS    R0, R0, #1
        BNE             EXCH_LOOP

STOP    B       STOP

        END
```

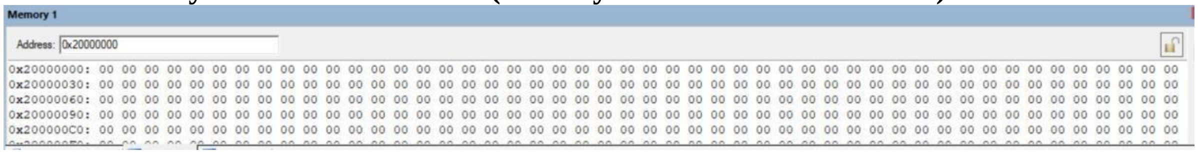
Procedure to edit and build in Keil IDE:

- Project -> New µVision Project. Give a suitable name to the project (No whitespaces) and Save.
- Under Select Device for Target window, STMicroelectronics -> STM32F4 Series ->STM32F407 -> STM32F407VG->STM32F407VGTx.
- Under the Software Component window, **only select OK**.
- Under the Project panel on the left-hand side, right-click Target 1 and select Options for Target ’Target 1’, go to Debug tab, and select the Use Simulator option.
- Expand Target 1 and right-click Source Group 1 -> Add New Item to Group ‘Source Group 1’, Select ASM file and give it a suitable name and select Add.
- Write the code in the created assembly file, save file.
- Go to project -> Build target.
- Go to Debug -> Start / Stop debug Session.
- Go to View -> Memory Windows -> Memory 1. Enter the address chosen for ROM.
- Go to View -> Memory Windows -> Memory 2. Enter the address chosen for RAM.
- Use the step option to execute one line at a time.
- Monitor the contents of the register and memory locations.

Output:

Before Execution:

RAM Memory window:0x20000000(Initially all contents will be zero)

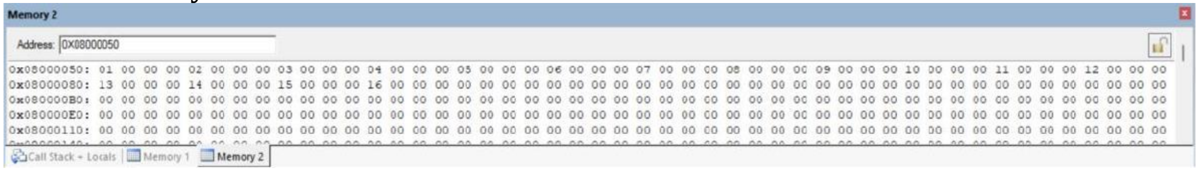


Memory 1

Address: 0x20000000

```
0x20000000: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x20000030: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x20000060: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x20000090: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x200000C0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

ROM Memory Window:0x08000050



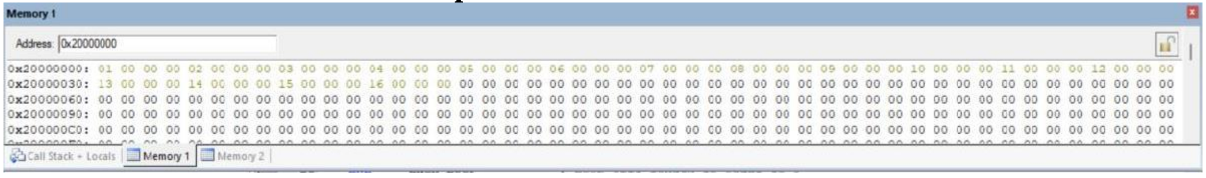
Memory 2

Address: 0X08000050

```
0x08000050: 01 00 00 00 02 00 00 00 03 00 00 00 04 00 00 00 05 00 00 00 06 00 00 00 07 00 00 00 08 00 00 00 09 00 00 00 10 00 00 00 11 00 00 00 12 00 00 00
0x08000080: 13 00 00 00 14 00 00 00 15 00 00 00 16 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x080000B0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x080000E0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x08000110: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

Call Stack + Locals | Memory 1 | Memory 2

After Execution: ROM data is copied to RAM



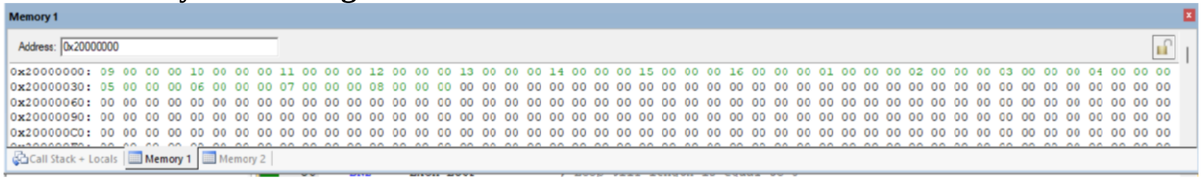
Memory 1

Address: 0x20000000

```
0x20000000: 01 00 00 00 02 00 00 00 03 00 00 00 04 00 00 00 05 00 00 00 06 00 00 00 07 00 00 00 08 00 00 00 09 00 00 00 10 00 00 00 11 00 00 00 12 00 00 00
0x20000030: 13 00 00 00 14 00 00 00 15 00 00 00 16 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x20000060: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x20000090: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x200000C0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

Call Stack + Locals | Memory 1 | Memory 2

Data in Arrays is exchanged:



Memory 1

Address: 0x20000000

```
0x20000000: 09 00 00 00 10 00 00 00 11 00 00 00 12 00 00 00 13 00 00 00 14 00 00 00 15 00 00 00 16 00 00 00 01 00 00 00 02 00 00 00 03 00 00 00 04 00 00 00
0x20000030: 05 00 00 00 06 00 00 00 07 00 00 00 08 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x20000060: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x20000090: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x200000C0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

Call Stack + Locals | Memory 1 | Memory 2

b) Write Thumb-2 ALP to transfer block of data from ROM memory to RAM and show data transfer operation with overlap in RAM memory.

The expected output of the program is as follows:

Before Execution: Assume an array of 8 words starting at an address 100h(RAM)

01	02	03	04	05	06	07	08	← Value
100	104	108	10C	110	114	118	11C	← Word

After Execution: Overlapped Array (Overlapped by 6)

01	02	**01**	**02**	**03**	**04**	**05**	**06**	07	08
100	104	108	10C	110	114	118	11C	120	124

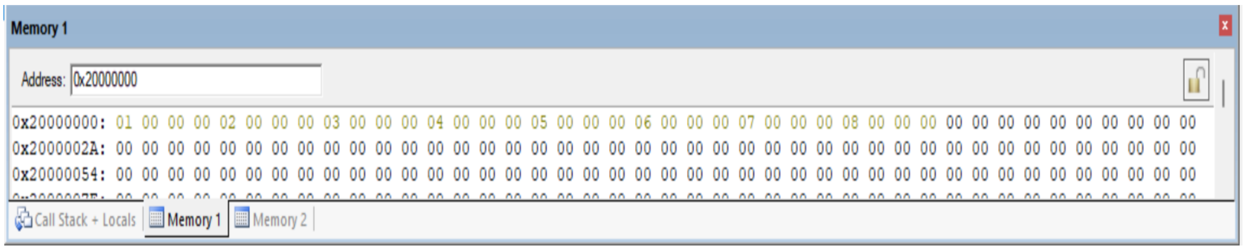
Algorithm:

1. Declare array in ROM an RAM memory.
2. Initialize pointers to arrays in RAM and ROM.
3. Write a loop with instructions to transfer data from ROM array to RAM array.
4. Initialize a pointer to last element of an array and another pointer to a location at an offset as indicated by non-overlap number of bytes.
5. Transfer the data from last elements until loop counter becomes zero.

Program:

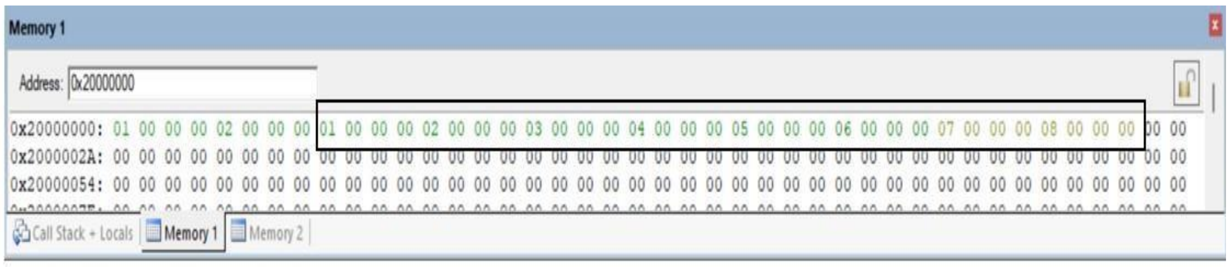
```
        PRESERVE8

Stack EQU 0x00000100

        AREA STACK, NOINIT, READWRITE,ALIGN=3
StackMem SPACE  Stack

        AREA RESET , DATA , READONLY
        EXPORT_Vectors

__Vectors
        DCD StackMem + Stack
        DCD Reset_Handler
        ;ALIGN
        AREA tempData, DATA, READONLY, ALIGN=3
ROMARRAY DCD        0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07,  0x08

        AREA data, DATA, READWRITE, ALIGN=3
LEN EQU         0x08
NOOVERLAP EQU  0x08                         ; Non overlapped offset bytes
ARRAY      SPACE        32

        AREA |.text| , CODE , READONLY , ALIGN=4
        ENTRY
        EXPORT Reset_Handler
Reset_Handler
Reset_Handler_End

        MOV R0, #0x08
        LDR         R1, =ROMARRAY + 28
        LDR         R2, =ARRAY + 28
MEMCPY_LOOP
        LDR    R3, [R1],       #-4
        STR    R3,[R2],        #-4
        SUBS R0, R0, #1
        BNE         MEMCPY_LOOP

        MOV         R0, #LEN
        LDR         R1,=ARRAY+28
        LDR         R2,=ARRAY+28+NOOVERLAP

TRANSFER_LOOP
        LDR         R3, [R1],#-4
        STR         R3, [R2], #-4
        SUBS R0, R0, #1
        BNE         TRANSFER_LOOP

STOP  B       STOP

        END
```

Output:
Before Execution: Array at RAM Location 0x20000000



```
Address: 0x20000000
0x20000000: 01 00 00 00 02 00 00 00 03 00 00 00 04 00 00 00 05 00 00 00 06 00 00 00 07 00 00 00 08 00 00 00 00 00 00 00 00 00 00 00 00 00
0x2000002A: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x20000054: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

After Execution: Overlapped array at RAM location 0x20000000



```
Address: 0x20000000
0x20000000: 01 00 00 00 02 00 00 00 01 00 00 00 02 00 00 00 03 00 00 00 04 00 00 00 05 00 00 00 06 00 00 00 07 00 00 00 08 00 00 00 00 00
0x2000002A: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x20000054: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

Space to answer viva questions:

Sl. No	**Criteria**	**Max Marks**	**Marks obtained**
Data sheet			
A	Analysis of program flow and instructions used/ Writing program	04	
B	Simulation/ Conduction of the experiment/ Analysis of Results	04	
C.	Viva	2	
	Total	10	

Experiment 02

PROGRAMS ON ARITHMETIC OPERATIONS

a) Write an ALP to perform 32-bit addition, subtraction, multiplication, and division on operands stored in RAM locations.

Algorithm:

1. Load a register with RAM location address 0x20000000 where condition to perform different operations is mentioned (1:Addition, 2:Subtraction, 3:multiplication, 4:Division)
2. Load 32-bit operands stored in subsequent RAM locations.
3. Check the condition and perform the specified operation as indicated in 1.
4. Store results in subsequent RAM locations.
5. Loop forever to terminate the program.

Program:

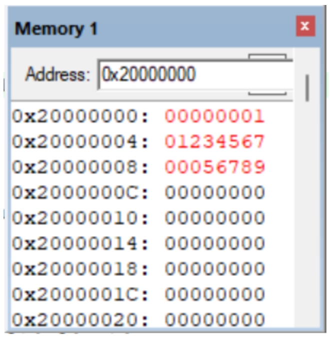
```
        PRESERVE8
Stack EQU 0x00000100
        AREA STACK, NOINIT, READWRITE
StackMem SPACE Stack

        AREA RESET, DATA , READONLY
        EXPORT_Vectors

__Vectors
        DCD StackMem + Stack
        DCD Reset_Handler

        AREA |.text| , CODE , READONLY
        ENTRY
        EXPORT Reset_Handler
Reset_Handler
Reset_Handler_End

        LDR R1, = 0X20000000                ;memory location for condition
        LDR R2, [R1]
        LDR R3, [R1, #4]                    ;Operand 1
        LDR R4, [R1, #8]                    ;Operand 2

        CMP R2, #1                          ;1: Addition
        BNE CHK_SUB
        ADDS R7, R3, R4                     ;R7=R3+R4
        B LAST

CHK_SUB CMP R2, #2                          ;2: subtraction
          BNE CHK_MUL
          SUBS R7, R3, R4                   ;R7=R3-R4
```

```
            B LAST

CHK_MUL CMP R2, #3                      ; 3: Multiplication
        BNE CHK_DIV
        UMULL R7, R8, R3, R4            ; [RdHi,RdLow]=Rn*Rm
        B LAST                          ;UMULL:Unsigned multiply long

CHK_DIV CMP R2, #4                      ; 4: Division
        BNE LAST
        UDIV R7, R3, R4                 ;R7=R3/R4

LAST    STR R7, [R1, #20]          ;store result in memory location 0x20000014
        STR R8, [R1, #24]          ;store the RdLow in memory location
                                   ;0x20000018 for multiplication.

STOP  B     STOP
        END
```

Output:
Before Execution:
In debug session, open memory window and type address:0x20000000. Right click on the content of any location and select unsigned -> int.
At RAM location 0x20000000, enter the condition code as per the table below.

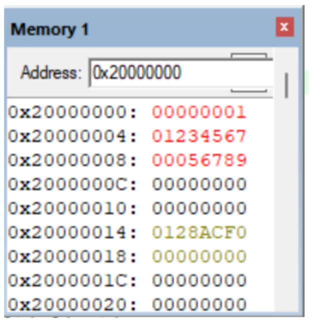
Operation	Condition code
Addition	1
Subtraction	2
Multiplication	3
Division	4

At subsequent RAM locations, enter 32-bit operands for the operation selected (addition operation)



```
0x20000000: 00000001
0x20000004: 01234567
0x20000008: 00056789
0x2000000C: 00000000
0x20000010: 00000000
0x20000014: 00000000
0x20000018: 00000000
0x2000001C: 00000000
0x20000020: 00000000
```

After Execution:



```
0x20000000: 00000001
0x20000004: 01234567
0x20000008: 00056789
0x2000000C: 00000000
0x20000010: 00000000
0x20000014: 0128ACF0
0x20000018: 00000000
0x2000001C: 00000000
0x20000020: 00000000
```

b) Write an ALP to perform 64-bit addition, subtraction and multiplication on operands stored in RAM locations.

Program:

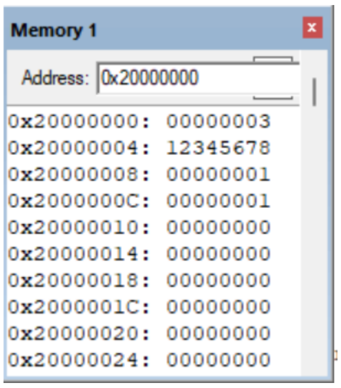
```
        PRESERVE8

Stack EQU 0x00000100

        AREA STACK, NOINIT, READWRITE
StackMem SPACE Stack

        AREA RESET, DATA , READONLY
        EXPORT_Vectors

__Vectors
        DCD StackMem + Stack
        DCD Reset_Handler

        AREA |.text| , CODE , READONLY
        ENTRY
        EXPORT Reset_Handler
Reset_Handler
Reset_Handler_End

        LDR R1, = 0X20000000                ; memory location for condition
        LDR R2, [R1]
        LDR R3, [R1, #4]            ; Operand 1: Lower 32 bits
        LDR R4, [R1, #8]            ; Operand 1: Upper 32 bits
        LDR R5, [R1, #12]           ; Operand 2: Lower 32 bits
        LDR R6, [R1, #16]           ; Operand 2: Upper 32 bits

        CMP R2, #1                  ;1= addition
        BNE CHK_SUB
        ADDS R7, R3, R5             ;Add the lower 32 bits
        ADC R8, R4, R6              ;Add the upper 32 bits with carry (if present)
        B LAST

CHK_SUB CMP R2, #2                          ;2=subtraction
```

```
            BNE CHK_MUL
            SUBS R7, R3, R5          ; Subtract the lower 32 bits
            SBC R8, R4, R6           ;Subtract the upper 32 bits with borrow (if present)
            B LAST

CHK_MUL CMP R2, #3                                 ;3=Multiplication
            BNE DEFAULT
            UMULL R7, R8, R3, R5          ;low*low
            UMULL R9, R10, R3, R6         ;low*high
            UMULL R6, R3, R4, R6          ;high*high
            ADDS R8, R8, R9
            ADCS R9, R10, R6
            ADC R10, R3, #0
            UMULL R5, R3, R4, R5          ;high*low
            ADDS R8, R8, R5
            ADCS R9, R9, R3
            ADC R10, R10, #0
            B LAST

DEFAULT MOV R7, #0                ;if no operation is done
           MOV R8, #0
           MOV R9, #0
           MOV R10, #0

LAST    STR R7, [R1, #20]        ;store result[31:0] in memory location 0x20000014
        STR R8, [R1, #24]        ;store result[63:32] in memory location 0x20000018
        STR R9, [R1, #28]        ;store result[95:64] in memory location 0x2000001c.
        STR R10, [R1, #32]       ;store result[127:96] in memory location 0x20000020.

STOP  B       STOP
           END
```

Output:
Before Execution:
In debug session, open memory window and type address:0x20000000. Right click on the content of any location and select unsigned -> int.
At RAM location 0x20000000, enter the condition code as per the table below.

Operation	Condition code
Addition	1
Subtraction	2
Multiplication	3

At subsequent RAM locations, enter 64-bit operands for the operation selected (multiplication operation).

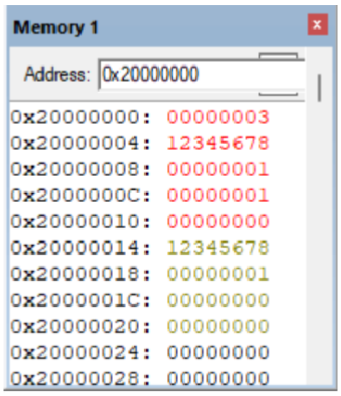


Memory 1

Address: 0x20000000

```
0x20000000: 00000003
0x20000004: 12345678
0x20000008: 00000001
0x2000000C: 00000001
0x20000010: 00000000
0x20000014: 00000000
0x20000018: 00000000
0x2000001C: 00000000
0x20000020: 00000000
0x20000024: 00000000
```

After Execution:



Memory 1

Address: 0x20000000

```
0x20000000: 00000003
0x20000004: 12345678
0x20000008: 00000001
0x2000000C: 00000001
0x20000010: 00000000
0x20000014: 12345678
0x20000018: 00000001
0x2000001C: 00000000
0x20000020: 00000000
0x20000024: 00000000
0x20000028: 00000000
```

Space to answer viva questions:

Sl. No	**Criteria**	**Max Marks**	**Marks obtained**
Data sheet			
A	Analysis of program flow and instructions used/ Writing program	04	
B	Simulation/ Conduction of the experiment/ Analysis of Results	04	
C.	Viva	2	
	Total	10	

Experiment 03

PROGRAMS ON SEARCHING & SORTING

a) Write an ALP to find an element in array of Elements using linear search.

Algorithm:

1. Set the first element of the array as the current element.
2. If the current element is the target element, return its index.
3. If the current element is not the target element and if there are more elements in the array, set the current element to the next element and repeat step 2.
4. If the current element is not the target element and there are no more elements in the array means that the element does not exist in the array.

Program:

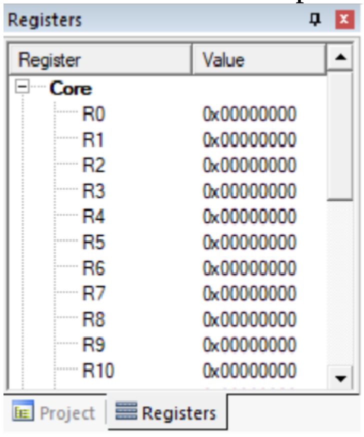
```
        PRESERVE8
Stack EQU 0x00000100

        AREA STACK,NOINIT,READWRITE
StackMem SPACE Stack

        AREA RESET, DATA , READONLY
        EXPORT_Vectors

__Vectors
        DCD StackMem + Stack
        DCD Reset_Handler

        AREA tempData, DATA, READONLY
Array DCD 1,2,3,55,5,6,7,8,9,10             ; 10 elements are initialized
Len EQU 10

        AREA |.text|, CODE , READONLY
        ENTRY
        EXPORT Reset_Handler
Reset_Handler
Reset_Handler_End


        LDR R0, =Array
        MOV R1, #Len
        MOV R2, #55                         ; Element to search for
        BL Linear_Search_Start


stop    B stop
```

```
Linear_Search_Start
        EXPORT Linear_Search_Start
        MOV R4, R0                          ; Loading base address of array to R4
        MOV R3, #0                          ; Initializing iterator
        MOV R0, #0                          ; Clearing R0 to make it available to
                                            ;return result

loop
        ADD R3,R3,#1
        LDR R5,[R4]
        CMP R5,R2
        ADDEQ R0,R0,R3
        MOVEQ PC,LR                         ; early stoping if element is found
        ADD R4,R4,#4
        CMP R3,R1
        BNE loop
        MOV PC,LR
        END
```

Output:

Before Execution:

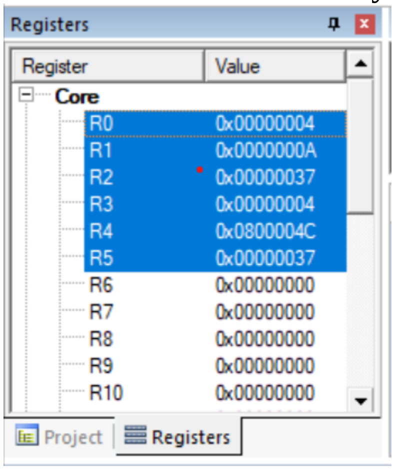
R0 is used to hold the position of search element if present and 0 otherwise.



```
Registers
Register    Value
⊟ Core
   R0       0x00000000
   R1       0x00000000
   R2       0x00000000
   R3       0x00000000
   R4       0x00000000
   R5       0x00000000
   R6       0x00000000
   R7       0x00000000
   R8       0x00000000
   R9       0x00000000
   R10      0x00000000
Project   Registers
```

After Execution:

R0 is 0x04 to indicate the position of element 55 in array.



```
Registers
Register    Value
⊟ Core
   R0       0x00000004
   R1       0x0000000A
   R2       0x00000037
   R3       0x00000004
   R4       0x0800004C
   R5       0x00000037
   R6       0x00000000
   R7       0x00000000
   R8       0x00000000
   R9       0x00000000
   R10      0x00000000
Project   Registers
```

b) Write an ALP to find an element in an array of Elements using binary search.

Algorithm:

1. Set the low index to the first element of the array and the high index to the last element.
2. Set the middle index to the average of the low and high indices.
3. If the element at the middle index is the target element, return the middle index.
4. If the target element is less than the element at the middle index, set the high index to the middle index – 1.
5. If the target element is greater than the element at the middle index, set the low index to the middle index + 1.
6. Repeat steps 3-6 until the element is found or the element is not present in the array.

Program:

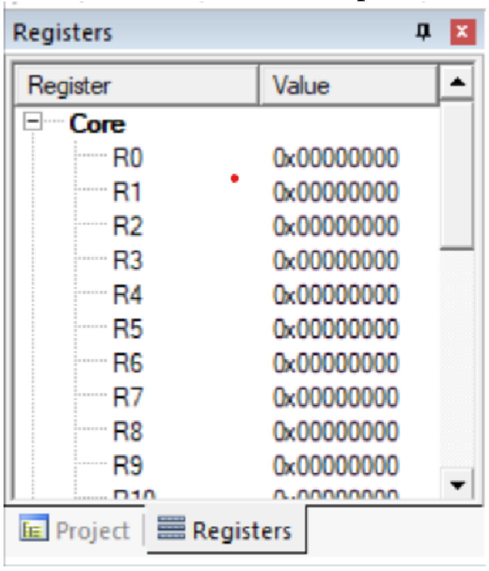
```
        PRESERVE8
Stack EQU 0x00000100
        AREA STACK,NOINIT,READWRITE
StackMem SPACE Stack

        AREA RESET,DATA,READONLY
        EXPORT_Vectors
__Vectors
        DCD StackMem + Stack
        DCD Reset_Handler
        AREA tempData,DATA,READONLY
Array DCD 1,2,3,4,5,6,7,8,9,10                    ; 10 elements are initialized


        AREA |.text| , CODE , READONLY
        ENTRY
        EXPORT Reset_Handler
Reset_Handler
Reset_Handler_End

        LDR R0, =Array                            ;Base address of an array
        MOV R1, #0                                ;Index of an first element
        MOV R2, #9                                ;Index of last element
        MOV R3, #0x8                              ;Search element
        BL Binary_Search_Start

foreverloop
```

```
        B foreverloop

Binary_Search_Start

        MOV R4, R0
        MOV R0, #0

Binary_Search_Loop
        ADD R5,R1,R2
        LSR R5,R5,#1                    ;Calculating mid index
        LSL R7,R5,#2                    ;Calculating element offset
        LDR R6,[R4,R7]
        CMP R3,R6
        ADDEQ R0,R0,R5
        ADDEQ R0,R0,#1
        MOVEQ PC,LR                      ; early stopping if element is found
        SUBLT R5,R5,#1
        MOVLT R2,R5
        ADDGT R5,R5,#1
        MOVGT R1,R5
        CMP R1,R2
        MOVEQ PC,R8
        B Binary_Search_Loop
        END
```

Output:

Before Execution:

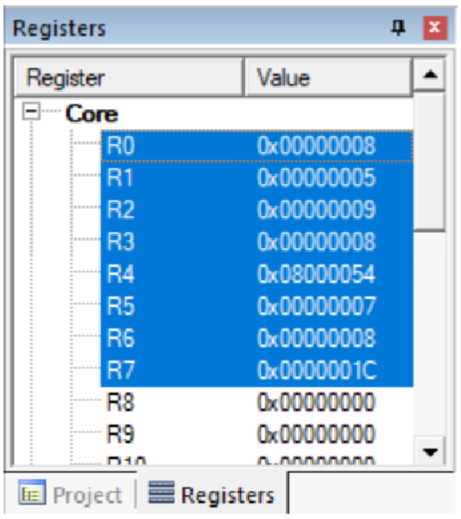
R0 is used to hold the position of search element if present and 0 otherwise.



```
Registers
Register    Value
Core
  R0        0x00000000
  R1        0x00000000
  R2        0x00000000
  R3        0x00000000
  R4        0x00000000
  R5        0x00000000
  R6        0x00000000
  R7        0x00000000
  R8        0x00000000
  R9        0x00000000
```

After Execution:

R0 is 0x08 to indicate the position of element 8 in array.



Register	Value
Core	
R0	0x00000008
R1	0x00000005
R2	0x00000009
R3	0x00000008
R4	0x08000054
R5	0x00000007
R6	0x00000008
R7	0x0000001C
R8	0x00000000
R9	0x00000000

Space to answer viva questions:

Sl. No	Criteria	Max Marks	Marks obtained
Data sheet			
A	Analysis of program flow and instructions used/ Writing program	04	
B	Simulation/ Conduction of the experiment/ Analysis of Results	04	
C.	Viva	2	
	Total	10	

DISCOVERY KIT WITH STM32F407VG MCU

The STM32F4DISCOVERY Discovery kit allows users to easily develop applications with the STM32F407VG high-performance microcontroller with the Arm® Cortex®-M4 32-bit core. Based on STM32F407VG, it includes an ST-LINK/V2-A embedded debug tool, one ST-MEMS digital accelerometer, one digital microphone, one audio DAC with integrated class D speaker driver, LEDs, push-buttons and a USB OTG Micro-AB connector. Specialized add-on boards can be connected by means of the extension header connectors.

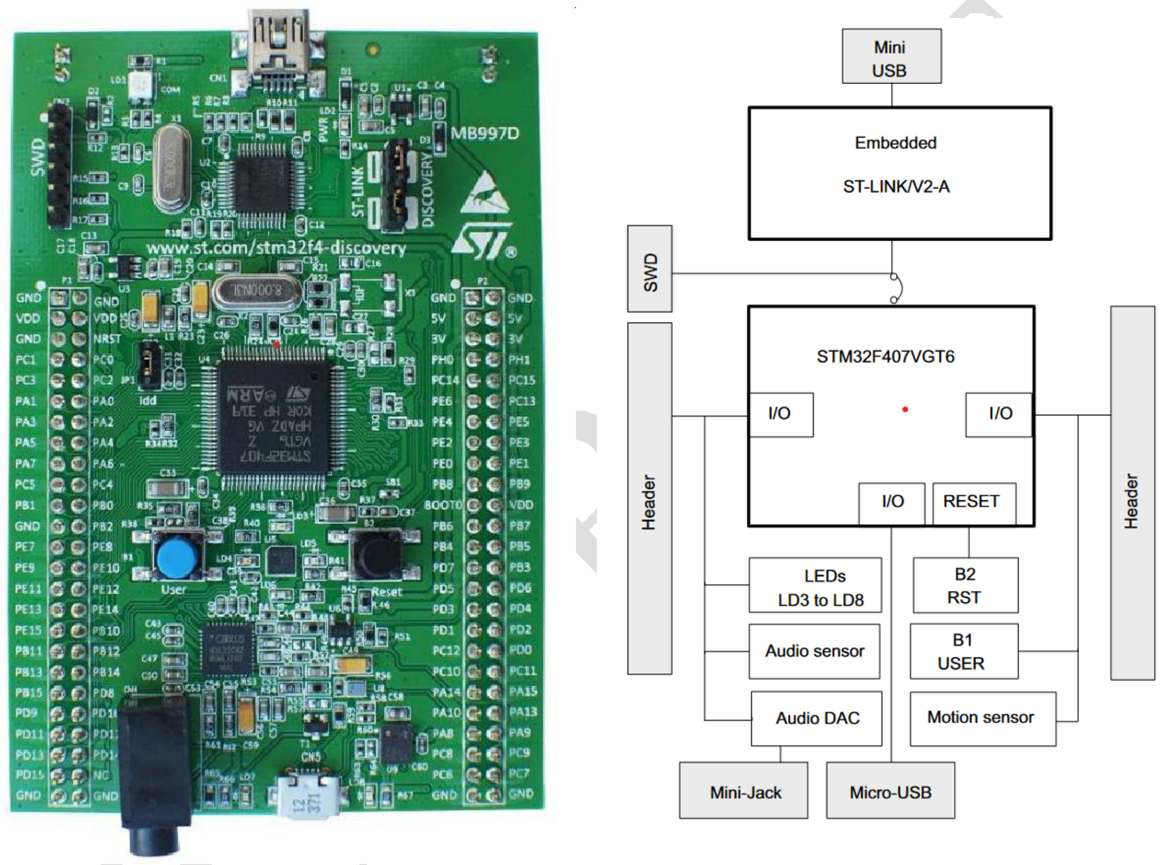


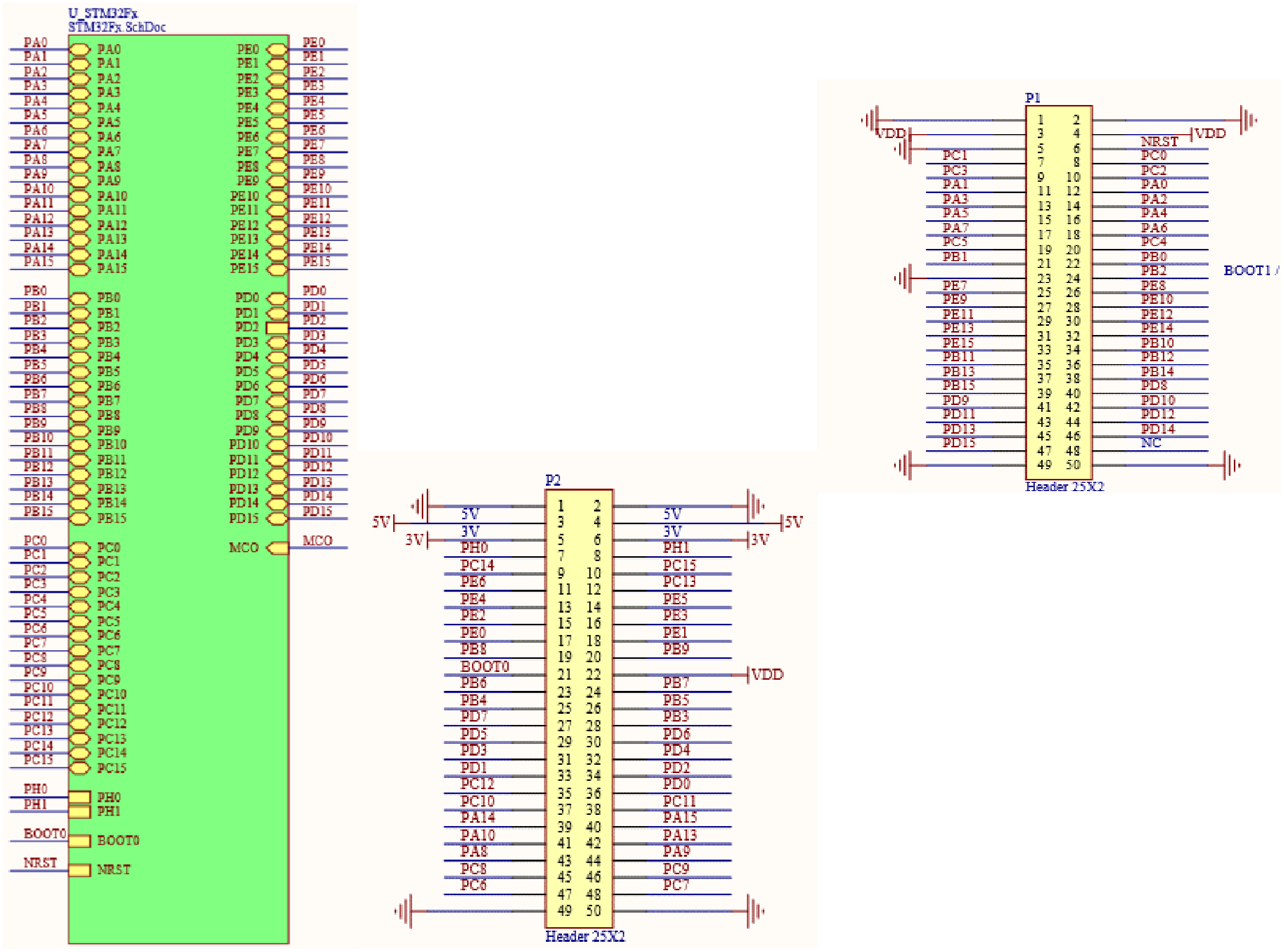
Figure: STM32F4DISCOVERY Discovery kit and block diagram

Features:

- STM32F407VGT6 microcontroller featuring 32-bit Arm ®(a) Cortex ® -M4 with FPU core, 1-Mbyte Flash memory, 192-Kbyte RAM in an LQFP100 package
- USB OTG FS
- ST MEMS 3-axis accelerometer
- ST-MEMS audio sensor omni-directional digital microphone
- Audio DAC with integrated class D speaker driver
- User and reset push-buttons
- Eight LEDs:
  - LD1 (red/green) for USB communication
  - LD2 (red) for 3.3 V power on

  - Four user LEDs, LD3 (orange), LD4 (green), LD5 (red) and LD6 (blue)
  - Two USB OTG LEDs, LD7 (green) V BUS and LD8 (red) over-current
- Board connectors:
  - USB with Micro-AB
  - Stereo headphone output jack
  - 2.54 mm pitch extension header for all LQFP100 I/Os for quick connection to prototyping board and easy probing
- Flexible power-supply options: ST-LINK, USB V BUS , or external sources
- External application power supply: 3 V and 5 V
- Comprehensive free software including a variety of examples, part of STM32CubeF4
- MCU Package, or STSW-STM32068 for using legacy standard libraries
- On-board ST-LINK/V2-A debugger/programmer with USB re-enumeration capability: mass storage, Virtual COM port, and debug port
- Support of a wide choice of Integrated Development Environments (IDEs) including IAR Embedded Workbench®, MDK-ARM, and STM32CubeIDE.

Schematics of MCU & Pin assignment to P1 and P2.



STM32CubeMX

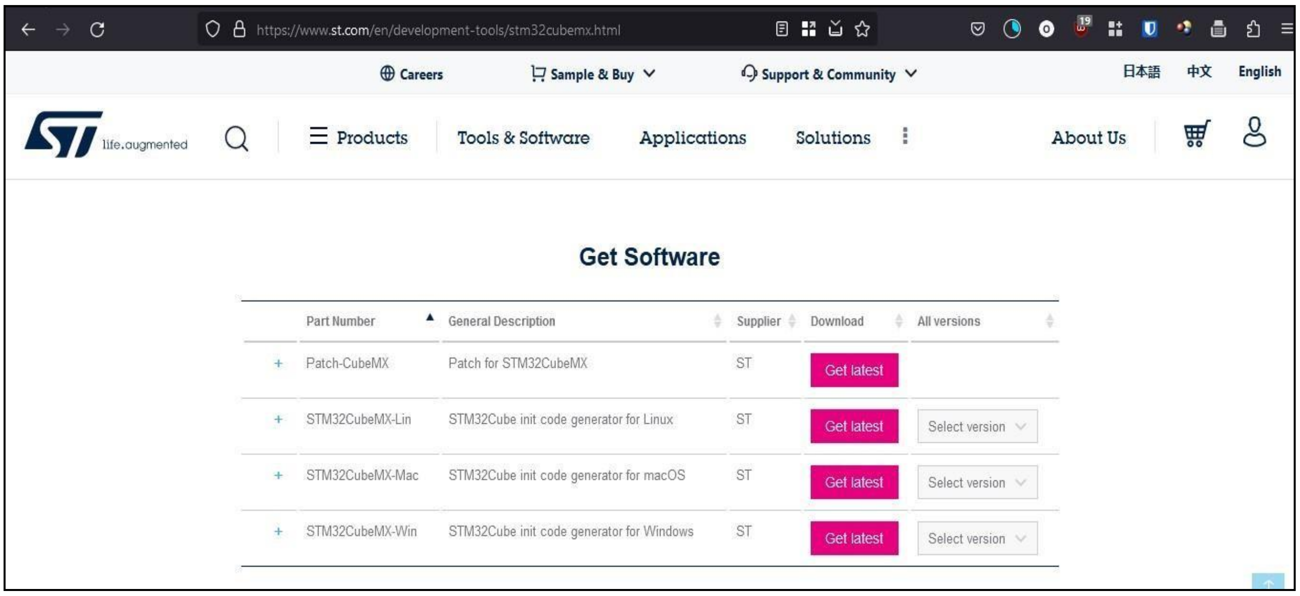
STM32CubeMX is a graphical tool that allows a very easy configuration of STM32 microcontrollers and microprocessors, as well as the generation of the corresponding initialization C code for the Arm® Cortex®-M core.

Features:

- Intuitive STM32 microcontroller and microprocessor selection
- Rich easy-to-use graphical user interface allowing the configuration of:
  - Pinout with automatic conflict resolution
  - Peripherals and middleware functional modes with dynamic validation of parameter constraints for Arm® Cortex®-M core
  - Clock tree with dynamic validation of the configuration
  - Power sequence with estimated consumption results
- Generation of initialization C code project, compliant with IAR Embedded Workbench®, MDK-ARM and STM32CubeIDE (GCC compilers) for Arm® Cortex®-M core
- Generation of a partial Linux® Device Tree for Arm® Cortex®-A core (STM32 microprocessors)
- Development of enhanced STM32Cube Expansion Packages thanks to STM32PackCreator
- Integration of STM32Cube Expansion packages into the project
- Availability as standalone software running on Windows®, Linux® and macOS® operating systems and 64-bit Java Runtime environment.

Downloading and Installing STM32CubeMX

Install STMCubeMX from the official website https://www.st.com/en/development-tools/stm32cubemx.html as shown in figure by clicking on Get Latest button or download.



Get Software

	Part Number	General Description	Supplier	Download	All versions
+	Patch-CubeMX	Patch for STM32CubeMX	ST	Get latest	
+	STM32CubeMX-Lin	STM32Cube init code generator for Linux	ST	Get latest	Select version
+	STM32CubeMX-Mac	STM32Cube init code generator for macOS	ST	Get latest	Select version
+	STM32CubeMX-Win	STM32Cube init code generator for Windows	ST	Get latest	Select version

INTERFACING LED,7 SEGMENT DISPLAY AND PUSH BUTTON

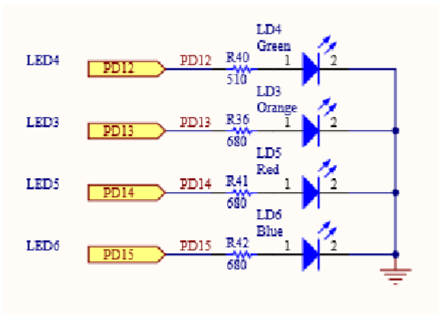
Experiment 04

a) Interfacing LED to GPIOS

Write C program to toggle LED connected to PD15.

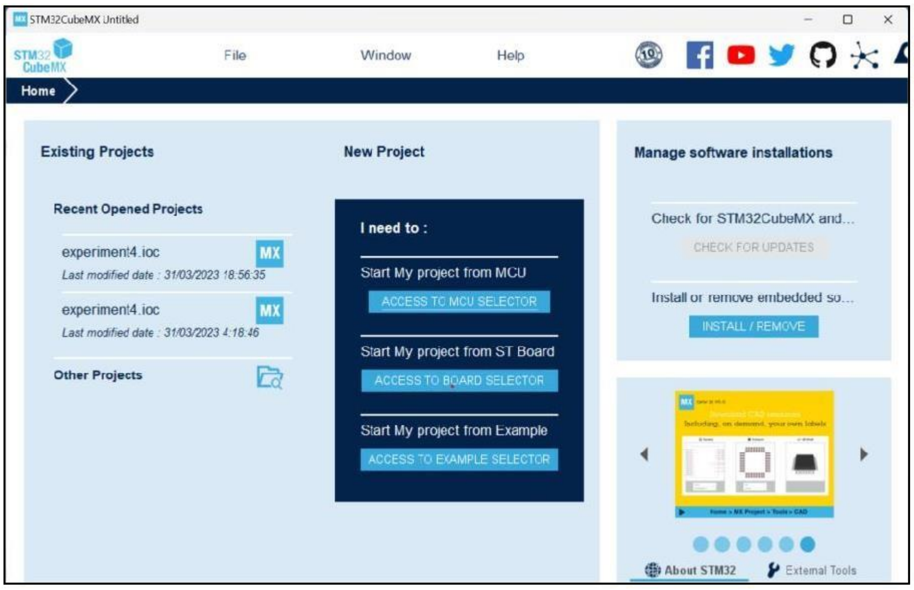
Schematics of LEDs

The board is supported with 4 LEDs connected to GPIOs as shown in figure below.

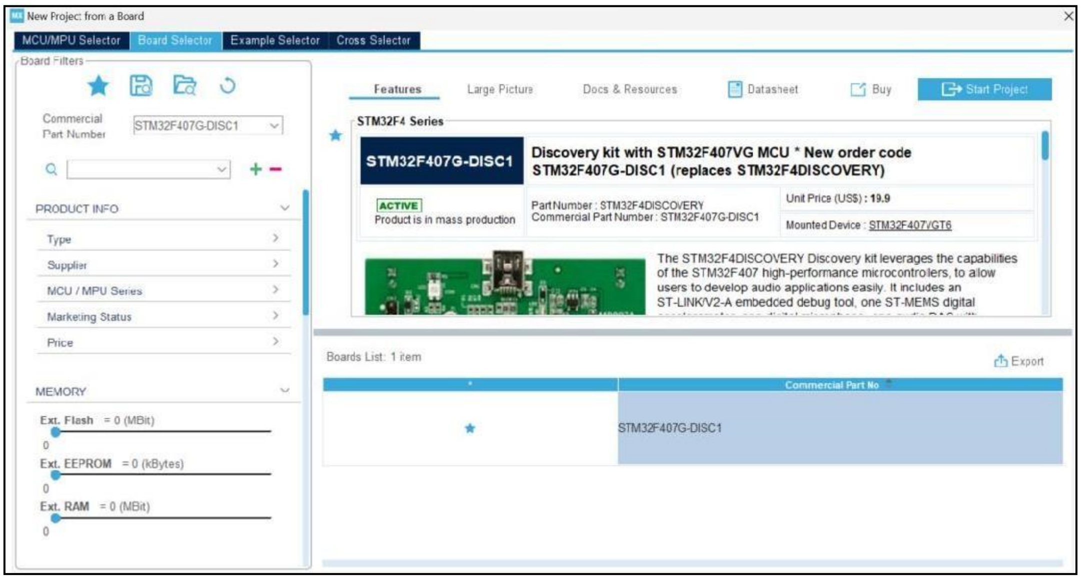


Procedure in STM32CubeMX to create project

1. Open the STM32CubeMx software and click on Access to board selector under new project. Select board as STM32F407G-DISC1.


STM32CubeMX Untitled
STM32 CubeMX
File
Window
Help
Home
Existing Projects
Recent Opened Projects
experiment4.ioc
Last modified date : 31/03/2023 18:56:35
experiment4.ioc
Last modified date : 31/03/2023 4:18:46
Other Projects
New Project
I need to :
Start My project from MCU
ACCESS TO MCU SELECTOR
Start My project from ST Board
ACCESS TO BOARD SELECTOR
Start My project from Example
ACCESS TO EXAMPLE SELECTOR
Manage software installations
Check for STM32CubeMX and...
CHECK FOR UPDATES
Install or remove embedded so...
INSTALL / REMOVE
About STM32
External Tools

2. Select board as STM32F407G-DISC1. Click on start project. Select yes for initialize all peripherals in default mode.



Commercial Part Number: STM32F407G-DISC1

Ext. Flash = 0 (MBit)

Ext. EEPROM = 0 (kBytes)

Ext. RAM = 0 (MBit)

STM32F4 Series

STM32F407G-DISC1

Discovery kit with STM32F407VG MCU * New order code STM32F407G-DISC1 (replaces STM32F4DISCOVERY)

ACTIVE
Product is in mass production

PartNumber : STM32F4DISCOVERY
Commercial Part Number : STM32F407G-DISC1

Unit Price (US$) : 19.9

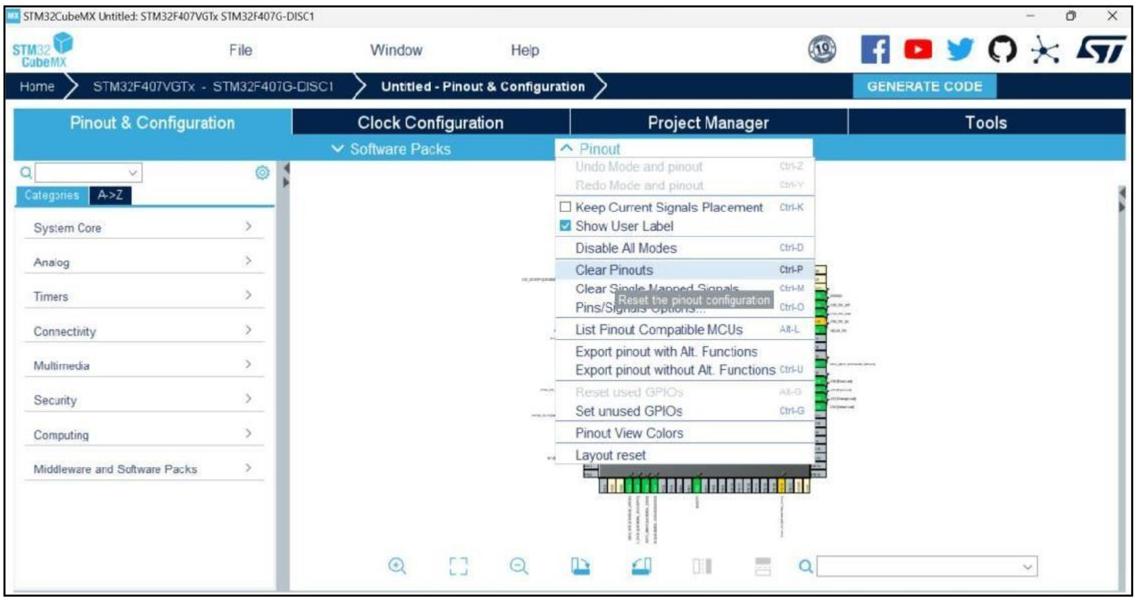
Mounted Device : STM32F407VGT6

The STM32F4DISCOVERY Discovery kit leverages the capabilities of the STM32F407 high-performance microcontrollers, to allow users to develop audio applications easily. It includes an ST-LINK/V2-A embedded debug tool, one ST-MEMS digital

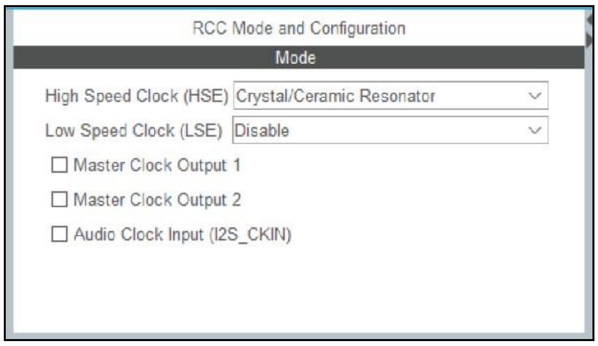
Boards List: 1 item

*	Commercial Part No
★	STM32F407G-DISC1

3. In STM32CubeMx window, in pinout and configuration tab, select pinout drop down list and click on clear pinout configuration.


STM32CubeMX Untitled: STM32F407VGTx STM32F407G-DISC1
File
Window
Help
Home
STM32F407VGTx - STM32F407G-DISC1
Untitled - Pinout & Configuration
GENERATE CODE
Pinout & Configuration
Clock Configuration
Project Manager
Tools
Software Packs
Pinout
Categories
A->Z
System Core
Analog
Timers
Connectivity
Multimedia
Security
Computing
Middleware and Software Packs
Undo Mode and pinout Ctrl-Z
Redo Mode and pinout Ctrl-Y
Keep Current Signals Placement Ctrl-K
Show User Label
Disable All Modes Ctrl-D
Clear Pinouts Ctrl-P
Clear Single Mapped Signals Ctrl-M
Pins/Signals Options... Ctrl-O
Reset the pinout configuration
List Pinout Compatible MCUs Alt-L
Export pinout with Alt. Functions
Export pinout without Alt. Functions Ctrl-U
Reset used GPIOs Alt-G
Set unused GPIOs Ctrl-G
Pinout View Colors
Layout reset

4. Under System Core, select the Reset and clock control(RCC) and for high speed clock(HSE) select Crystal/Ceramic Resonator. Select SYS and for debug select serial wire.



RCC Mode and Configuration

Mode

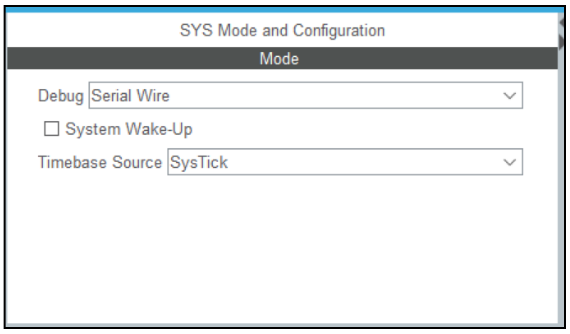
High Speed Clock (HSE) Crystal/Ceramic Resonator

Low Speed Clock (LSE) Disable

☐ Master Clock Output 1

☐ Master Clock Output 2

☐ Audio Clock Input (I2S_CKIN)



SYS Mode and Configuration

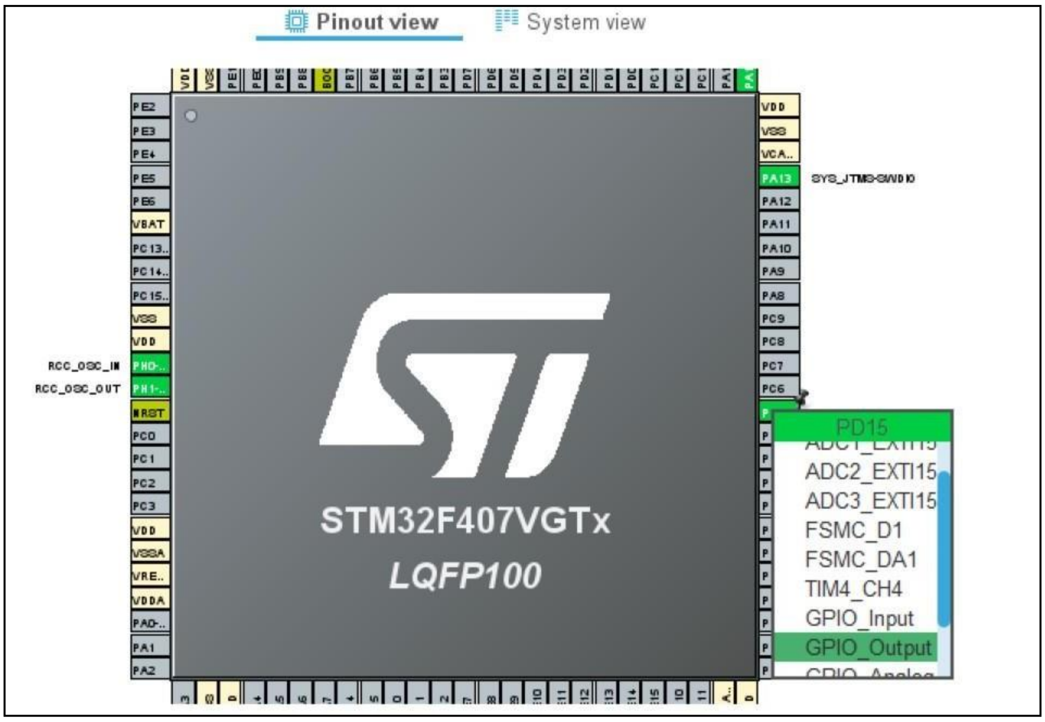
Mode

Debug Serial Wire

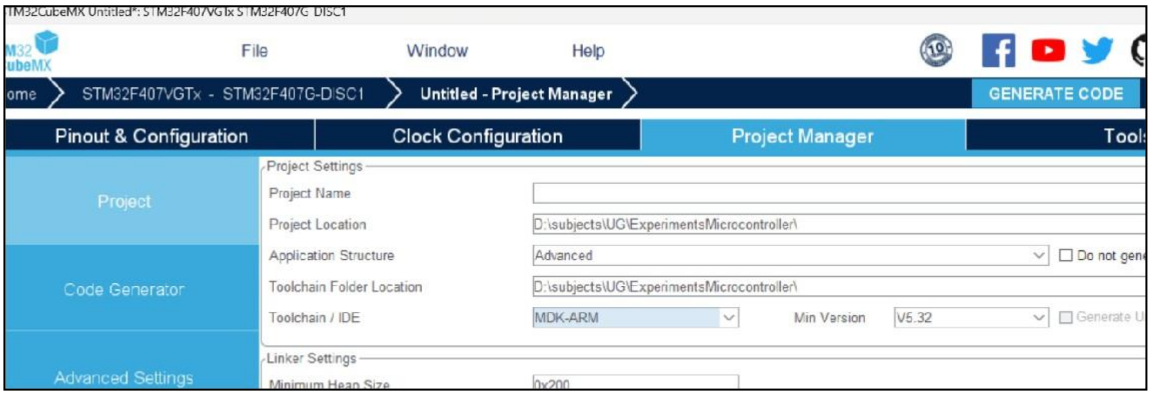
☐ System Wake-Up

Timebase Source SysTick

5. Select PD15(connected to blue LED). Left click on PD15 and select the GPIO_Output functionality in Pinout view. Note that, other pins (PD12,PD13,PD14) can be used.



6. Default configuration works for clock. Note that, by default internal RC oscillator running at 16 MHz is selected for operation. The user can change the clock to source external crystal running at 8 MHz.
7. Select Project manger tab. Enter project name, project location. Select tool chain IDE as MDK-ARM and let the other options at default values.


STM32CubeMX Untitled*: STM32F407VGTx STM32F407G-DISC1

File Window Help

Home > STM32F407VGTx - STM32F407G-DISC1 > Untitled - Project Manager > GENERATE CODE

Pinout & Configuration | Clock Configuration | Project Manager | Tools

Project | Code Generator | Advanced Settings

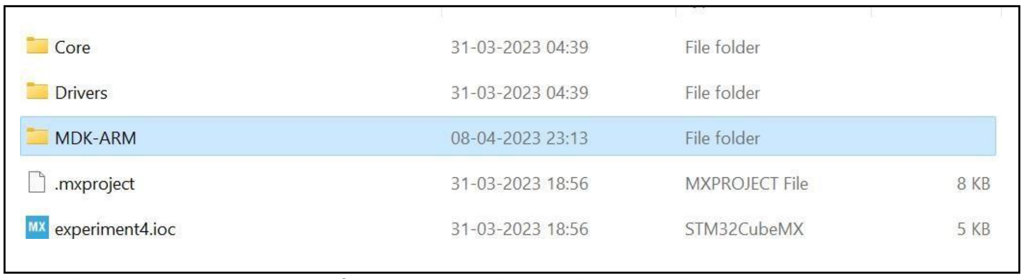
Project Settings
Project Name
Project Location D:\subjects\UG\ExperimentsMicrocontroller\
Application Structure Advanced ☐ Do not gen
Toolchain Folder Location D:\subjects\UG\ExperimentsMicrocontroller\
Toolchain / IDE MDK-ARM Min Version V5.32 ☐ Generate U

Linker Settings
Minimum Heap Size 0x200

8. Click on Generate code. This option will generate project file in Keil and initial configuration files in the location specified.

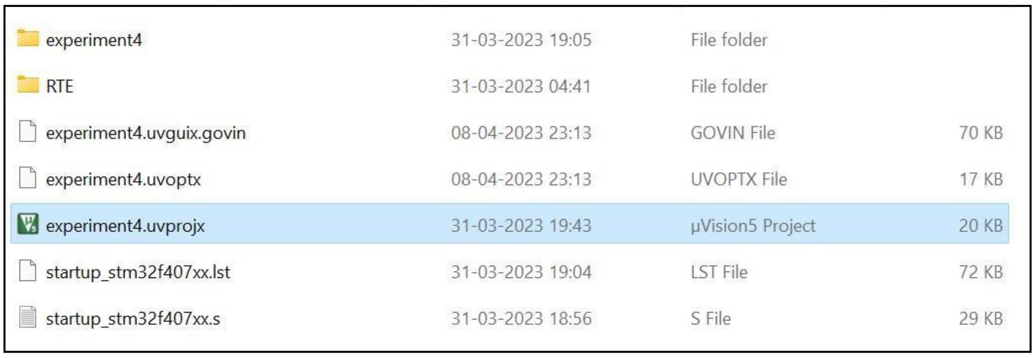
Procedure in Keil IDE to execute program

1. Go to project folder and open folder MDK-ARM.



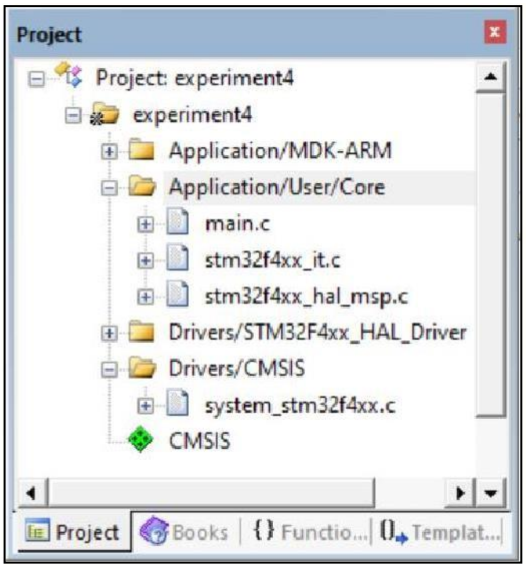
Core	31-03-2023 04:39	File folder	
Drivers	31-03-2023 04:39	File folder	
MDK-ARM	08-04-2023 23:13	File folder	
.mxproject	31-03-2023 18:56	MXPROJECT File	8 KB
experiment4.ioc	31-03-2023 18:56	STM32CubeMX	5 KB

2. Click on uvision project file to open the code in Keil IDE.



experiment4	31-03-2023 19:05	File folder	
RTE	31-03-2023 04:41	File folder	
experiment4.uvguix.govin	08-04-2023 23:13	GOVIN File	70 KB
experiment4.uvoptx	08-04-2023 23:13	UVOPTX File	17 KB
experiment4.uvprojx	31-03-2023 19:43	µVision5 Project	20 KB
startup_stm32f407xx.lst	31-03-2023 19:04	LST File	72 KB
startup_stm32f407xx.s	31-03-2023 18:56	S File	29 KB

3. In Keil IDE, different folders and files generated can be seen in project window. The main.c is listed under Application/User/Core. In the main.c file, user can add statements to perform intended operation. In folder Drivers/STM32F4xx_HAL_Driver, supporting files for peripherals will be generated.



```
Project
Project: experiment4
  experiment4
    Application/MDK-ARM
    Application/User/Core
      main.c
      stm32f4xx_it.c
      stm32f4xx_hal_msp.c
    Drivers/STM32F4xx_HAL_Driver
    Drivers/CMSIS
      system_stm32f4xx.c
    CMSIS
```

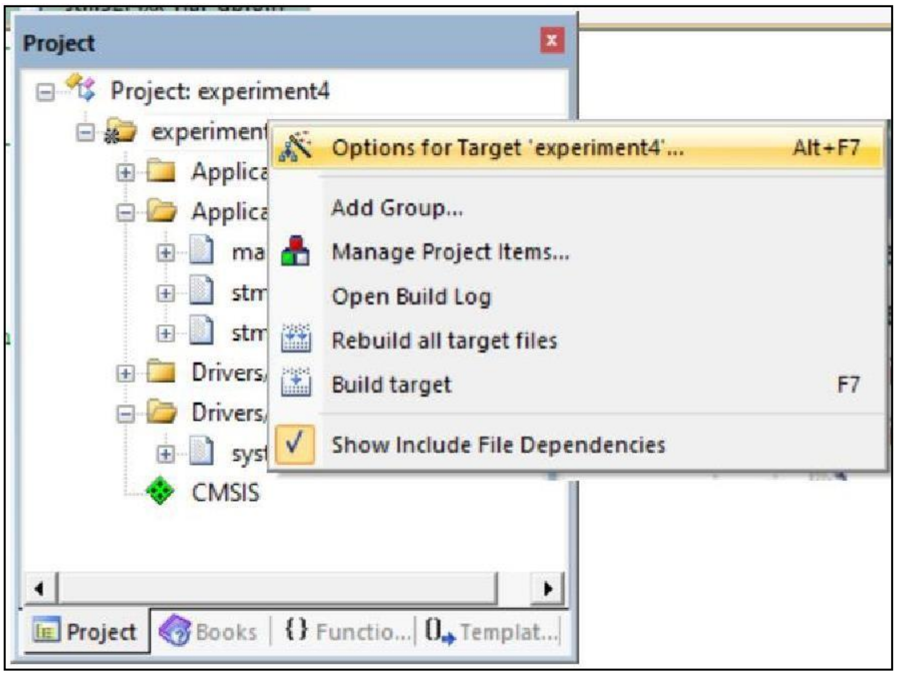
4. Open the main.c to edit the code as follows. Note that, only statements shown in bold must be typed and all other statements will be automatically generated.

Program:

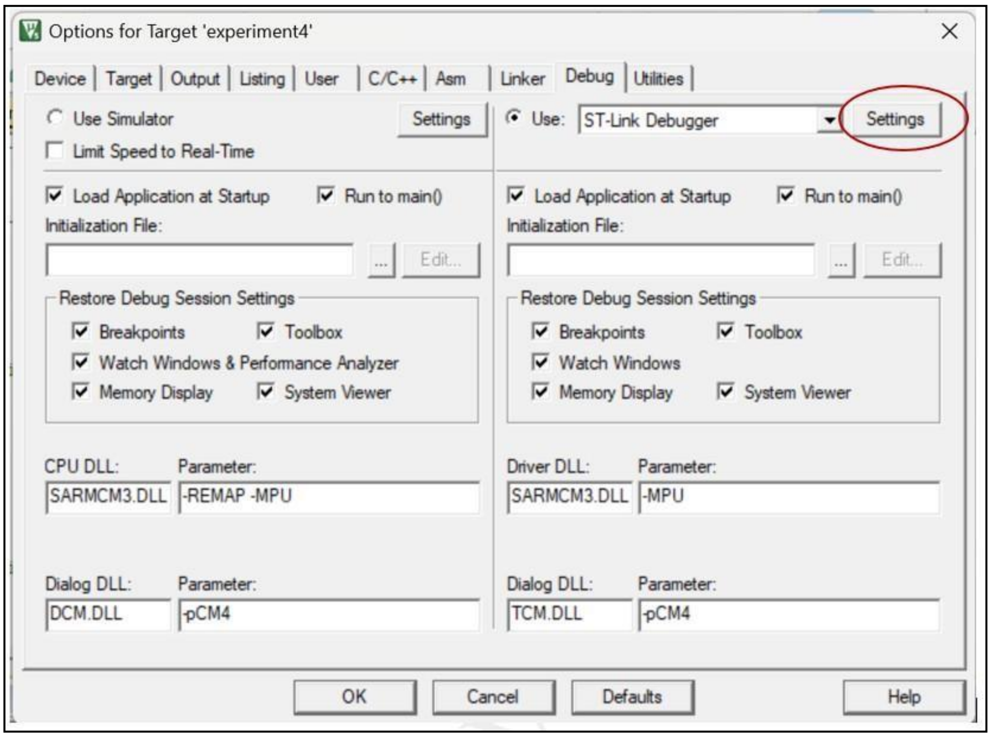
```
#include "main.h"

void SystemClock_Config(void);
static void MX_GPIO_Init(void);

int main(void)
{
```

```
            HAL_Init(); // Reset of all peripherals, Initializes Flash interface and the
                              Systick.

            SystemClock_Config(); //Configure the system clock

            MX_GPIO_Init();         //Initialize all configured peripherals

      while (1)
      {
            HAL_GPIO_WritePin(GPIOD,GPIO_PIN_15,GPIO_PIN_SET);
            HAL_Delay(1000);
            HAL_GPIO_WritePin(GPIOD,GPIO_PIN_15,GPIO_PIN_RESET);
            HAL_Delay(1000);
      }
}
```

5. Right click on experiment title in project window and select the options for target in pop up window.



6. In debug tab, select the settings for ST-Link debugger.


Options for Target 'experiment4'

Device | Target | Output | Listing | User | C/C++ | Asm | Linker | Debug | Utilities

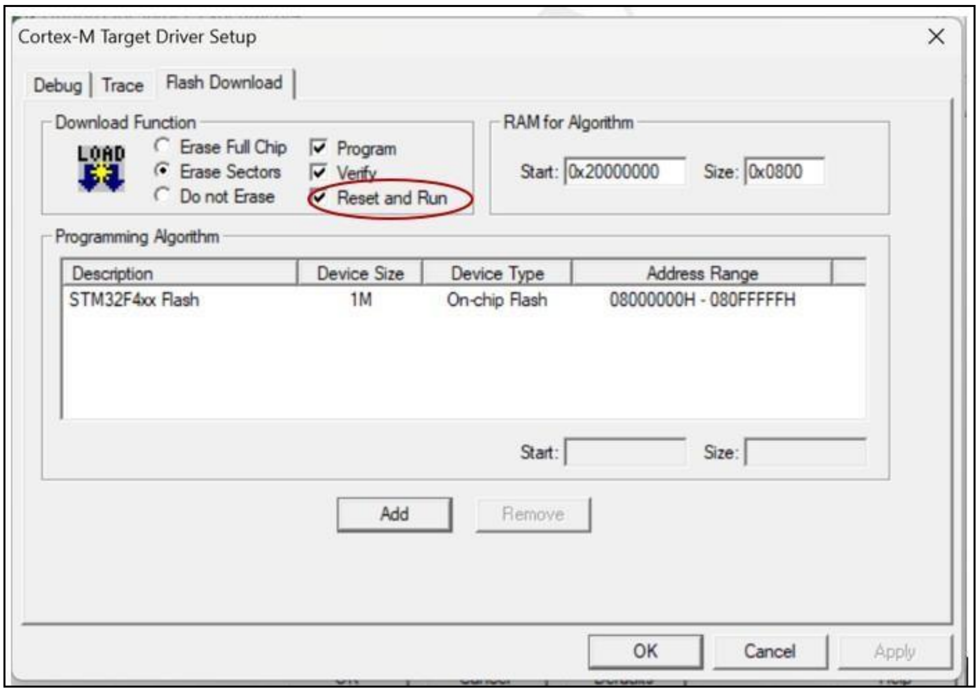
Use Simulator [ ] — Settings
Limit Speed to Real-Time [ ]
[✓ Use: ST-Link Debugger] — Settings

[✓] Load Application at Startup	[✓] Run to main()
Initialization File:	
Restore Debug Session Settings	
[✓] Breakpoints	[✓] Toolbox
[✓] Watch Windows & Performance Analyzer	
[✓] Memory Display	[✓] System Viewer
CPU DLL: SARMCM3.DLL	Parameter: -REMAP -MPU
Dialog DLL: DCM.DLL	Parameter: -pCM4

[✓] Load Application at Startup	[✓] Run to main()
Initialization File:	
Restore Debug Session Settings	
[✓] Breakpoints	[✓] Toolbox
[✓] Watch Windows	
[✓] Memory Display	[✓] System Viewer
Driver DLL: SARMCM3.DLL	Parameter: -MPU
Dialog DLL: TCM.DLL	Parameter: -pCM4

OK | Cancel | Defaults | Help

7. Check Reset and Run option in Flash download tab and click on Ok.


Cortex-M Target Driver Setup

Debug | Trace | Flash Download

Download Function
- Erase Full Chip [ ]
- [✓ Erase Sectors]
- Do not Erase [ ]
- [✓] Program
- [✓] Verify
- [✓] Reset and Run

RAM for Algorithm
Start: 0x20000000 Size: 0x0800

Programming Algorithm

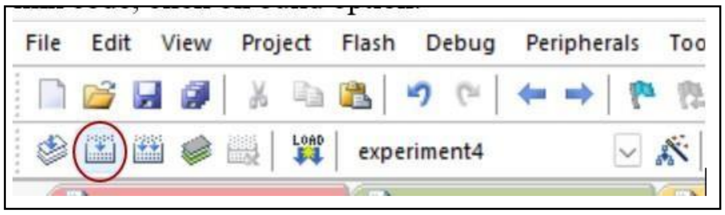
Description	Device Size	Device Type	Address Range
STM32F4xx Flash	1M	On-chip Flash	08000000H - 080FFFFFH

Start: Size:

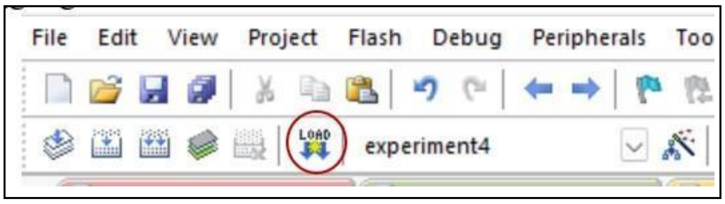
Add | Remove

OK | Cancel | Apply

8. To compile and link code, click on build option.



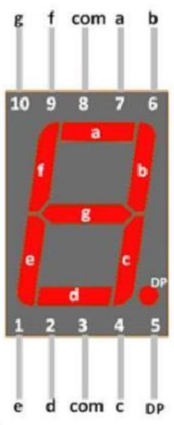
9. If the source code is error free, program will successfully build and target will be created. Make sure the board is connected and load the application by clicking on load option. The toggling of blue LED can be seen on the board.



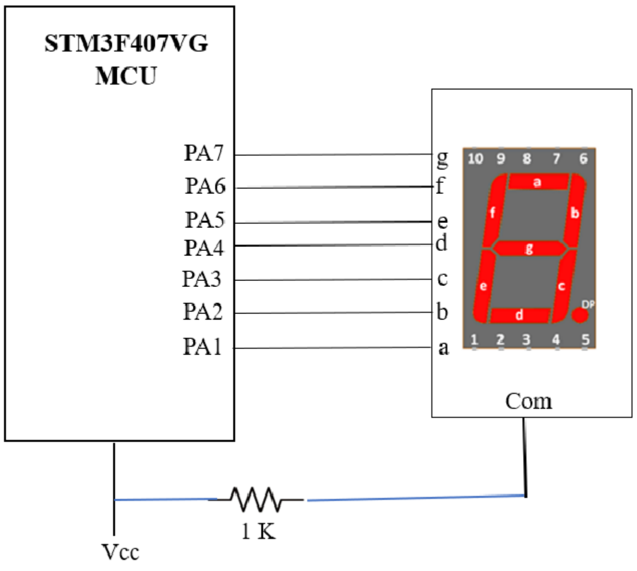
b) Interfacing Seven Segment display

Write C program to display digits between 0 to 9 on seven segment interface.

FND 507 Pin diagram: Common anode type



Interfacing diagram:



Follow the similar procedure in STM32CubeMx and configure PA1 to PA7 as GPIO_Output.

Program:

```
#include "main.h"

void SystemClock_Config(void);
```

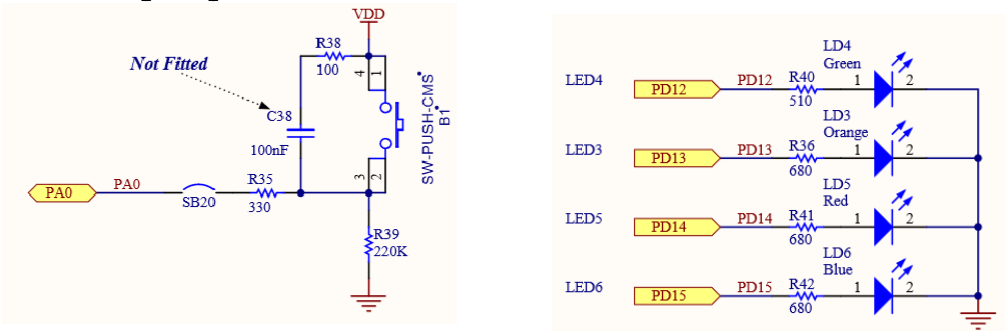
```
static void MX_GPIO_Init(void);
int main(void)
{
  HAL_Init();
  SystemClock_Config();
  MX_GPIO_Init();

  while (1)
  {

    //Display 0
    HAL_GPIO_WritePin(GPIOA,GPIO_PIN_1|GPIO_PIN_2|GPIO_PIN_3|GPIO_PIN_4|
                              GPIO_PIN_5| GPIO_PIN_6, GPIO_PIN_RESET);
    HAL_GPIO_WritePin(GPIOA,GPIO_PIN_7, GPIO_PIN_SET);
    HAL_Delay(1000);

    //Display 1
    HAL_GPIO_WritePin(GPIOA,GPIO_PIN_2|GPIO_PIN_3, GPIO_PIN_RESET);
    HAL_GPIO_WritePin(GPIOA,GPIO_PIN_1|GPIO_PIN_4|GPIO_PIN_5|
                                      GPIO_PIN_6|GPIO_PIN_7, GPIO_PIN_SET);
    HAL_Delay(1000);

    //Display 2
    HAL_GPIO_WritePin(GPIOA,GPIO_PIN_1|GPIO_PIN_2|GPIO_PIN_7|GPIO_PIN_5|G
                                      PIO_PIN_4, GPIO_PIN_RESET);
    HAL_GPIO_WritePin(GPIOA,GPIO_PIN_3|GPIO_PIN_6, GPIO_PIN_SET);
    HAL_Delay(1000);

    //Display 3
    HAL_GPIO_WritePin(GPIOA,GPIO_PIN_1|GPIO_PIN_2|GPIO_PIN_7|GPIO_PIN_3|
                                   GPIO_PIN_4, GPIO_PIN_RESET);
    HAL_GPIO_WritePin(GPIOA,GPIO_PIN_5|GPIO_PIN_6, GPIO_PIN_SET);
    HAL_Delay(1000);

    //Display 4
    HAL_GPIO_WritePin(GPIOA,GPIO_PIN_6|GPIO_PIN_2|GPIO_PIN_7|GPIO_PIN_3
                           ,GPIO_PIN_RESET);

    HAL_GPIO_WritePin(GPIOA,GPIO_PIN_1|GPIO_PIN_4|GPIO_PIN_5,
                               GPIO_PIN_SET);
    HAL_Delay(1000);
    //Display 5
    HAL_GPIO_WritePin(GPIOA,GPIO_PIN_6|GPIO_PIN_1|GPIO_PIN_7|GPIO_PIN_
                                        3|GPIO_PIN_4 ,GPIO_PIN_RESET);
    HAL_GPIO_WritePin(GPIOA,GPIO_PIN_2|GPIO_PIN_5, GPIO_PIN_SET);
    HAL_Delay(1000);
  //Display 6
  HAL_GPIO_WritePin(GPIOA,GPIO_PIN_6|GPIO_PIN_1|GPIO_PIN_7|GPIO_PIN_
                                        3|GPIO_PIN_4|GPIO_PIN_5,GPIO_PIN_RESET);
         HAL_GPIO_WritePin(GPIOA,GPIO_PIN_2 , GPIO_PIN_SET);
  HAL_Delay(1000);

  //Display 7
  HAL_GPIO_WritePin(GPIOA,GPIO_PIN_1|GPIO_PIN_2|GPIO_PIN_3
                             ,GPIO_PIN_RESET);
  HAL_GPIO_WritePin(GPIOA,GPIO_PIN_5|GPIO_PIN_6|GPIO_PIN_7|GPIO_PIN_
                                   4 , GPIO_PIN_SET);
  HAL_Delay(1000);

    //Display 8
         HAL_GPIO_WritePin(GPIOA,GPIO_PIN_1|GPIO_PIN_2|GPIO_PIN_3|GPIO_PIN_4|
                                      GPIO_PIN_5|GPIO_PIN_6|GPIO_PIN_7,GPIO_PIN_RESET);
         HAL_Delay(1000);

  //Display9
  HAL_GPIO_WritePin(GPIOA,GPIO_PIN_1|GPIO_PIN_2|GPIO_PIN_3|GPIO_PIN_
                    4|GPIO_PIN_7|GPIO_PIN_6,GPIO_PIN_RESET);
    HAL_GPIO_WritePin(GPIOA,GPIO_PIN_5 , GPIO_PIN_SET);
     HAL_Delay(1000);
        }
      }
```

In Keil IDE, follow the similar procedure as mentioned in the previous program to edit, build and run program.

c) Interfacing push button and LEDs

Write a C program to interface push button connected to PA0 and toggle orange and blue LEDs upon push button press event else toggle green and red LEDs.

Interfacing diagram:



Follow the similar procedure in STM32CubeMx and configure PA0 as GPIO_Input and PD12 to PD15 as GPIO_Output.

Program:

```
#include "main.h"
void SystemClock_Config(void);
static void MX_GPIO_Init(void);
int main(void)
{
  HAL_Init();
  SystemClock_Config();
  MX_GPIO_Init();
  while (1)
  {

    if(HAL_GPIO_ReadPin(GPIOA,GPIO_PIN_0)==GPIO_PIN_SET){
            HAL_GPIO_WritePin(GPIOD,GPIO_PIN_13|GPIO_PIN_15,GPIO_PIN_SET);
            HAL_Delay(500);
            HAL_GPIO_WritePin(GPIOD,GPIO_PIN_13|GPIO_PIN_15,GPIO_PIN_RESET);
            HAL_Delay(500);
      }
   else{
            HAL_GPIO_WritePin(GPIOD,GPIO_PIN_12|GPIO_PIN_14,GPIO_PIN_SET);
            HAL_Delay(500);
            HAL_GPIO_WritePin(GPIOD,GPIO_PIN_12|GPIO_PIN_14,GPIO_PIN_RESET);
            HAL_Delay(500);
            }
         }


       }
```

Space to answer viva questions:

Sl. No	**Criteria**	**Max Marks**	**Marks obtained**
Data sheet			
A	Analysis of program flow and instructions used/ Writing program	04	
B	Simulation/ Conduction of the experiment/ Analysis of Results	04	
C.	Viva	2	
	Total	10	

PROGRAM DIGITAL IOs TO CONTROL STEPPER AND MOTOR DRIVERS

Experiment 05

The stepper motor is a widely used device that translates electrical pulses into mechanical movement. In applications such as disk drives, dot matrix printers, and robotics, the stepper motor is used for position control. Stepper motors commonly have a central magnet called rotor or shaft surrounded by stator. The most common stepper motors have four stator windings that are paired with center-tapped common as shown in figure 5.1. This type of stepper motor is commonly called as a four-phase or unipolar stepper motor. Hence, the unipolar stepper motors have 6/5 leads: 4 representing the four stator windings and 2 commons for the center tapped leads.

The stepper motor moves in fixed repeatable increment, which allows one to move it to a fixed precise position. The direction of rotation is dictated by the stator poles. The stator poles are determined by the current sent through the wire coils. As direction of current changed, the polarity is also changed causing motor rotate in opposite direction. As the sequence of power is applied to each stator winding, the rotor will rotate. There are several widely used sequences where each has a different degree of precision. The figure 5.2 shows 1-phase, 4-step stepping sequence. It is 1-phase because at any time only one coil is excited.

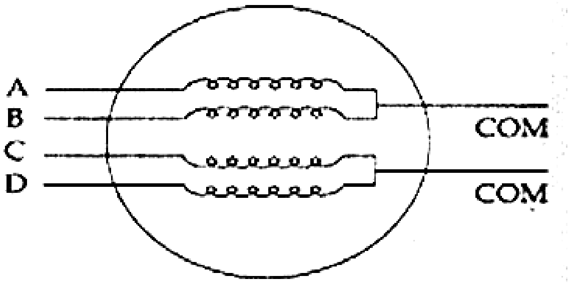
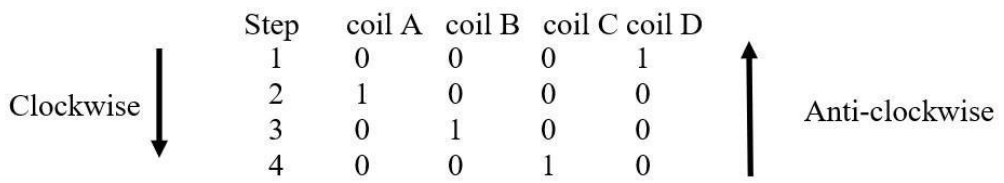


Figure 5.1: Unipolar stepper motor


Clockwise
Anti-clockwise

Step	coil A	coil B	coil C	coil D
1	0	0	0	1
2	1	0	0	0
3	0	1	0	0
4	0	0	1	0

Figure 5.2: Stepping sequence

Anyone of these values forms the initial value. The step angle is minimum degree of rotation associated with a single step. This depends on number teeth on rotor and stator. The step angle of motor using in lab is 1.8º. So, the steps per revolution is 200 (360^0 / 1.8^0). Note that, after completing every four steps, the rotor moves only by one tooth pitch. Therefore, in a stepper motor with 200 steps per revolution, the rotor has 50 teeth since 4 x 50=200 steps are needed to complete one revolution (refer figure 5.3). The curtain amount of minimum of current is required to energize the coils. Since, microcontroller lacks sufficient current to drive the coils of stepper motor, a driver IC ULN 2003AN (8 Darlington emitter pairs) is used. The pin layout, block diagram and circuit diagram of ULN 2003 is shown in figure 5.4. The connection of stepper motor to driver IC is shown in figure 5.5.

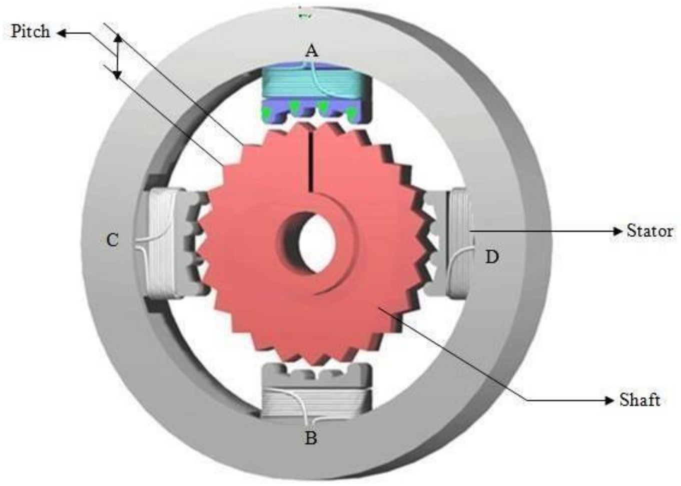
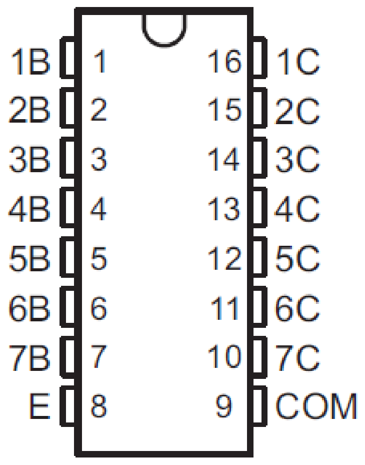
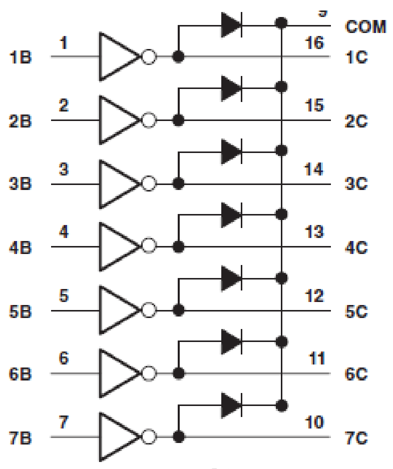


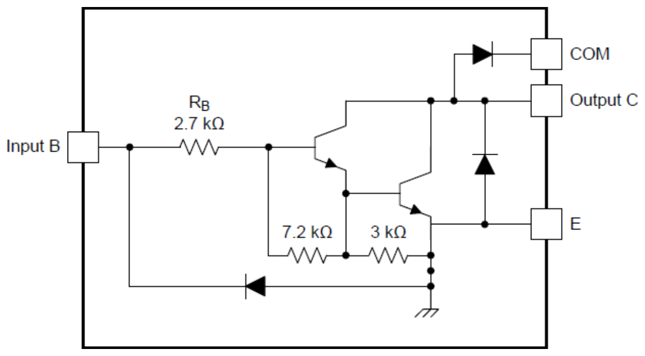
Figure 5.3: Stepper motor internal structure



(a)



(b)



(c)

Figure 5.4: ULN 2003AN (a) Pin diagram (b) Simplified block diagram (c)Circuit diagram

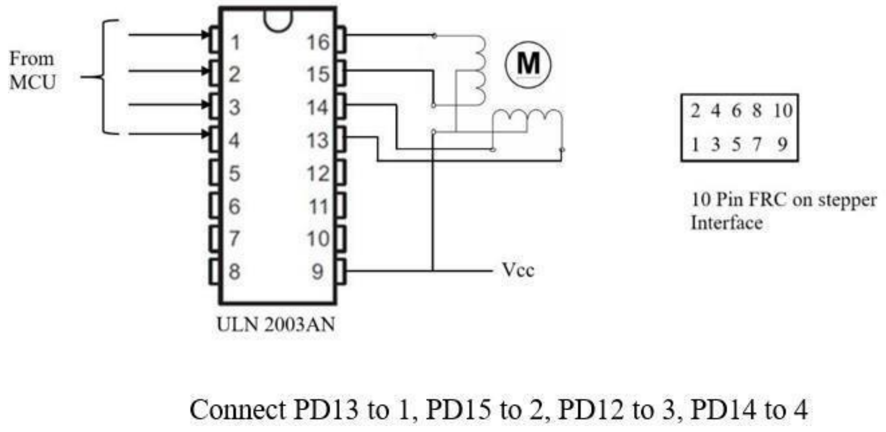
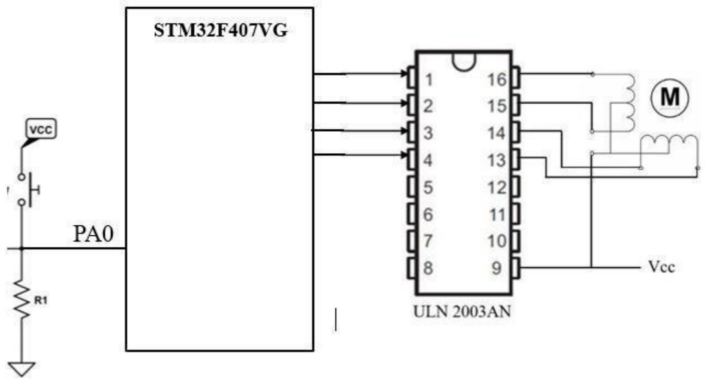

From MCU
1
2
3
4
5
6
7
8
16
15
14
13
12
11
10
9
M
Vcc
ULN 2003AN
2 4 6 8 10
1 3 5 7 9
10 Pin FRC on stepper Interface
Connect PD13 to 1, PD15 to 2, PD12 to 3, PD14 to 4

Figure 5.5: Connection of stepper motor to ULN 2003AN driver and Pin layout of 10 pin FRC.

a) Write a program to rotate a motor continuously both in clockwise and anticlockwise direction. Use a button connected to GPIO Pins on target board. The program should monitor button status and perform the following.
   - If the button is closed, the stepper motor moves in an anticlockwise direction.
   - If the button is opened, the stepper motor moves in a clockwise direction.

Interfacing diagram:



Follow the similar procedure in STM32CubeMX as mentioned in previous experiment and configure PD15, PD14, PD13 and PD12 as GPIO outputs.

Program:

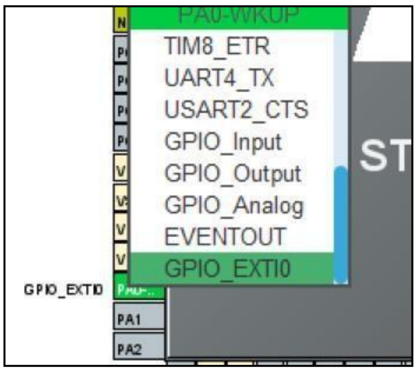
```
#include "main.h"

void Step_Sequence1(void);
void Step_Sequence2(void);
void Step_Sequence3(void);
void Step_Sequence4(void);

GPIO_PinState direction;
```

```
void SystemClock_Config(void);
static void MX_GPIO_Init(void);

int main(void)
{
  HAL_Init();
  SystemClock_Config();
  MX_GPIO_Init();

  while (1)
  {
                direction = HAL_GPIO_ReadPin(GPIOA, GPIO_PIN_0);

                if(direction == GPIO_PIN_SET)
                {
                        Step_Sequence1();
                        HAL_Delay(50);
                        Step_Sequence2();
                        HAL_Delay(50);
                        Step_Sequence3();
                        HAL_Delay(50);
                        Step_Sequence4();
                        HAL_Delay(50);
                }
                else
                {
                        Step_Sequence4();
                        HAL_Delay(50);
                        Step_Sequence3();
                        HAL_Delay(50);
                        Step_Sequence2();
                        HAL_Delay(50);
                        Step_Sequence1();
                        HAL_Delay(50);


                }
        }
}

void Step_Sequence1() //0x01 -pin 12
{
        HAL_GPIO_WritePin(GPIOD, GPIO_PIN_12, GPIO_PIN_SET);
        HAL_GPIO_WritePin(GPIOD, GPIO_PIN_13, GPIO_PIN_RESET);
        HAL_GPIO_WritePin(GPIOD, GPIO_PIN_14, GPIO_PIN_RESET);
        HAL_GPIO_WritePin(GPIOD, GPIO_PIN_15, GPIO_PIN_RESET);
}

void Step_Sequence2() //0x08 - Pin 15
{
        HAL_GPIO_WritePin(GPIOD, GPIO_PIN_12, GPIO_PIN_RESET);
        HAL_GPIO_WritePin(GPIOD, GPIO_PIN_13, GPIO_PIN_RESET);
        HAL_GPIO_WritePin(GPIOD, GPIO_PIN_14, GPIO_PIN_RESET);
        HAL_GPIO_WritePin(GPIOD, GPIO_PIN_15, GPIO_PIN_SET);
}

void Step_Sequence3() //0x04 - Pin 13
{
        HAL_GPIO_WritePin(GPIOD, GPIO_PIN_12, GPIO_PIN_RESET);
        HAL_GPIO_WritePin(GPIOD, GPIO_PIN_13, GPIO_PIN_SET);
        HAL_GPIO_WritePin(GPIOD, GPIO_PIN_14, GPIO_PIN_RESET);
        HAL_GPIO_WritePin(GPIOD, GPIO_PIN_15, GPIO_PIN_RESET);
}

void Step_Sequence4() //0x02 - Pin 14
{
        HAL_GPIO_WritePin(GPIOD, GPIO_PIN_12, GPIO_PIN_RESET);
        HAL_GPIO_WritePin(GPIOD, GPIO_PIN_13, GPIO_PIN_RESET);
        HAL_GPIO_WritePin(GPIOD, GPIO_PIN_14, GPIO_PIN_SET);
        HAL_GPIO_WritePin(GPIOD, GPIO_PIN_15, GPIO_PIN_RESET);
}
```

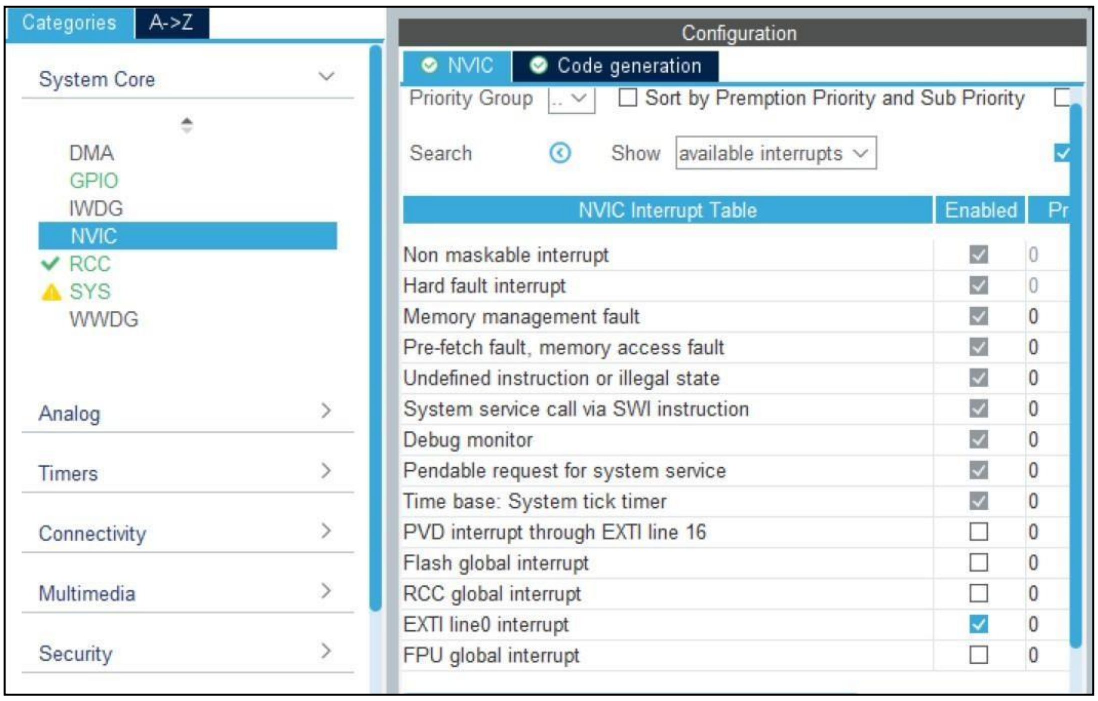
Alternate Program using Interrupts:
The push button connected to PA0 can be made to generate interrupt to change direction of stepper motor as shown in program below.

Procedure to follow in STM32CubeMx

1. Inthe STM32cubeMX, follow the similar procedure in STM32CubeMX as mentioned in previous experiment and configure PD15, PD14, PD13 and PD12 as GPIO outputs. In addition, select PA0 as GPIO_EXTI0 to enable interrupt on button press.



2. Under categories, select NVIC and enable by checking EXT1 line 0 interrupt as shown below.



Priority Group .. ☐ Sort by Premption Priority and Sub Priority

Search Show available interrupts

NVIC Interrupt Table	Enabled	Pr
Non maskable interrupt	✓	0
Hard fault interrupt	✓	0
Memory management fault	✓	0
Pre-fetch fault, memory access fault	✓	0
Undefined instruction or illegal state	✓	0
System service call via SWI instruction	✓	0
Debug monitor	✓	0
Pendable request for system service	✓	0
Time base: System tick timer	✓	0
PVD interrupt through EXTI line 16	☐	0
Flash global interrupt	☐	0
RCC global interrupt	☐	0
EXTI line0 interrupt	✓	0
FPU global interrupt	☐	0

3. Generate code by selecting the path and tool chain.

Program:

```
  #include "main.h"

  void  SystemClock_Config(void);
  static void MX_GPIO_Init(void);
  uint8_t flag=0;

 //Interrupt service routine to flip flag value
 void HAL_GPIO_EXTI_Callback(uint16_t GPIO_Pin)
  {
          /* Prevent unused argument(s) compilation warning */
          UNUSED(GPIO_Pin);

          flag=~flag;
   }
int main(void)
{
   HAL_Init();
   SystemClock_Config();

   MX_GPIO_Init();



          while (1)
          {
                    if(flag)
                    {
                        Step_Sequence1();
                        HAL_Delay(50);
                        Step_Sequence2();
                        HAL_Delay(50);
                        Step_Sequence3();
                        HAL_Delay(50);
                        Step_Sequence4();
                        HAL_Delay(50);
                }
                else
                {
                        Step_Sequence4();
                        HAL_Delay(50);
                        Step_Sequence3();
                        HAL_Delay(50);
                        Step_Sequence2();
                        HAL_Delay(50);
                        Step_Sequence1();
                        HAL_Delay(50);
                }
        }
}



void Step_Sequence1() //0x01 -pin 12
{
        HAL_GPIO_WritePin(GPIOD, GPIO_PIN_12, GPIO_PIN_SET);
        HAL_GPIO_WritePin(GPIOD, GPIO_PIN_13, GPIO_PIN_RESET);
        HAL_GPIO_WritePin(GPIOD, GPIO_PIN_14, GPIO_PIN_RESET);
        HAL_GPIO_WritePin(GPIOD, GPIO_PIN_15, GPIO_PIN_RESET);
}

void Step_Sequence2() //0x08 - Pin 15
{
        HAL_GPIO_WritePin(GPIOD, GPIO_PIN_12, GPIO_PIN_RESET);
        HAL_GPIO_WritePin(GPIOD, GPIO_PIN_13, GPIO_PIN_RESET);
        HAL_GPIO_WritePin(GPIOD, GPIO_PIN_14, GPIO_PIN_RESET);
        HAL_GPIO_WritePin(GPIOD, GPIO_PIN_15, GPIO_PIN_SET);
}


void Step_Sequence3() //0x04 - Pin 13
{
        HAL_GPIO_WritePin(GPIOD, GPIO_PIN_12, GPIO_PIN_RESET);
        HAL_GPIO_WritePin(GPIOD, GPIO_PIN_13, GPIO_PIN_SET);
        HAL_GPIO_WritePin(GPIOD, GPIO_PIN_14, GPIO_PIN_RESET);
        HAL_GPIO_WritePin(GPIOD, GPIO_PIN_15, GPIO_PIN_RESET);
}
```

```
void Step_Sequence4() //0x02 - Pin 14
```

```
{
        HAL_GPIO_WritePin(GPIOD, GPIO_PIN_12, GPIO_PIN_RESET);
        HAL_GPIO_WritePin(GPIOD, GPIO_PIN_13, GPIO_PIN_RESET);
        HAL_GPIO_WritePin(GPIOD, GPIO_PIN_14, GPIO_PIN_SET);
        HAL_GPIO_WritePin(GPIOD, GPIO_PIN_15, GPIO_PIN_RESET);
}
```

b) Write a program to rotate motor 180º in the clockwise direction. The motor step angle is 1.8º. Use the 4-step sequence.

The given data is,
Step angle=1.8º /step
Therefore, for 4-step the angle of rotation= 4x1.8 =7.2º.
For 180º, the 4-step sequence must be repeated for 25 times(180º / 7.2º)

Program:

```
#include "main.h"
void SystemClock_Config(void);
static void MX_GPIO_Init(void);
void Step_Sequence1(void);
void Step_Sequence2(void);
void Step_Sequence3(void);
void Step_Sequence4(void);
int main(void)
{
        HAL_Init();
        SystemClock_Config();
        MX_GPIO_Init();
        int i = 25;
    while(i>0)
        {
                        Step_Sequence1();
                        HAL_Delay(50);
                        Step_Sequence2();
                        HAL_Delay(50);
                        Step_Sequence3();
                        HAL_Delay(50);
                        Step_Sequence4();
                        HAL_Delay(50);
                        i--;
        }
        while (1);


void Step_Sequence1() //0x01 -pin 12
   {
```

```
HAL_GPIO_WritePin(GPIOD, GPIO_PIN_12, GPIO_PIN_RESET);
HAL_GPIO_WritePin(GPIOD, GPIO_PIN_12, GPIO_PIN_SET);
HAL_GPIO_WritePin(GPIOD, GPIO_PIN_13, GPIO_PIN_RESET);
HAL_GPIO_WritePin(GPIOD, GPIO_PIN_12, GPIO_PIN_RESET);
```

```
          HAL_GPIO_WritePin(GPIOD, GPIO_PIN_14, GPIO_PIN_RESET);
          HAL_GPIO_WritePin(GPIOD, GPIO_PIN_15, GPIO_PIN_RESET);
    }

void Step_Sequence2() //0x08 - Pin 15
   {
          HAL_GPIO_WritePin(GPIOD, GPIO_PIN_12, GPIO_PIN_RESET);
          HAL_GPIO_WritePin(GPIOD, GPIO_PIN_13, GPIO_PIN_RESET);
          HAL_GPIO_WritePin(GPIOD, GPIO_PIN_14, GPIO_PIN_RESET);
          HAL_GPIO_WritePin(GPIOD, GPIO_PIN_15, GPIO_PIN_SET);
    }

void Step_Sequence3() //0x04 - Pin 13
    {
          HAL_GPIO_WritePin(GPIOD, GPIO_PIN_12, GPIO_PIN_RESET);
          HAL_GPIO_WritePin(GPIOD, GPIO_PIN_13, GPIO_PIN_SET);
          HAL_GPIO_WritePin(GPIOD, GPIO_PIN_14, GPIO_PIN_RESET);
          HAL_GPIO_WritePin(GPIOD, GPIO_PIN_15, GPIO_PIN_RESET);
     }
void Step_Sequence4() //0x02 - Pin 14
    {
          HAL_GPIO_WritePin(GPIOD, GPIO_PIN_12, GPIO_PIN_RESET);
          HAL_GPIO_WritePin(GPIOD, GPIO_PIN_13, GPIO_PIN_RESET);
          HAL_GPIO_WritePin(GPIOD, GPIO_PIN_14, GPIO_PIN_SET);
          HAL_GPIO_WritePin(GPIOD, GPIO_PIN_15, GPIO_PIN_RESET);

     }
```

Space to answer viva questions:

Sl. No	**Criteria**	**Max Marks**	**Marks obtained**
Data sheet			
A	Analysis of program flow and instructions used/ Writing program	04	
B	Simulation/ Conduction of the experiment/ Analysis of Results	04	
C.	Viva	2	
	Total	10	

ANALOG TO DIGTAL CONVERTERS(ADC)

Experiment 06

The A/D conversion is a quantizing process whereby an analog signal is converted into equivalent binary word. An ADC has n-bit resolution where n is 8, 10, 12, 16 or even 24 bits. The higher resolution ADC provides smaller step size, where step size is the smallest change in input analog voltage, an ADC can detect. ADC is governed by conversion time which is defined as time taken to convert the analog input to digital value. The step size is also called as resolution some times. The step size is defined as ratio of a change in value of input voltage needed to change a digital output by 1 LSB. If the full-scale input voltage required to cause a digital output of all 1's is V_{IFS}, then resolution (step size) can be given as,

Resolution=V_{IFS} / 2^n -1 Where, n=no. of output bits

Quantization Error

The figure below shows transfer curve of 3-bit ADC. From the figure, it is observed that there is an unavoidable uncertainty about the exact value of V_{IN} for all digital output. This uncertainty is specified as quantization error. Its value is ± ½ LSB as shown in figure 6.1.

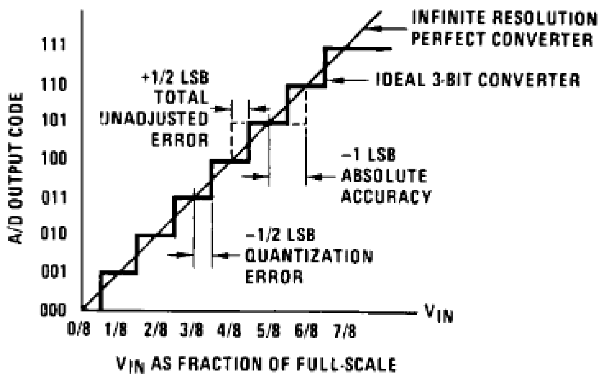


Figure 6.1: Input and output relation of ADC

ADC Module in STM32F407VG MCU:

The 12-bit ADC is a successive approximation analog-to-digital converter. It has up to 19 multiplexed channels allowing it to measure signals from 16 external sources, two internal sources, and the V_{BAT} channel as shown in figure 6.2. The user can select any channel for connecting analog input voltage.

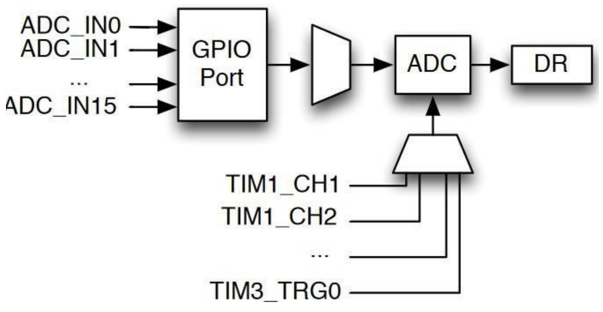


Figure 6.2: ADC module with different sources and Triggers

On the discovery board, PA0-PA7, PB0-PB1, and PC0-PC5 may all be used as analog inputs which can be multiplexed to the ADC. The ADC may be configured to sample any subset of these inputs in succession. There are two basic modes of operation: **single and continuous conversion**. With single conversion, once the ADC is triggered, it converts a single input and stores the result in its data register (DR). The trigger may either come from software or signal such as a timer. In continuous mode, the ADC starts another conversion as soon as it finishes one. The ADC may also operate in **scan mode** where a set of inputs to be scanned is configured. A single conversion is performed for each configured input in succession. Scans may also be continuous in the sense that a new scan begins as soon as one is completed. The digital data in data registered can be left aligned or right aligned.

Clock Configuration:

The ADC clock(ADCCLK) is generated from the APB2 peripheral clock2(PCLK2) divided by a programmable prescaler that allows the ADC to work at PCLK2 /2, /4, /6 or /8(default is 2).

The ADC samples the input voltage for a few of ADCCLK cycles that can be configured by user.

The total conversion time(for 12 bit resolution) is calculated as follows:

$$T_{conv} = \text{Sampling time} + 12 \text{ cycles}$$

The figure 6.3 shows timing diagram of ADC conversion.

Example:

With ADCCLK = 30 MHz and sampling time = 3 cycles:

$$T_{conv} = 3 + 12 = 15 \text{ cycles} = 0.5\ \mu s \text{ with PCLK2 at 60 MHz}$$

Triggering ADC:

Analog to digital conversion can be triggered by external events (E.g.: timer capture, EXTIline) or set by software by setting **SWSTART** bit of ADC control register 2. The end of conversion (EOC) is indicated by setting a **EOC** flag in ADC status register by hardware. This must be cleared by software or data read from data register.

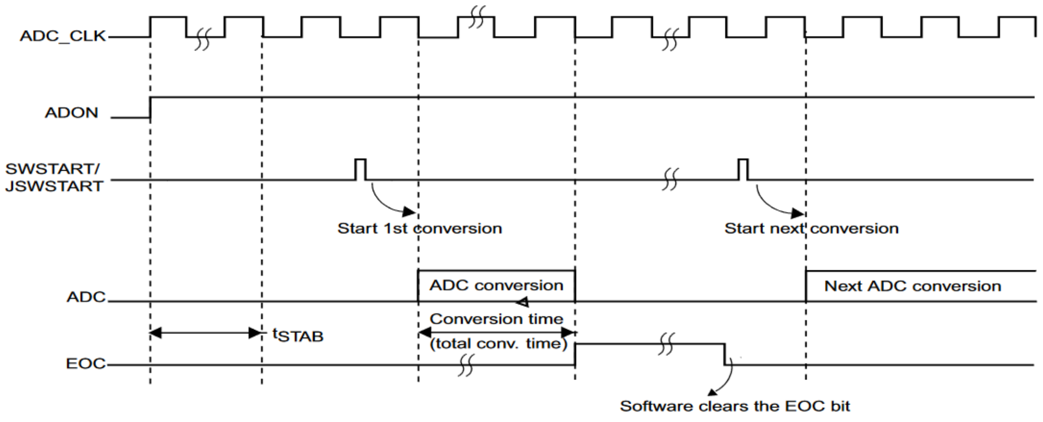
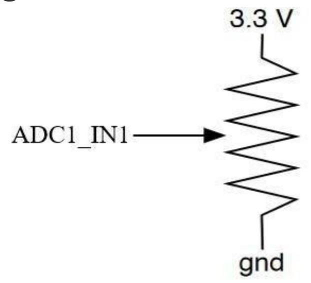
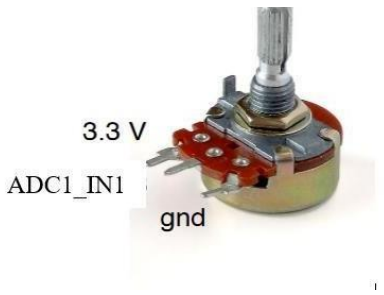


Figure 6.3: Timing diagram

a) Use STM32CubeMX to configure ADC1 of STM32F407VG MCU to following parameters:
Channel: IN1(PA1), PCLK2:60 MHz, ADCCLK:30MHz, Sampling time:3 ADCCLKs Resolution:12 bits, Triggering: S/W controlled, Analog input range(dynamic range):0 V to 3.3 V(DC)
Write application code in Keil IDE to use ADC to analog to digital conversion and display equivalent digital value in debug window.

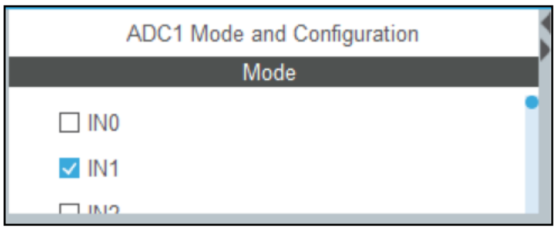
Connection diagram:

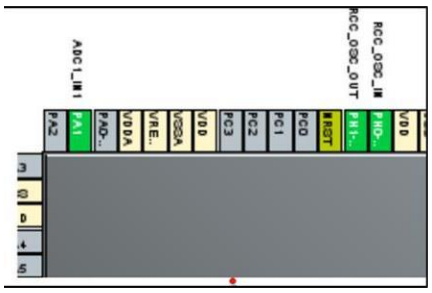




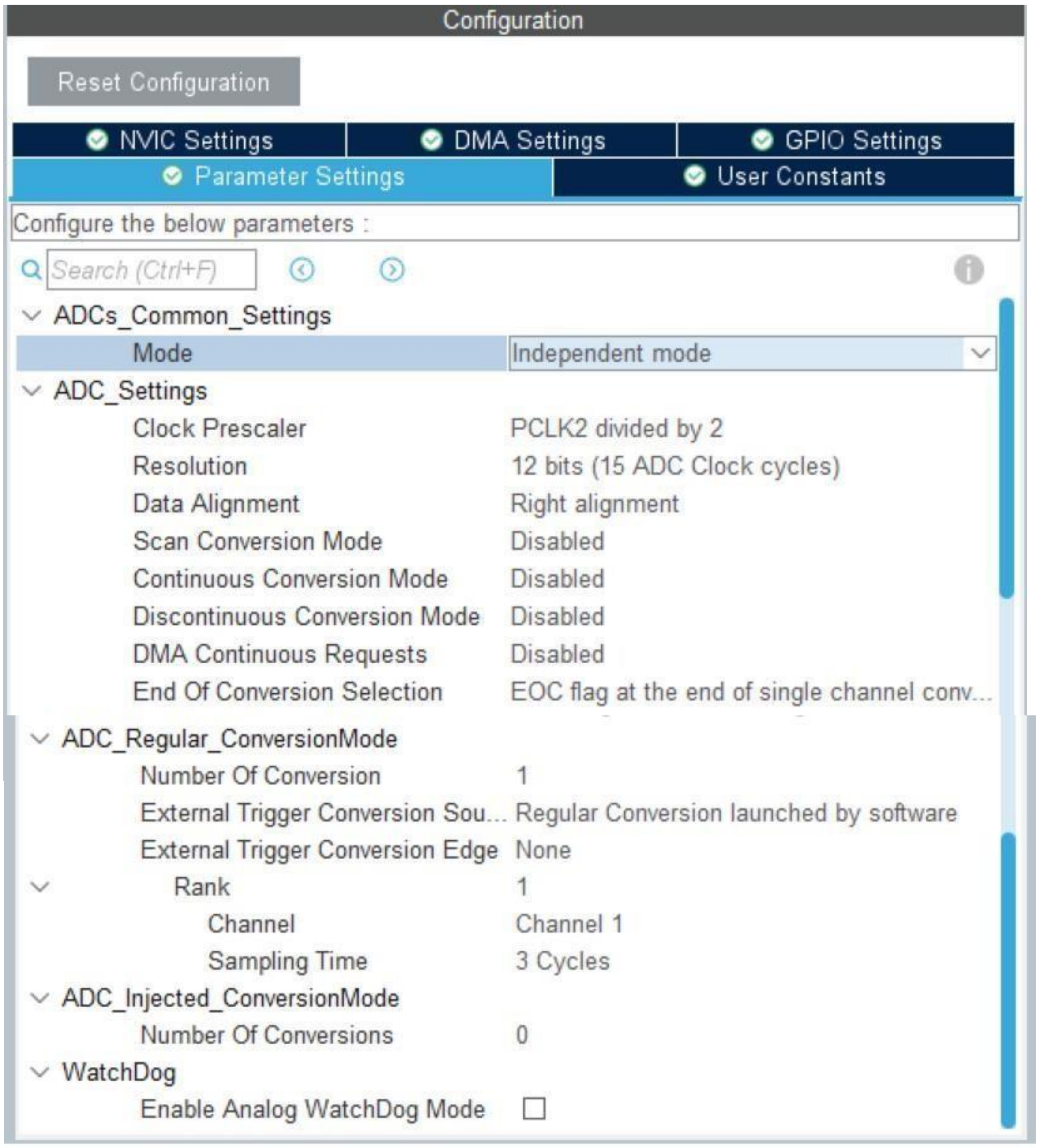
Procedure in STM32CubeMX to create project:

1. Open the STM32CubeMx software and click on Access to board selector under new project. Select board as STM32F407G-DISC1.
2. Select board as STM32F407G-DISC1. Click on start project. Select yes for initialize all peripherals in default mode.
3. In STM32CubeMx window, in pinout and configuration tab, select pinout drop down list and click on clear pinout configuration.
4. Under System Core, select the Reset and clock control(RCC) and for high speed clock(HSE) select Crystal/Ceramic Resonator. Select SYS and for debug select serial wire.
5. In clock configuration, select HSI as main clock source. Choose proper values for PLL pre (M and P) and post(N) scalers and APB2 prescaler to get APB2 peripheral clock as 60 MHz.
6. Select Analog under categories and select ADC1. In ADC 1 mode, select IN1 which corresponds to PA1 pin of the MCU. After this selection, PA1 pin on layout will be highlighted as ADC1_IN1 to which analog signal can be interface.


ADC1 Mode and Configuration
Mode
☐ IN0
☑ IN1



7. Under parameter setting of ADC 1 configuration, select the parameter values as shown below.



ADCs_Common_Settings
Mode: Independent mode
ADC_Settings
Clock Prescaler: PCLK2 divided by 2
Resolution: 12 bits (15 ADC Clock cycles)
Data Alignment: Right alignment
Scan Conversion Mode: Disabled
Continuous Conversion Mode: Disabled
Discontinuous Conversion Mode: Disabled
DMA Continuous Requests: Disabled
End Of Conversion Selection: EOC flag at the end of single channel conv...
ADC_Regular_ConversionMode
Number Of Conversion: 1
External Trigger Conversion Sou...: Regular Conversion launched by software
External Trigger Conversion Edge: None
Rank: 1
Channel: Channel 1
Sampling Time: 3 Cycles
ADC_Injected_ConversionMode
Number Of Conversions: 0
WatchDog
Enable Analog WatchDog Mode: ☐

8. Select Project manger tab. Enter project name, project location. Select tool chain IDE as MDK-ARM and let the other options at default values.
9. Click on Generate code. This option will generate project file in Keil and initial configuration files in the location specified.

Program:
In the Keil IDE, follow the process as indicated in the previous experiments.

```
#include "main.h"

ADC_HandleTypeDef hadc1; // handler for ADC

void  SystemClock_Config(void);
static void MX_GPIO_Init(void);
static void MX_ADC1_Init(void);
```

```
uint32_t analogValue;    // 32- bit variable for ADC value

int main(void)
{
  HAL_Init();

  SystemClock_Config();

  /* Initialize all configured peripherals */
  MX_GPIO_Init();
  MX_ADC1_Init();

              while (1)
              {

                     // Start ADC Conversion
                     HAL_ADC_Start(&hadc1);
                     // Poll ADC1 Peripherals & TimeOut = 1mSec
                     if(HAL_ADC_PollForConversion(&hadc1, 1)==HAL_OK)
                            analogValue = HAL_ADC_GetValue(&hadc1);
                     else
                            analogValue=0;
              }



    }
```

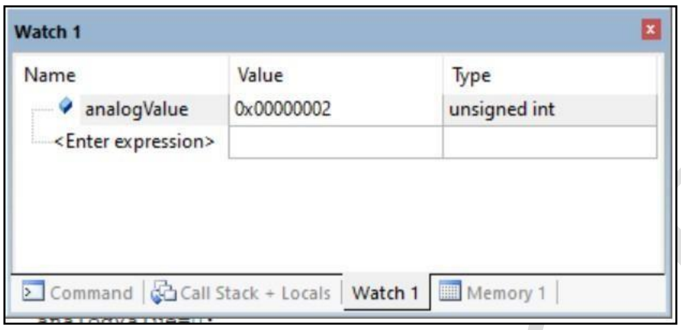
HAL APIs of ADC used in code:
HAL_StatusTypeDef HAL_ADC_Start(ADC_HandleTypeDef hadc)*
This Enables ADC and starts conversion of the regular channels.
Parameters: hadc pointer to a ADC_HandleTypeDef structure that contains the configuration information for the specified ADC.
Return type: HAL status

HAL_StatusTypeDef HAL_ADC_PollForConversion(ADC_HandleTypeDef hadc, uint32_t Timeout)*
Poll for regular conversion complete. ADC conversion flags EOS (end of sequence) and EOC (end of conversion) are cleared by this function.
Parameters: **hadc** pointer to a ADC_HandleTypeDef structure that contains the configuration information for the specified ADC.
Timeout Timeout value in millisecond.
Return type: HAL status

uint32_t HAL_ADC_GetValue(ADC_HandleTypeDef hadc)*
Gets the converted value from data register of regular channel.

Procedure to be followed in Keil IDE:

1. After editing and successful build, to run program select Start/Stop debug session under debug tab.
2. Open View->Watch windows->Watch1. Add analogValue variable to watch window. Run the program by selecting run option under debug tab. Connect PA1 to analog source between 0 to 3.3 V. In watch window, digital equivalent can be found and validate the same by taking multiple values.


Watch 1

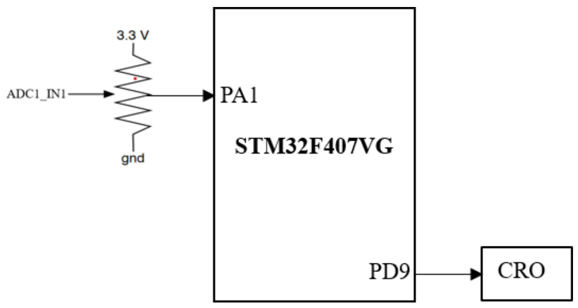
Name	Value	Type
analogValue	0x00000002	unsigned int
<Enter expression>		

3. Tabulate the observations:

Voltage values	Digital equivalent	
	Practical	Theoretical

b) Use STM32CubeMX to configure ADC1 of STM32F407VG MCU to following parameters:
Channel: IN1(PA1), PCLK2:60 MHz, ADCCLK:30MHz, Sampling time:3 ADCCLKs
Resolution:12 bits, Triggering: S/W controlled, Analog input range(dynamic range):0 V to 3.3 V(DC)
Write application code in Keil IDE to use ADC to analog to digital conversion and generate PWM wave on GPIO pin PD9 in accordance to analog input voltage.

Interfacing diagram:



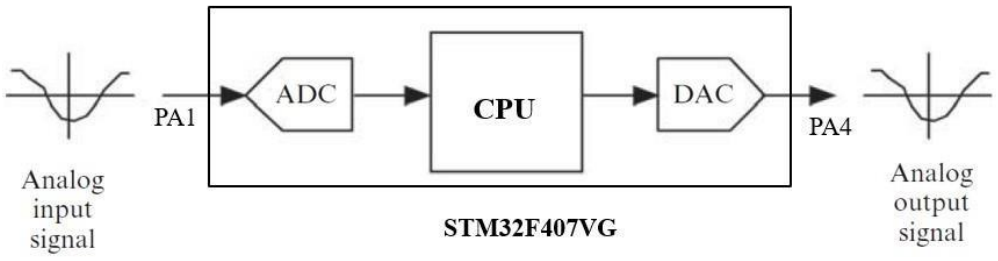
Procedure:

- Follow the similar procedure of previous program to configure the ADC. In addition, configure a port pin PD9 as GPIO output.

Program:

```
#include "main.h"
/* Private variables..........................................................................*/
ADC_HandleTypeDef hadc1;

/* Private function prototypes_________________________________*/
void SystemClock_Config(void);
static void MX_GPIO_Init(void);
static void MX_ADC1_Init(void);

uint16_t analogValue,i;
int main(void)
{
  HAL_Init();

 SystemClock_Config();

  MX_GPIO_Init();
  MX_ADC1_Init();


    while (1)
  {
                HAL_ADC_Start(&hadc1);
                // Poll ADC1 Peripherals & TimeOut = 1mSec
                if(HAL_ADC_PollForConversion(&hadc1, 1)==HAL_OK)
                        analogValue = HAL_ADC_GetValue(&hadc1);
                else
                        analogValue=0;

                HAL_GPIO_WritePin(GPIOD,GPIO_PIN_9,GPIO_PIN_SET);
                for(i=0;i<analogValue;i++);
                for(i=0;i<analogValue;i++);

                analogValue=(~analogValue)&0x0FFF; // Masking upper four bits.
                HAL_GPIO_WritePin(GPIOD,GPIO_PIN_9,GPIO_PIN_RESET);
                for(i=0;i<analogValue;i++);
                for(i=0;i<analogValue;i++);
    }
  }
```

c) Use STM32CubeMX to configure ADC1 and DAC of STM32F407VG MCU to realize following processing chain:



ADC Channel: IN1(PA1), PCLK2:60 MHz, ADCCLK:30MHz, Resolution:12 bits, Triggering: S/W controlled

DAC channel: OUT 1(PA4), $t_{SETTLING}$=3 µs (Digital to analog conversion time)

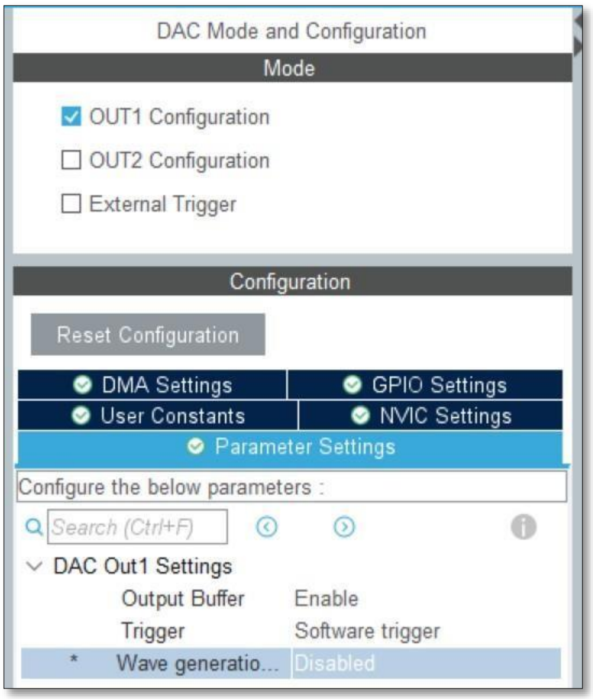
Set the sampling rate of ADC to match the settling time of DAC.

Input: sine signal of 2 V_{PP} with 1V DC offset and frequency: 1 Hz to 50 KHz in steps 3 KHz

Write application code in Keil IDE to use ADC for analog to digital conversion and DAC to reconstruct signal.

Procedure in STM32CubeMX:

1. Follow the similar procedure of previous program to configure the ADC. Set the sampling rate of ADC to match the settling time of DAC.
2. Configure DAC under Analog option for the following options and generate code.



☑ OUT1 Configuration
☐ OUT2 Configuration
☐ External Trigger

DAC Out1 Settings

Output Buffer	Enable
Trigger	Software trigger
* Wave generatio...	Disabled

3. Tabulate the observations in the following table.

Input sine frequency	Output signal shape And frequency	Remarks
1 KHz		
2 KHz		
3 KHz		
4 KHz		
5 KHz		
8 KHz		
12 KHz		
17 KHz		
25 KHz		
30 KHz		
35 KHz		
40 KHz		
45 KHz		
50 KHz		

Program:

```
#include "main.h"
ADC_HandleTypeDef hadc1;

DAC_HandleTypeDef hdac;

void  SystemClock_Config(void);
static  void  MX_GPIO_Init(void);
static void MX_ADC1_Init(void);
static void MX_DAC_Init(void);

uint16_t analogValue;
int main(void)
{
  /* Reset of all peripherals, Initializes the Flash interface and the Systick. */
  HAL_Init();

  /* Configure the system clock */
  SystemClock_Config();

  /* Initialize all configured peripherals */
  MX_GPIO_Init();
  MX_ADC1_Init();
  MX_DAC_Init();

  while (1)
  {
        HAL_ADC_Start(&hadc1);

                if(HAL_ADC_PollForConversion(&hadc1, 1)==HAL_OK)
                        analogValue = HAL_ADC_GetValue(&hadc1);
                else
                        analogValue=0;

        HAL_DAC_SetValue(&hdac,DAC_CHANNEL_1,DAC_ALIGN_12B_R,analogValue);
        HAL_DAC_Start(&hdac,DAC_CHANNEL_1);
  }
}
```

HAL APIs of DAC used in code:

*HAL_StatusTypeDef HAL_DAC_SetValue(DAC_HandleTypeDef *hdac, uint32_t Channel, uint32_t Alignment, uint32_t Data)*

Set the specified data holding register value for DAC channel.

hdac pointer to a DAC_HandleTypeDef structure that contains the configuration information for the specified DAC.

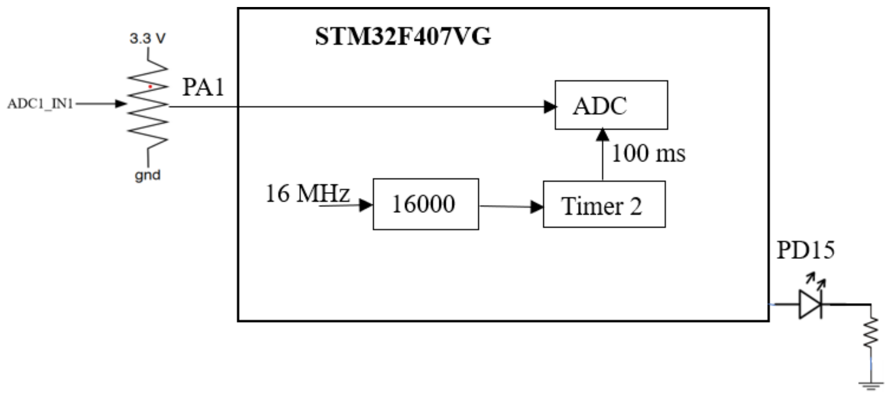
Channel The selected DAC channel: DAC_CHANNEL_1: DAC Channel1 selected
Alignment Specifies the data alignment: DAC_ALIGN_12B_R: 12bit right data alignment selected
Data to be loaded in the selected data holding register.
HAL status return value

*HAL_StatusTypeDef HAL_DAC_Start(DAC_HandleTypeDef *hdac, uint32_t Channel)*
Enables DAC and starts conversion of channel.

d) Use STM32CubeMX to configure ADC1 to sample data on IN1 channel with following specifications.

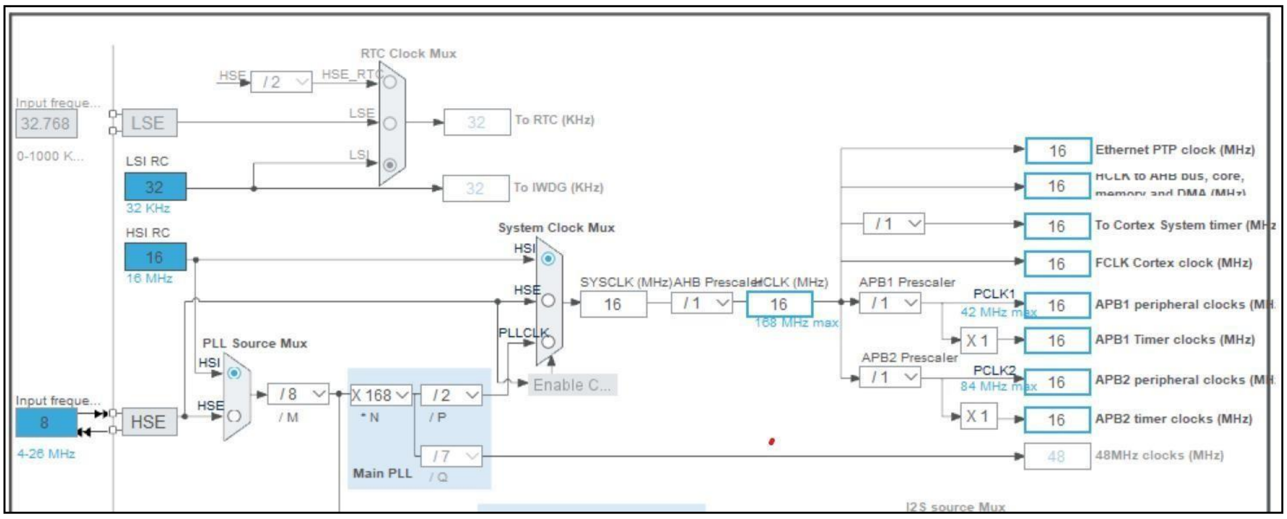
Channel: IN1(PA1), PCLK2:16 MHz, ADCCLK: 8MHz, Sampling time:28 ADCCLKs
Resolution:8 bits, Triggering: Timer 2 trigger out event (at 100 msec), Analog input range(dynamic range):0 V to 3.3 V(DC)
Enable interrupts for ADC on conversion completion to read digital value and toggle LED on PD15 in interrupt service routine.
Write application code in Keil IDE to use ADC to analog to digital conversion triggered by timer 2 at 100 ms. The ADC output value must be validated in debug window andinterrupt rate in CRO.

Interfacing Diagram:

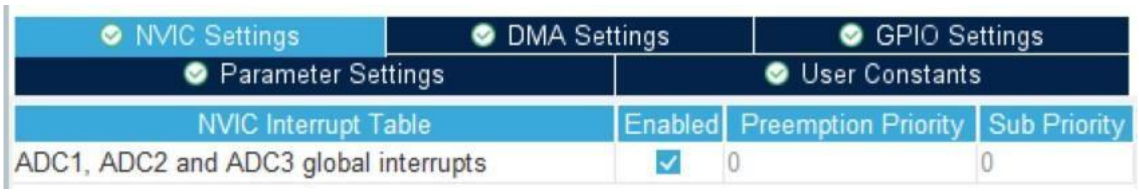


Procedure:

1. Open the STM32CubeMx software and click on Access to board selector under new project. Select board as STM32F407G-DISC1.
2. Select board as STM32F407G-DISC1. Click on start project. Select yes for initialize all peripherals in default mode.
3. In STM32CubeMx window, in pinout and configuration tab, select pinout drop down list and click on clear pinout configuration.
4. Under System Core, select the Reset and clock control(RCC) and for high speed clock(HSE) select Crystal/Ceramic Resonator. Select SYS and for debug select serial wire.
5. In clock configuration, select options as below:



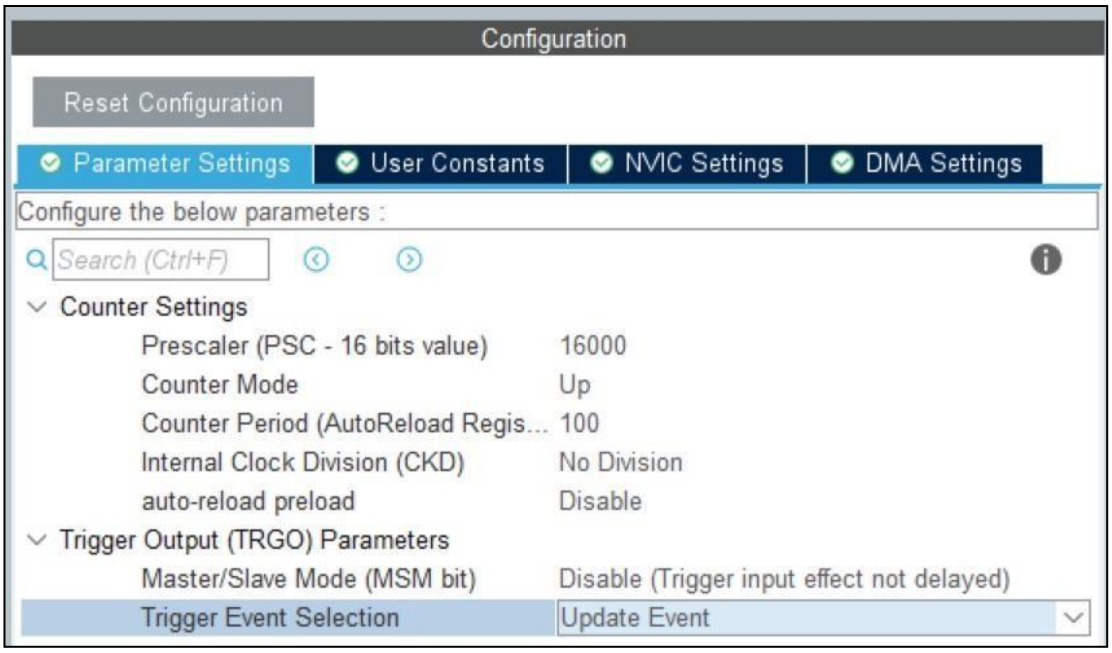
6. Select Analog under categories and select ADC1. In ADC 1 mode, select IN1 which corresponds to PA1 pin of the MCU. Select other parameters as follows:
   Sampling time:28 ADCCLKs Resolution:8 bits, Triggering: Timer 2 trigger out event (at 100 msec)
7. Under NVIC settings, enable interrupt on channel 1 of ADC 1 as follows:



NVIC Settings | DMA Settings | GPIO Settings
Parameter Settings | User Constants

NVIC Interrupt Table	Enabled	Preemption Priority	Sub Priority
ADC1, ADC2 and ADC3 global interrupts	✓	0	0

8. Under Timers, select timer 2 and set the parameters as follows:



Configuration

Reset Configuration

Parameter Settings | User Constants | NVIC Settings | DMA Settings

Configure the below parameters :

Search (Ctrl+F)

Counter Settings	
Prescaler (PSC - 16 bits value)	16000
Counter Mode	Up
Counter Period (AutoReload Regis...	100
Internal Clock Division (CKD)	No Division
auto-reload preload	Disable
Trigger Output (TRGO) Parameters	
Master/Slave Mode (MSM bit)	Disable (Trigger input effect not delayed)
Trigger Event Selection	Update Event

9. Configure PD15 as GPIO output for LED toggling.
10. Select Project manger tab. Enter project name, project location. Select tool chain IDE as MDK-ARM and let the other options at default values.
11. Click on Generate code. This option will generate project file in Keil and initial configuration files in the location specified.

Program:

```
#include "main.h"

ADC_HandleTypeDef hadc1;
TIM_HandleTypeDef htim2;

uint8_t analogValue;

void  SystemClock_Config(void);
static  void  MX_GPIO_Init(void);
static void MX_ADC1_Init(void);
static void MX_TIM2_Init(void);

//ADC interrupt service routine
void HAL_ADC_ConvCpltCallback(ADC_HandleTypeDef* hadc)
{
	analogValue=HAL_ADC_GetValue(&hadc1);
	HAL_GPIO_TogglePin(GPIOD,GPIO_PIN_15);
}

int main(void)
{
  HAL_Init();
  SystemClock_Config();
  MX_GPIO_Init();
  MX_ADC1_Init();
  MX_TIM2_Init();

  HAL_TIM_Base_Start(&htim2);
  HAL_ADC_Start_IT(&hadc1);// Start ADC in interrupt mode

  while (1)
   {
     //Nothing
   }

 }
```

Procedure to be followed in Keil IDE:

1. After editing and successful build, to run program select Start/Stop debug session under debug tab.
2. Open View->Watch windows->Watch1. Add analogValue variable to watch window. Run the program by selecting run option under debug tab. Connect PA1 to analog source between

0 to 3.3 V. In watch window, digital equivalent can be found and validate the same by taking multiple values.

3. Tabulate the observations:

Voltage values	**Digital equivalent**	
	Practical	**Theoretical**

4. To validate the interrupt rate, connect CRO to PD15 and measure the toggling rate.

Space to answer viva questions:

Sl. No	Criteria	Max Marks	Marks obtained
Data sheet			
A	Analysis of program flow and instructions used/ Writing program	04	
B	Simulation/ Conduction of the experiment/ Analysis of Results	04	
C.	Viva	2	
	Total	10	

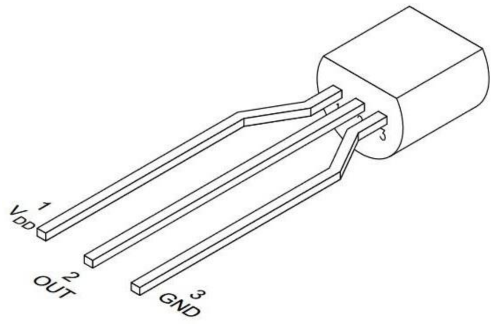
INTERFACING ANALOG TEMPERATURE SENSOR & PROGRAMMING USART FOR DISPLAY

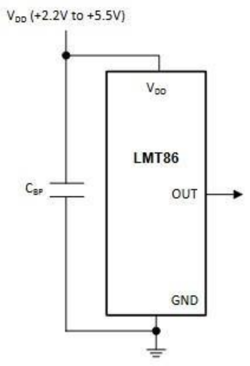
Experiment 07

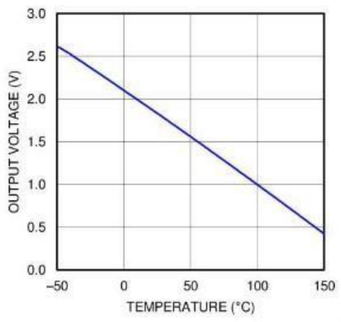
Design a suitable to scheme to interface analog temperature sensor LMT86 to STM32F407VG MCU and program USART2 to display digital value.

LMT86:

The LMT86 are precision CMOS temperature sensors with ±0.4°C typical accuracy (±2.7°C maximum) and a linear analog output voltage that is inversely proportional to temperature. The pin layout and typical characteristics are shown below.







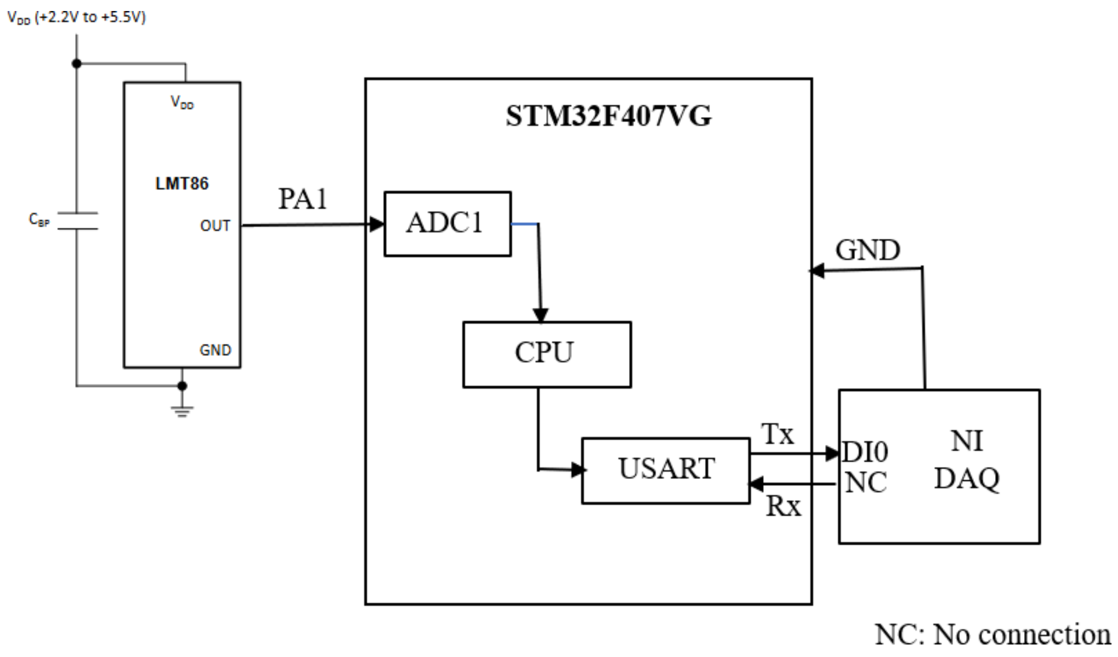
The table below shows LMT86 transfer functions.

TEMP (°C)	V_{OUT} (mV)	TEMP (°C)	V_{OUT} (mV)	TEMP (°C)	V_{OUT} (mV)	TEMP (°C)	V_{OUT} (mV)	TEMP (°C)	V_{OUT} (mV)
-50	2616	-10	2207	30	1777	70	1335	110	883
-49	2607	-9	2197	31	1766	71	1324	111	872
-48	2598	-8	2186	32	1756	72	1313	112	860
-47	2589	-7	2175	33	1745	73	1301	113	849
-46	2580	-6	2164	34	1734	74	1290	114	837
-45	2571	-5	2154	35	1723	75	1279	115	826
-44	2562	-4	2143	36	1712	76	1268	116	814
-43	2553	-3	2132	37	1701	77	1257	117	803
-42	2543	-2	2122	38	1690	78	1245	118	791
-41	2533	-1	2111	39	1679	79	1234	119	780
-40	2522	0	2100	40	1668	80	1223	120	769
-39	2512	1	2089	41	1657	81	1212	121	757
-38	2501	2	2079	42	1646	82	1201	122	745
-37	2491	3	2068	43	1635	83	1189	123	734

ADC Configuration:

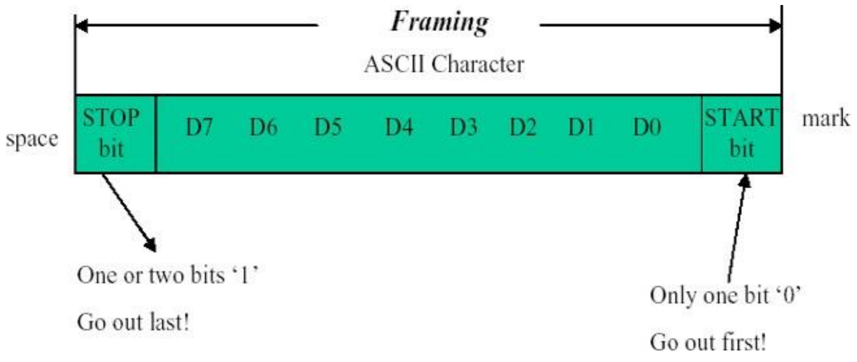
Use STM32CubeMX to configure ADC1 of STM32F407VG MCU to following parameters:
Channel: IN1(PA1), PCLK2:60 MHz, ADCCLK:30MHz, Sampling time:3 ADCCLKs
Resolution:12 bits, Triggering: S/W controlled.
Connect OUT pin of LMT86 to PA1(ADC channel IN1).

Interfacing Diagram:



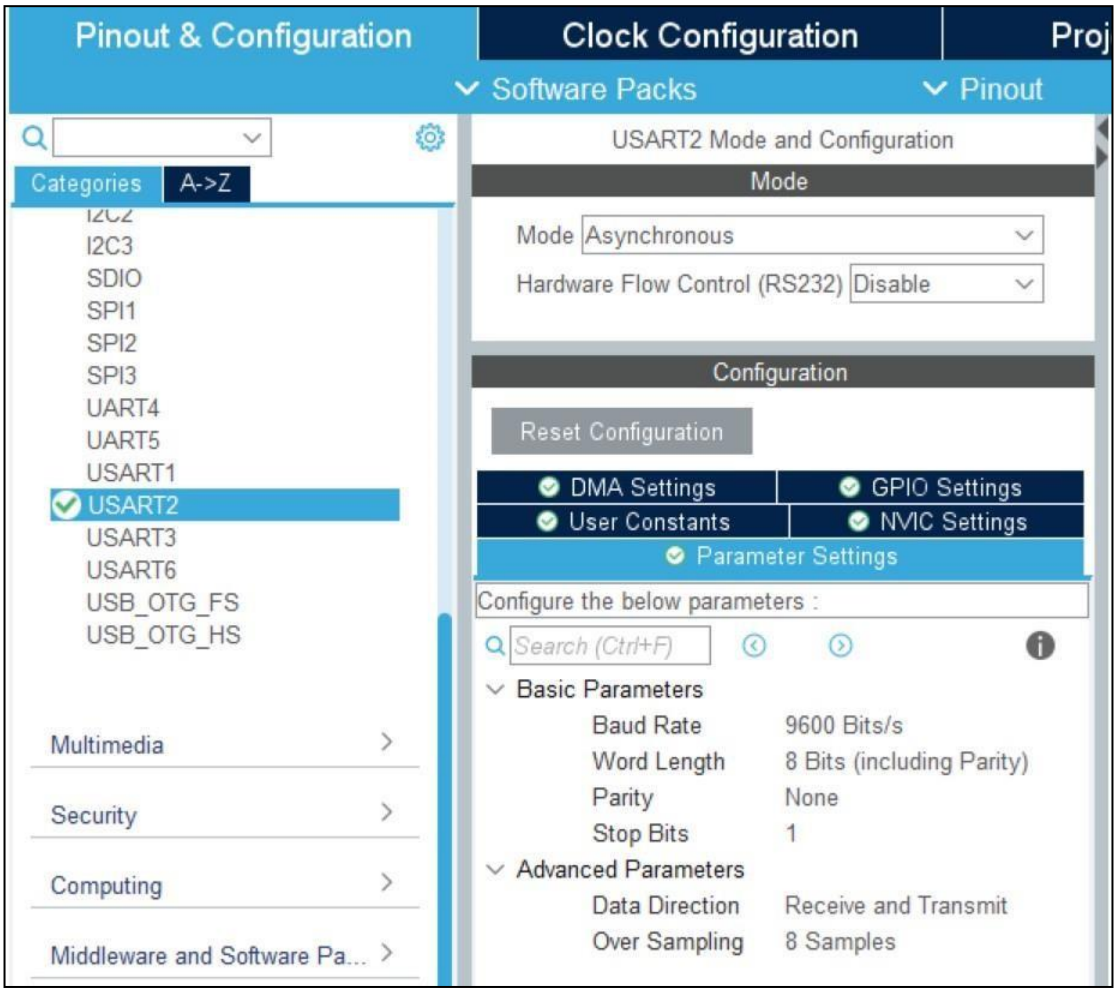
USART:

The universal synchronous asynchronous receiver transmitter (USART) offers a flexible means of full-duplex data exchange with external equipment. The USART can be configured to transfer data asynchronously with framing as shown below. The UART is characterized byTx and Rx for bidirectional data transfer. This serial port is characterized by baud rate which determine rate at which data transfer is carried on Tx and Rx lines.



Procedure in STM32CubeMX

1. Follow the similar procedure of previous experiment program (a) to configure the ADC.
2. Under connectivity, select USART 2 and set the parameters as shown in figure below.



USART2 Mode and Configuration

Mode

Mode Asynchronous

Hardware Flow Control (RS232) Disable

Configuration

Configure the below parameters :

Basic Parameters

Baud Rate	9600 Bits/s
Word Length	8 Bits (including Parity)
Parity	None
Stop Bits	1

Advanced Parameters

Data Direction	Receive and Transmit
Over Sampling	8 Samples

3. Generate the code with tool chain as MDK ARM.

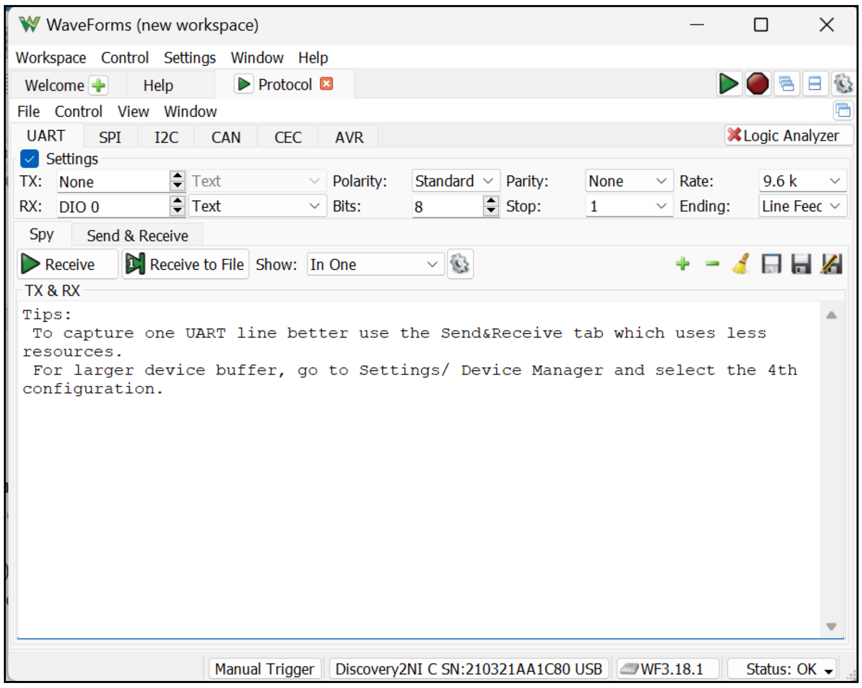
Program:

```
#include "main.h"
#include <string.h>
#include<stdio.h>


ADC_HandleTypeDef hadc1;
UART_HandleTypeDef huart2;

/* Private function prototypes-----------------------------------------------*/
void SystemClock_Config(void);
static void MX_GPIO_Init(void);
static void MX_ADC1_Init(void);
static void MX_USART2_UART_Init(void);
uint32_t analogValue;
```

```
 int main(void)
 {
    const char Array[]="\nTemperature ";

    char tempValue[20],i=0;

   HAL_Init();


   /* Configure the system clock */
   SystemClock_Config();

   MX_GPIO_Init();
   MX_ADC1_Init();
   MX_USART2_UART_Init();
   /* USER CODE BEGIN WHILE */
   while (1)
   {
                HAL_ADC_Start(&hadc1);
                // Poll ADC1 Peripherals & TimeOut = 1mSec
                 if(HAL_ADC_PollForConversion(&hadc1, 1)==HAL_OK)
                                analogValue = HAL_ADC_GetValue(&hadc1);
                else
                                analogValue=0;


                HAL_UART_Transmit(&huart2,(uint8_t *)Array,strlen(Array),10);
                i=0;
               while(analogValue!=0){
                                tempValue[i]=analogValue%10+'0';
                                analogValue=analogValue/10;
                                 i++;
                                          }
                  tempValue[i]='\0';
    HAL_UART_Transmit(&huart2,(uint8_t *)tempValue,strlen(tempValue),10);
     HAL_Delay(1000);

   }
}
```

Procedure to run program:

- After successful build and load the executable file to microcontroller board.
- Make the connections as shown in interfacing diagram and Tx(PA2) to Digital input(DI0) of the NI DAQ. Connect DAQ GND to microcontroller board GND.
- Open waveforms software and select discovery 2NI. Select protocol and UART. Select the configuration parameters as follows.



Settings
TX: None Text
RX: DIO 0 Text
Polarity: Standard
Bits: 8
Parity: None
Stop: 1
Rate: 9.6 k
Ending: Line Feec
Show: In One

```
Tips:
 To capture one UART line better use the Send&Receive tab which uses less
resources.
 For larger device buffer, go to Settings/ Device Manager and select the 4th
configuration.
```

- Click on Receive. The string Temperature followed by digital equivalent of temperature will be displayed.

Space to answer viva questions:

Sl. No	**Criteria**	**Max Marks**	**Marks obtained**
Data sheet			
A	Analysis of program flow and instructions used/ Writing program	04	
B	Simulation/ Conduction of the experiment/ Analysis of Results	04	
C.	Viva	2	
	Total	10	

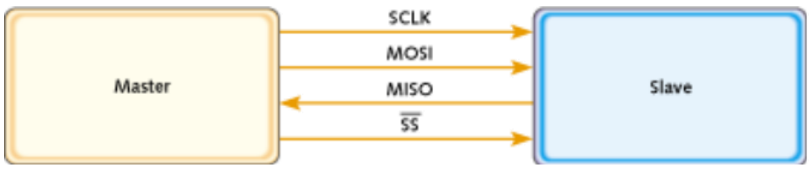
PROGRAM SPI & SHOW DATA TRANSFER BETWEEN SPI SLAVE DEVICE &MASTER

Experiment 08

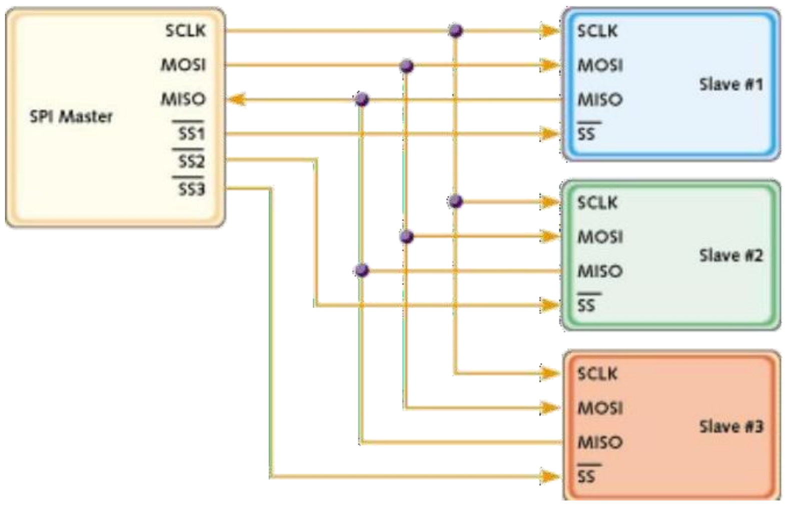
Serial Peripheral Interface (SPI): Features

- It is a synchronous serial data link that operates in full duplex
- A serial clock line synchronizes the shifting and sampling of the information on two serial data lines.
- The SPI is mainly used to allow a microcontroller to communicate with peripheral devices such as EEPROMs.
- SPI devices communicate using a master-slave relationship**.**
- **SPI signals:**
  - Clock: SCLK
  - Master Data Output, Slave Data Input: MOSI
  - Master Data Input, Slave Data Output: MISO
  - Slave Select: SS

- **Single master single slave**

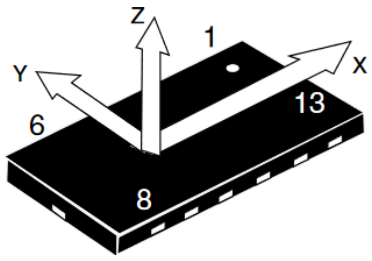


- **Single master multiple slaves**



LIS302DL 3-axis Accelerometer

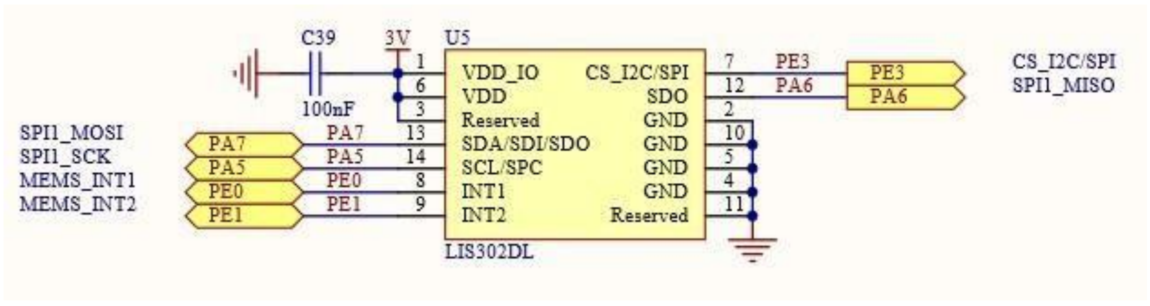
The LIS302DL is an ultra-compact low-power three axes linear accelerometer. It includes a sensing element and an IC interface able to provide the measured acceleration to the external world through I2C/SPI serial interface. The LIS302DL has dynamically user selectable full scales of ± 2g/± 8g.



The LIS302DL SPI is a bus slave. The SPI allows to write and read the registers of the device. The Serial Interface interacts with the outside world with 4 wires: CS, SPC, SDI and SDO. CS is the Serial Port Enable and it is controlled by the SPI master. It goes low at the start of the transmission and goes back high at the end. SPC is the Serial Port Clock and it is controlled by the SPI master. It is stopped high when CS is high (no transmission). SDI and SDO are respectively the Serial Port Data Input and Output. Those lines are driven at the falling edge of SPC and should be captured at the rising edge of SPC.
(More details can be found in datasheet)

a) Write a program to configure SPI of STM32F407VGT6x in full duplex master mode. Configure LIS302DL 3-axis accelerometer as slave to transfer data to master.

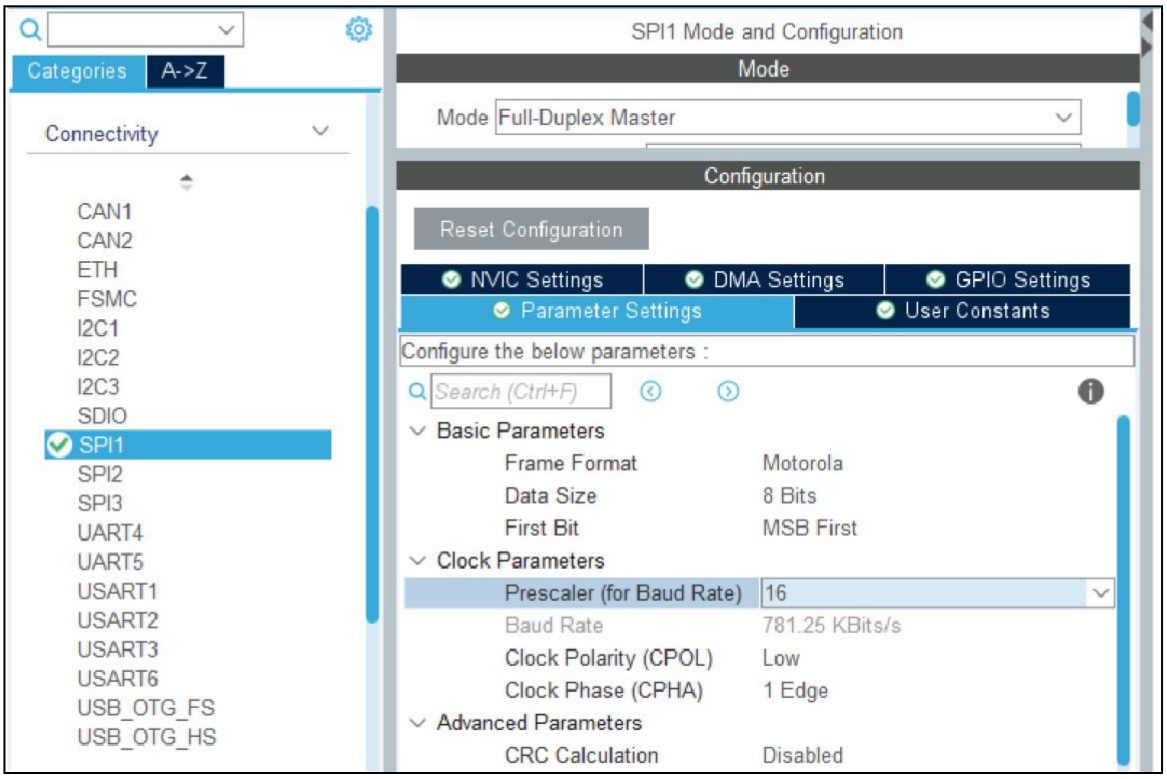
Interfacing Diagram:



Procedure to create project STM32CubeMX

1. Open the STM32CubeMx software and click on Access to board selector under new project. Select board as STM32F407G-DISC1.
2. Select board as STM32F407G-DISC1. Click on start project. Select yes for initialize all peripherals in default mode.
3. In STM32CubeMx window, in pinout and configuration tab, select pinout drop down list and click on clear pinout configuration.
4. Under System Core, select the Reset and clock control(RCC) and for high speed clock(HSE) select Crystal/Ceramic Resonator. Select SYS and for debug select serial wire.
5. Select PD15(connected to blue LED). Left click on PD13 and select the GPIO_Output functionality in Pinout view. Note that, other pins (PD12, PD15, PD14) can be used.

6. Default configuration works for clock. Note that, by default internal RC oscillator running at 16 MHz is selected for operation. The user can change the clock to source external crystal running at 8 MHz.
7. In categories, under connectivity select SPI1 and enable in Full duplex master mode and set the configuration as shown below. Select the prescaler to 16 to select baud rate less than 1 MHz.



Mode: Full-Duplex Master

Basic Parameters	
Frame Format	Motorola
Data Size	8 Bits
First Bit	MSB First
Clock Parameters	
Prescaler (for Baud Rate)	16
Baud Rate	781.25 KBits/s
Clock Polarity (CPOL)	Low
Clock Phase (CPHA)	1 Edge
Advanced Parameters	
CRC Calculation	Disabled

8. Configure PE3 as GPIO_Output to work as CS for accelerometer.
9. Configure PE0 as GPIO_EXTI0 to receive interrupt from slave device.
10. Select Project manager tab. Enter project name, project location. Select tool chain IDE as MDK-ARM and let the other options at default values.
11. Click on Generate code. This option will generate project file in Keil and initial configuration files in the location specified.

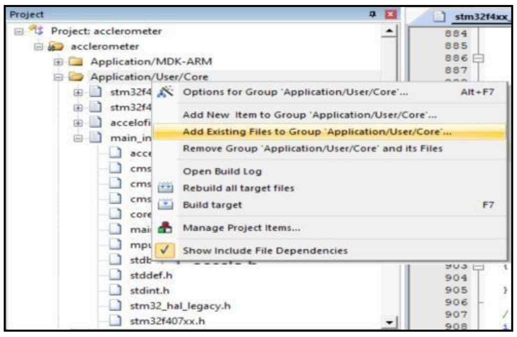
Program:

```
#include "main.h"
#include "accelo.h"


SPI_HandleTypeDef hspi1;

void SystemClock_Config(void);
static void MX_GPIO_Init(void);
static void MX_SPI1_Init(void);

LIS3DSH_DataScaled myData;
```

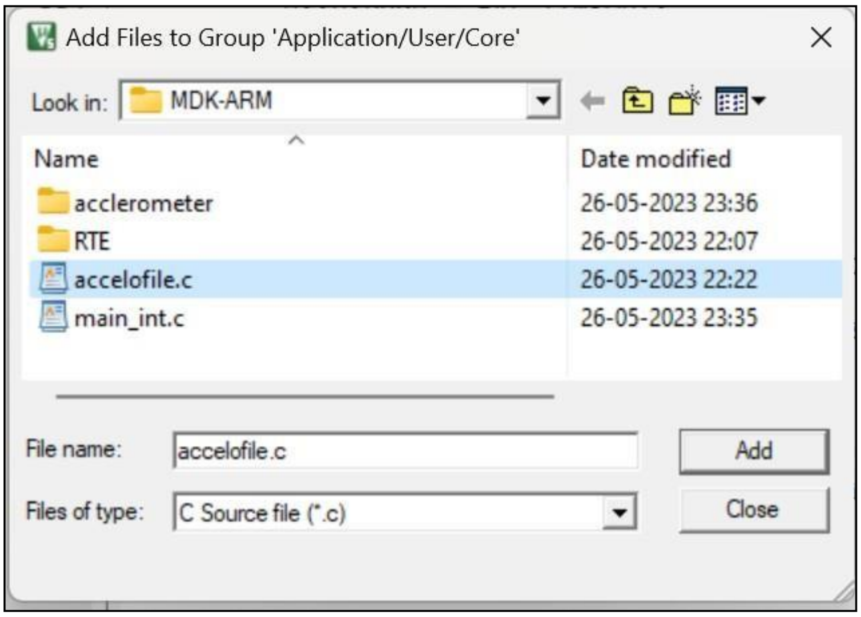
```
int main(void)
{
        LIS3DSH_InitTypeDef myaccelo_config;

        HAL_Init();

        SystemClock_Config();

        myaccelo_config.dataRate=LIS3DSH_DATARATE_12_5;
        myaccelo_config.fullScale=LIS3DSH_FULLSCALE_4;
        myaccelo_config.antiAliasingBW=LIS3DSH_FILTER_BW_50;
        myaccelo_config.enableAxes=LIS3DSH_XYZ_ENABLE;
        myaccelo_config.interruptEnable=false;

        MX_GPIO_Init();
        MX_SPI1_Init();

        LIS3DSH_Init(&hspi1,&myaccelo_config);

         while (1)
        {

                        if(LIS3DSH_PollDRDY(1000)==true){
                         myData= LIS3DSH_GetDataScaled();
                         HAL_GPIO_TogglePin(GPIOD,GPIO_PIN_13);
        }
    }

   }
```

Procedure to follow in Keil IDE:

1. Edit the program as shown in the listing. Make sure MDK-ARM folder of under project folder contain ‘accelo.h’ and ‘accelofile.c’.
2. Right click on the Applications/User/Core folder in project window. Select Add existing files to source group Applications/User/Core.


Project
Project: acclerometer
acclerometer
Application/MDK-ARM
Application/User/Core
Options for Group 'Application/User/Core'... Alt+F7
Add New Item to Group 'Application/User/Core'...
Add Existing Files to Group 'Application/User/Core'...
Remove Group 'Application/User/Core' and its Files
Open Build Log
Rebuild all target files
Build target F7
Manage Project Items...
Show Include File Dependencies
stddef.h
stdint.h
stm32_hal_legacy.h
stm32f407xx.h

3. Browse for the file ‘accelofile.c’ in MDK-ARM. Select the file and add toproject.


Add Files to Group 'Application/User/Core'

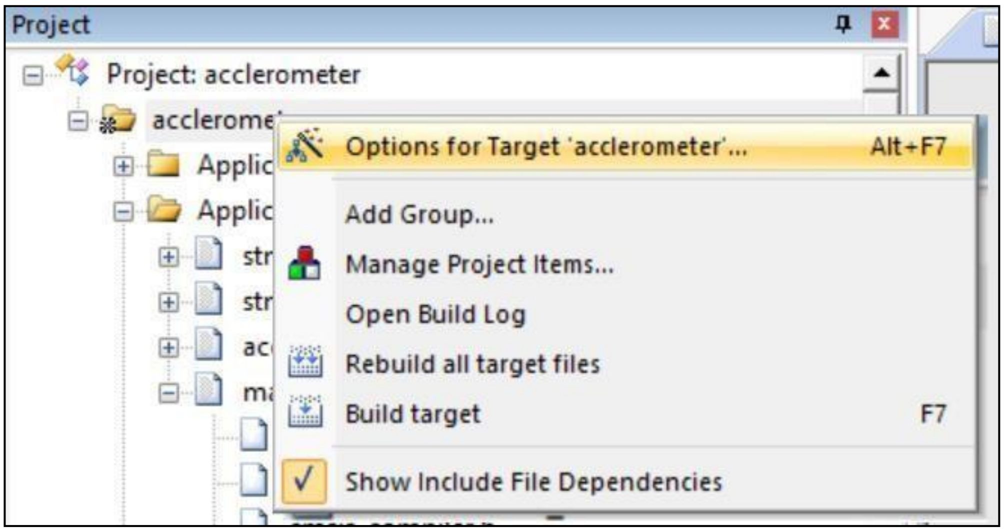
Look in: MDK-ARM

Name	Date modified
acclerometer	26-05-2023 23:36
RTE	26-05-2023 22:07
accelofile.c	26-05-2023 22:22
main_int.c	26-05-2023 23:35

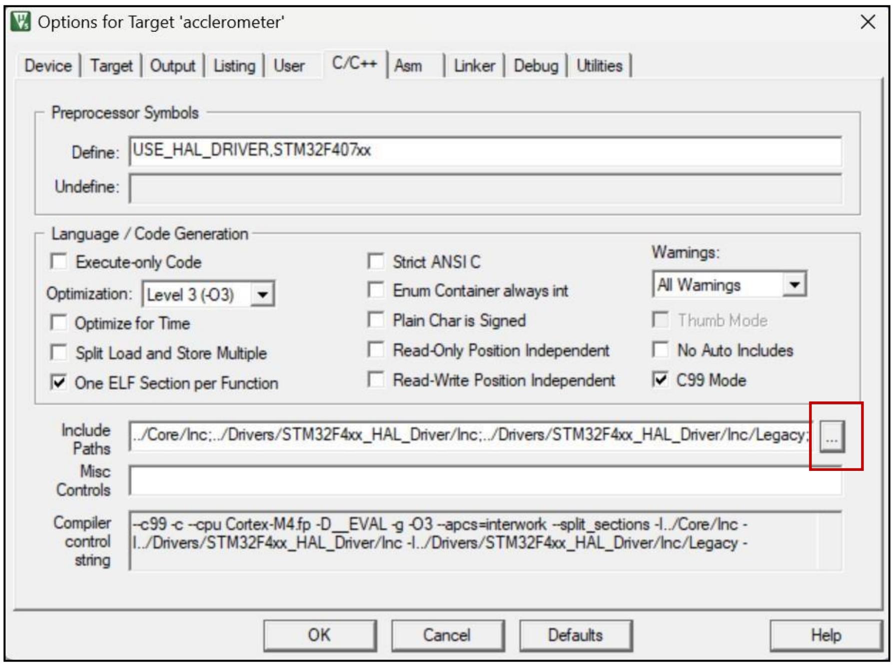
File name: accelofile.c

Files of type: C Source file (*.c)

4. In project window, right click project title and select options for target.



5. Select C/C++ tab and click on include path and add path of MDK-ARM.


Options for Target 'acclerometer'

Device | Target | Output | Listing | User | C/C++ | Asm | Linker | Debug | Utilities

Preprocessor Symbols

Define: USE_HAL_DRIVER,STM32F407xx

Undefine:

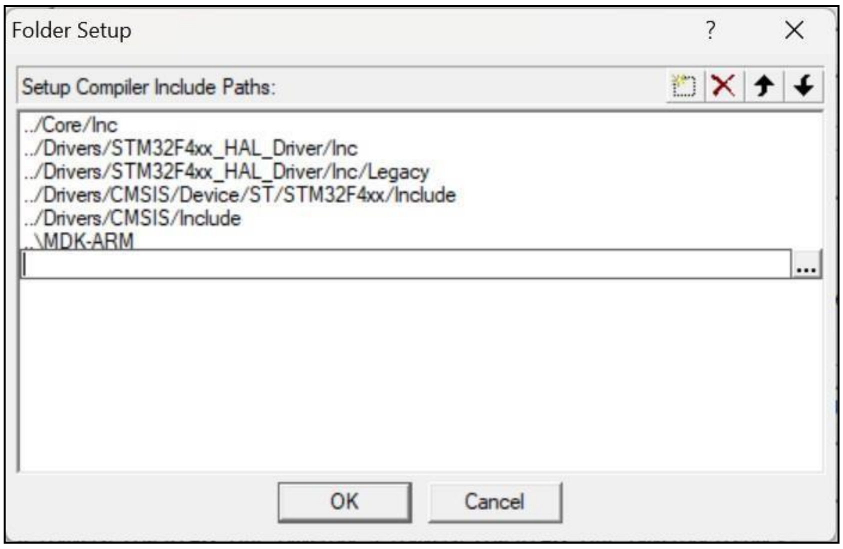
Language / Code Generation

- [ ] Execute-only Code
- Optimization: Level 3 (-O3)
- [ ] Optimize for Time
- [ ] Split Load and Store Multiple
- [x] One ELF Section per Function
- [ ] Strict ANSI C
- [ ] Enum Container always int
- [ ] Plain Char is Signed
- [ ] Read-Only Position Independent
- [ ] Read-Write Position Independent
- Warnings: All Warnings
- [ ] Thumb Mode
- [ ] No Auto Includes
- [x] C99 Mode

Include Paths: ../Core/Inc;../Drivers/STM32F4xx_HAL_Driver/Inc;../Drivers/STM32F4xx_HAL_Driver/Inc/Legacy;

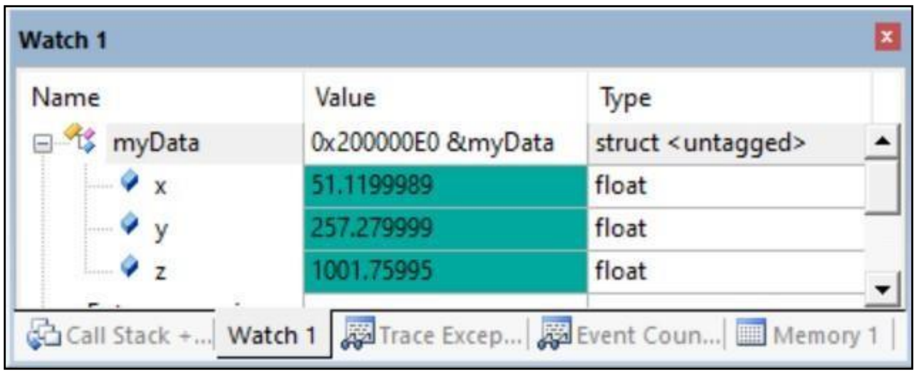
Misc Controls:

Compiler control string:
```
--c99 -c --cpu Cortex-M4.fp -D__EVAL -g -O3 --apcs=interwork --split_sections -I../Core/Inc -
I../Drivers/STM32F4xx_HAL_Driver/Inc -I../Drivers/STM32F4xx_HAL_Driver/Inc/Legacy -
```


Folder Setup

Setup Compiler Include Paths:
```
../Core/Inc
../Drivers/STM32F4xx_HAL_Driver/Inc
../Drivers/STM32F4xx_HAL_Driver/Inc/Legacy
../Drivers/CMSIS/Device/ST/STM32F4xx/Include
../Drivers/CMSIS/Include
..\MDK-ARM
```

6. After successful build, start debug session and add myData to watch window to 3 axis accelerometer data.


Watch 1

Name	Value	Type
myData	0x200000E0 &myData	struct <untagged>
x	51.1199989	float
y	257.279999	float
z	1001.75995	float

Alternate program using interrupt:

```
#include "main.h"
#include "accelo.h"

SPI_HandleTypeDef hspi1;

void SystemClock_Config(void);
static void MX_GPIO_Init(void);
static void MX_SPI1_Init(void);

LIS3DSH_DataScaled myData;
uint8_t flag=0;

void HAL_GPIO_EXTI_Callback(uint16_t GPIO_Pin)
{

      UNUSED(GPIO_Pin);
        flag=1;
      HAL_GPIO_TogglePin(GPIOD,GPIO_PIN_13);
}

int main(void)
{

    LIS3DSH_InitTypeDef myaccelo_config;

    HAL_Init();
    SystemClock_Config();

     myaccelo_config.dataRate=LIS3DSH_DATARATE_12_5;
    myaccelo_config.fullScale=LIS3DSH_FULLSCALE_4;
    myaccelo_config.antiAliasingBW=LIS3DSH_FILTER_BW_50;
    myaccelo_config.enableAxes=LIS3DSH_XYZ_ENABLE;
    myaccelo_config.interruptEnable=true;
    MX_GPIO_Init();
    MX_SPI1_Init();

    LIS3DSH_Init(&hspi1,&myaccelo_config);

    while (1)
    {

          if(flag==1){
                flag=0;
               myData= LIS3DSH_GetDataScaled();
              HAL_GPIO_TogglePin(GPIOD,GPIO_PIN_13);
    }
  }
}
```

Space to answer viva questions:

Sl. No	**Criteria**	**Max Marks**	**Marks obtained**
Data sheet			
A	Analysis of program flow and instructions used/ Writing program	04	
B	Simulation/ Conduction of the experiment/ Analysis of Results	04	
C.	Viva	2	
	Total	10	

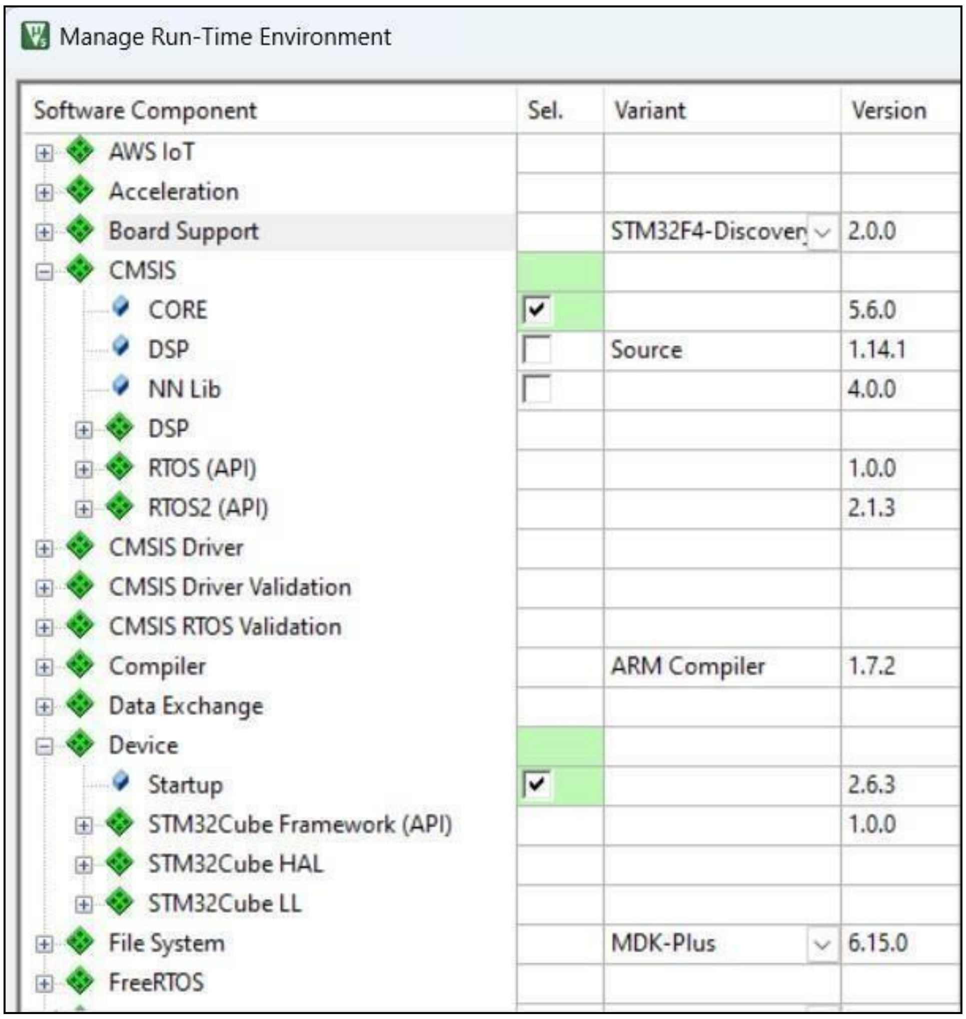
PROGRAMMING NVIC AND WRITING INTERRUPT SERVICE ROUTINES

Experiment 09

a) Write a program toggle a RED LED three times on push button press event by writing interrupt service routine otherwise toggle green LED.

Procedure to edit and build code in Keil IDE:

- Project -> New µVision Project. Give a suitable name to the project (No whitespaces) and Save.
- Under Select Device for Target window, STMicroelectronics -> STM32F4 Series ->STM32F407 -> STM32F407VG->STM32F407VGTx.
- Under Software Component window, select the options as shown below and click Ok.

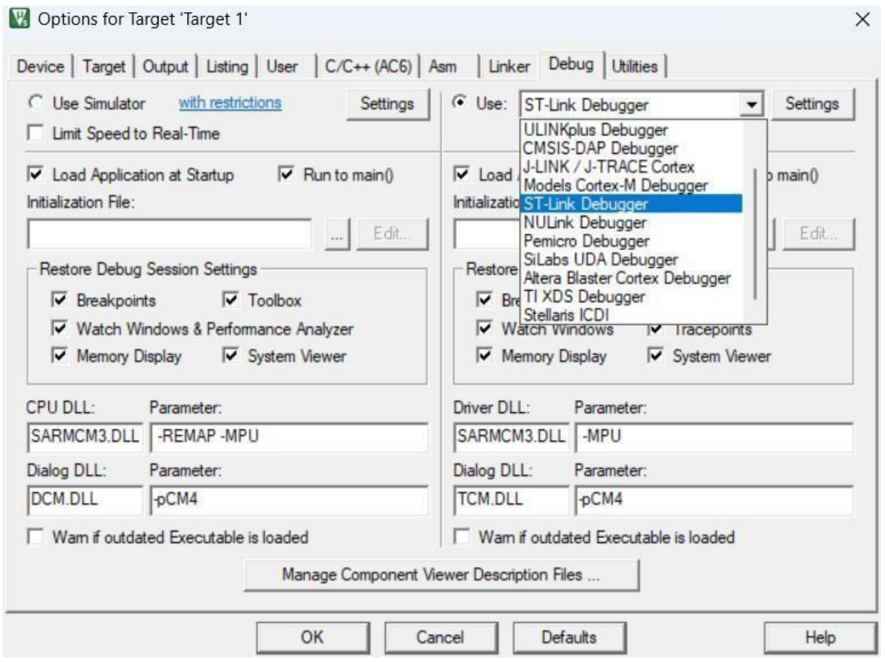


Manage Run-Time Environment

Software Component	Sel.	Variant	Version
AWS IoT			
Acceleration			
Board Support		STM32F4-Discovery	2.0.0
CMSIS			
CORE	✓		5.6.0
DSP	☐	Source	1.14.1
NN Lib	☐		4.0.0
DSP			
RTOS (API)			1.0.0
RTOS2 (API)			2.1.3
CMSIS Driver			
CMSIS Driver Validation			
CMSIS RTOS Validation			
Compiler		ARM Compiler	1.7.2
Data Exchange			
Device			
Startup	✓		2.6.3
STM32Cube Framework (API)			1.0.0
STM32Cube HAL			
STM32Cube LL			
File System		MDK-Plus	6.15.0
FreeRTOS			

An empty project will be created.

- Expand Target 1 and right-click on Source Group 1 -> Add New Item to Group 'Source Group 1', Select C file and give it a suitable name and select Add.
- For debugging connection need to be selected click on Debug Tab select use emulator option and select debugger as ST-LINK Debugger and click OK and complete target option setting.



Options for Target 'Target 1'

Device | Target | Output | Listing | User | C/C++ (AC6) | Asm | Linker | Debug | Utilities

( ) Use Simulator with restrictions Settings
☐ Limit Speed to Real-Time
☑ Load Application at Startup ☑ Run to main()
Initialization File:
Restore Debug Session Settings
☑ Breakpoints ☑ Toolbox
☑ Watch Windows & Performance Analyzer
☑ Memory Display ☑ System Viewer
CPU DLL: SARMCM3.DLL Parameter: -REMAP -MPU
Dialog DLL: DCM.DLL Parameter: -pCM4
☐ Warn if outdated Executable is loaded

(●) Use: ST-Link Debugger Settings
ULINKplus Debugger
CMSIS-DAP Debugger
J-LINK / J-TRACE Cortex
Models Cortex-M Debugger
ST-Link Debugger
NULink Debugger
Pemicro Debugger
SiLabs UDA Debugger
Altera Blaster Cortex Debugger
TI XDS Debugger
Stellaris ICDI
☑ Load ☑ Run to main()
Initialization File:
Restore Debug Session Settings
☑ Breakpoints
☑ Watch Windows ☑ Tracepoints
☑ Memory Display ☑ System Viewer
Driver DLL: SARMCM3.DLL Parameter: -MPU
Dialog DLL: TCM.DLL Parameter: -pCM4
☐ Warn if outdated Executable is loaded

Manage Component Viewer Description Files ...

OK Cancel Defaults Help

- Write the code in the created C file and save.

Program:

```
#include "stm32f4xx.h"
void delayMs(int n);


int main(void)
{
    RCC->AHB1ENR |=  9;              /* enable Cloock to GPIOA & GPIOD */
    RCC->APB2ENR |= (1<<14);      /* enable Cloock to System Configuration Controller */


    GPIOD->MODER |=  0x11000000;      /* set pin 12&14 to output mode Green & orange LED*/
    GPIOA->MODER &= ~(0x00000003);       /* clear pin mode to input mode  */


    SYSCFG->EXTICR[0]&=~(0xF<<0);         /* enable EXTI0 for PA0 */
    EXTI->IMR |= (1<<0);                     /* Interrupt is not Masked on LIne0  */
    EXTI->RTSR |= (1<<0);                   /* Rising Edge trigger enabled */
    EXTI->FTSR &= ~(1<<0);                /* Falling Edge trigger disabled */


     NVIC->ISER[0] |= (1<<6);


        while(1)
            {
              GPIOD->BSRRL = 0x1000; /* turn ON green LED */
              delayMs(300);
              GPIOD->BSRRH = 0x1000; /* turn OFF green LED */
```

```
          delayMs(300);
      }
}

void EXTI0_IRQHandler(void)
{
            int i;
            for(i=0;i<3;i++)
            {
                  GPIOD->BSRRL = 0x4000; /* turn ON RED LED*/
                  delayMs(300);
                   GPIOD->BSRRH = 0x4000; /* turn OFF RED LED */
                    delayMs(300);
            }
            EXTI->PR |=(1<<0);
}

void delayMs(int n)
{
    int i;
    for (; n > 0; n--)
        for (i = 0; i < 3195; i++) ;
}
```

- Go to project and select build target.
- Load executable on to target and run the program.

Space to answer viva questions:

.,

Sl. No	Criteria	Max Marks	Marks obtained
Data sheet			
A	Analysis of program flow and instructions used/ Writing program	04	
B	Simulation/ Conduction of the experiment/ Analysis of Results	04	
C.	Viva	2	
	Total	10	

Part III: Innovative Experiments

1. Program SPI and show the configuration and data transfer between SPI slave device and master.
2. Program ADC and show interfacing of analog sensor for given specifications.
3. Data transfer in polling, interrupt and DMA based modes.
4. Real time Audio applications: Flanging effect.