ELECTRICAL AND ELECTRONICS
ENGINEERING DEPARTMENT

EE447 Introduction to Microprocessors

Term Project Final Report

Audio Frequency Based
Stepper Motor Driver

Name : Mustafa Barış Emektar
ID : 2304533
Sec : 1 / TH-2
Date : 06.02.2022

CONTENTS

Introduction

The major goals of the term project are to encapsulate the work into sub-tasks and maintain module cooperation as well as to understand and implement serial communication on the TM4C123G and to use the facility on the SPI protocol. Also, the student will gain the ability to write a multi-task software with a complex setup and an understanding of the cooperation of utility modules. Finally, the one will be familiar with the ARM CMSIS DSP software library after this project.

In this project, it is expected to design an audio frequency-based stepper motor driver, as can be understood from the title. The speed of the step motor will be adjusted according to the dominant frequency in the environment. The system has three major functions: audio sampling, frequency detection, and user interface.

In this final report, the overall project will be discussed. The details of the components with their connections will be shared. The tables and figures will be given for connections and the current project setup. Then the details of all the subsystems and the critical point of the project will be discussed. The necessary parts of the codes will be shared and discussed.

Overall Project

In the project, all the operations are handled by the interrupts and subroutines. The main program is only used for GPIO, GPTM, or ADC initializations. The components with their detailed port connections will be shown in this part of the report. Also, the implementation of those connections will be shown with taken photos.

Components & Connections

As mentioned above, TM4C123G board and different components are used for the project. The component list can be seen in Table 1.

Also, table 2 shows the connections for the components

Components of Project
TM4C123G board
GY-MAX9814 Microphone Module
NOKIA 5110 LCD Screen
1 Potentiometer (optional)
1 4x4 Keypad (optional)
2 Buttons and RGB LED Placed on the TM4C123G Board
Stepper Motor

Table 1: Components of Project

Peripherals	**Pins on Peripheral / Pin Name**	**TM4C123G**
LCD Screen	RST	PA7
	SCE	PA3
	D/C	PA6
	DN	PA5
	SCL	PA2
	VCC	3.3V
	LED	VBUS
	GND	GND
Stepper Motor	IN1	PB7
	IN2	PB6
	IN3	PB5
	IN4	PB4
	VCC	VBUS
	GND	GND
Microphone	Output	PE3
	VDD	VBUS
	GND	GND
2 LEDs	Red	PF1
	Green	PF3
	Blue	PF2
Buttons	Sw1	PF4
	Sw2	PF0

Table 2: Connections of the Components

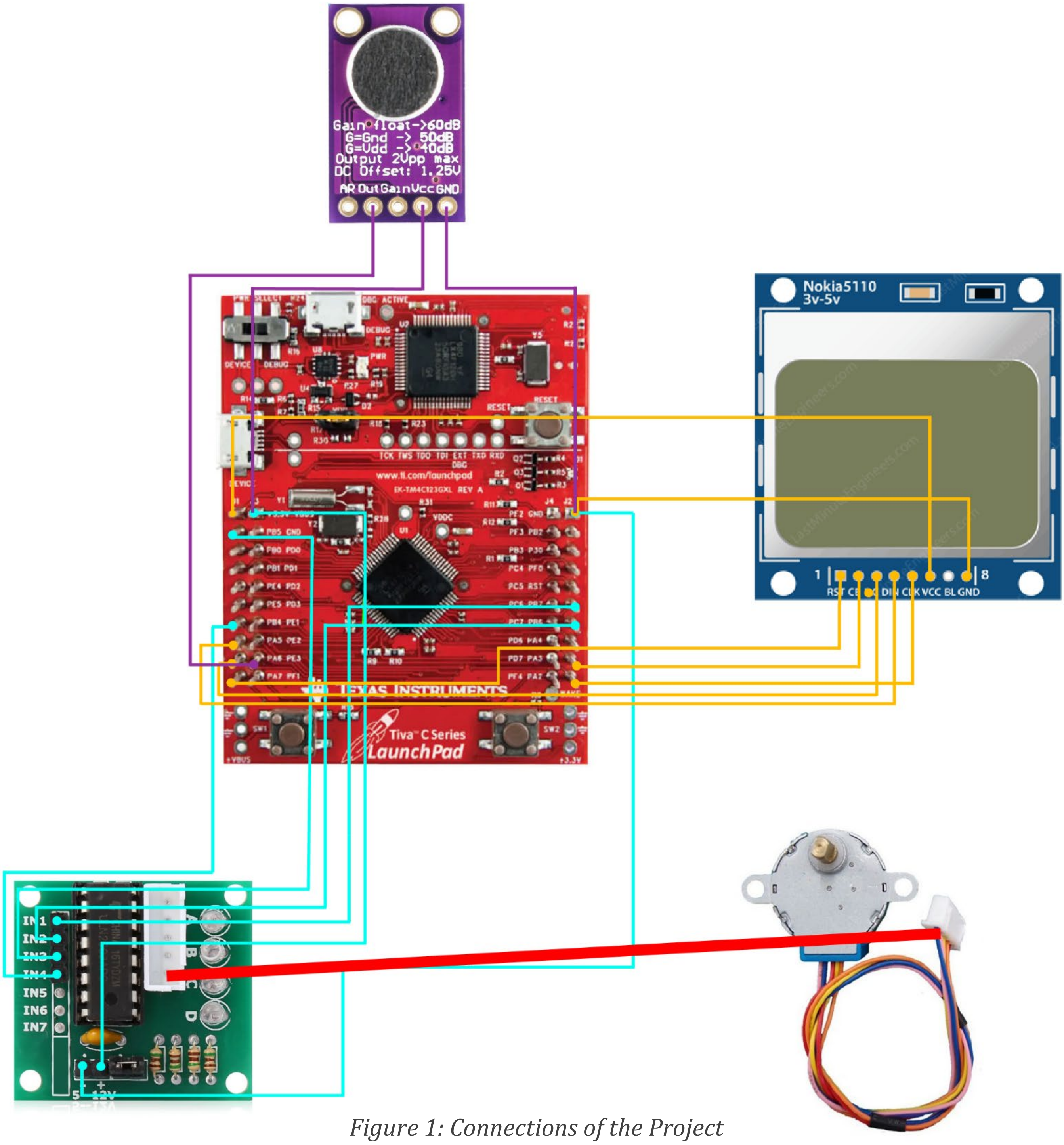


Figure 1: Connections of the Project

Pin connections are illustrated in the figure above. More than one component may use a specific pin on the Tiva board. So, a breadboard is used to multiply the same port. Also, the final product is shown in the following figures 3 and 4 with photos.

The ground, 3.3V, and VBUS pins of the Tiva board are multiplied on the breadboard. The red cables for VBUS, Orange cables for 3.3V, and others for ground connections.

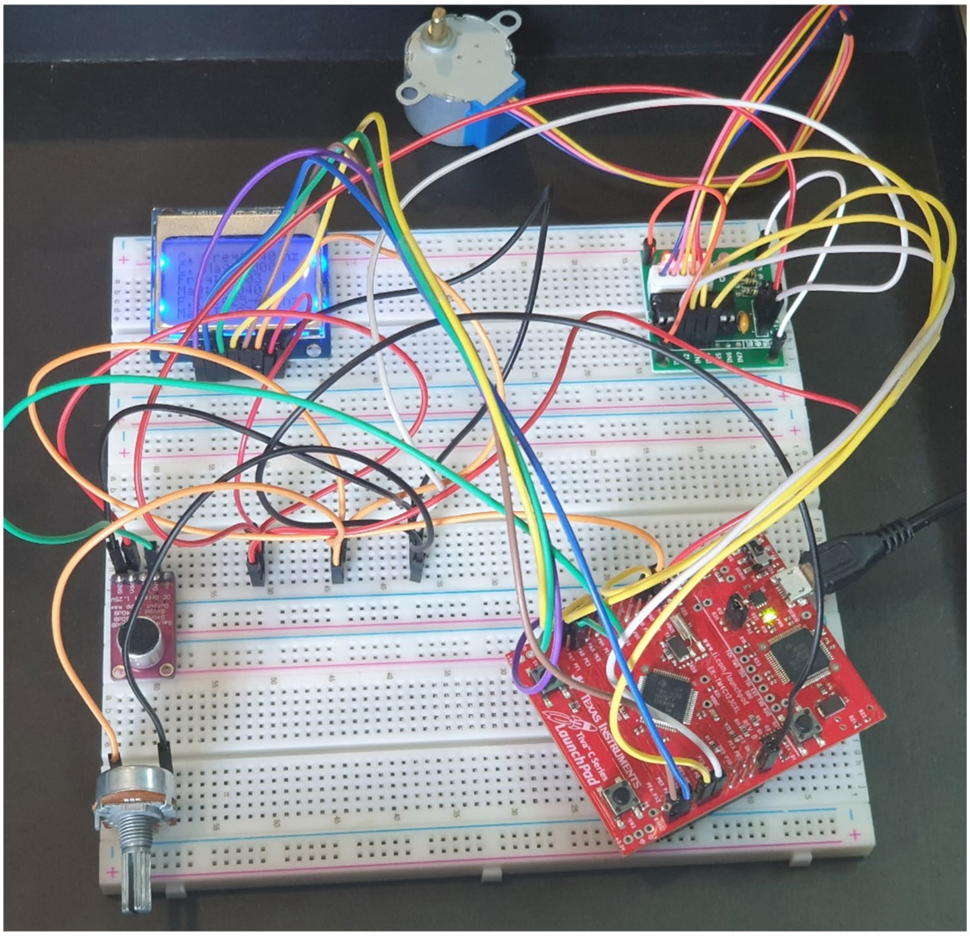


Figure 2: Photo of the project setup

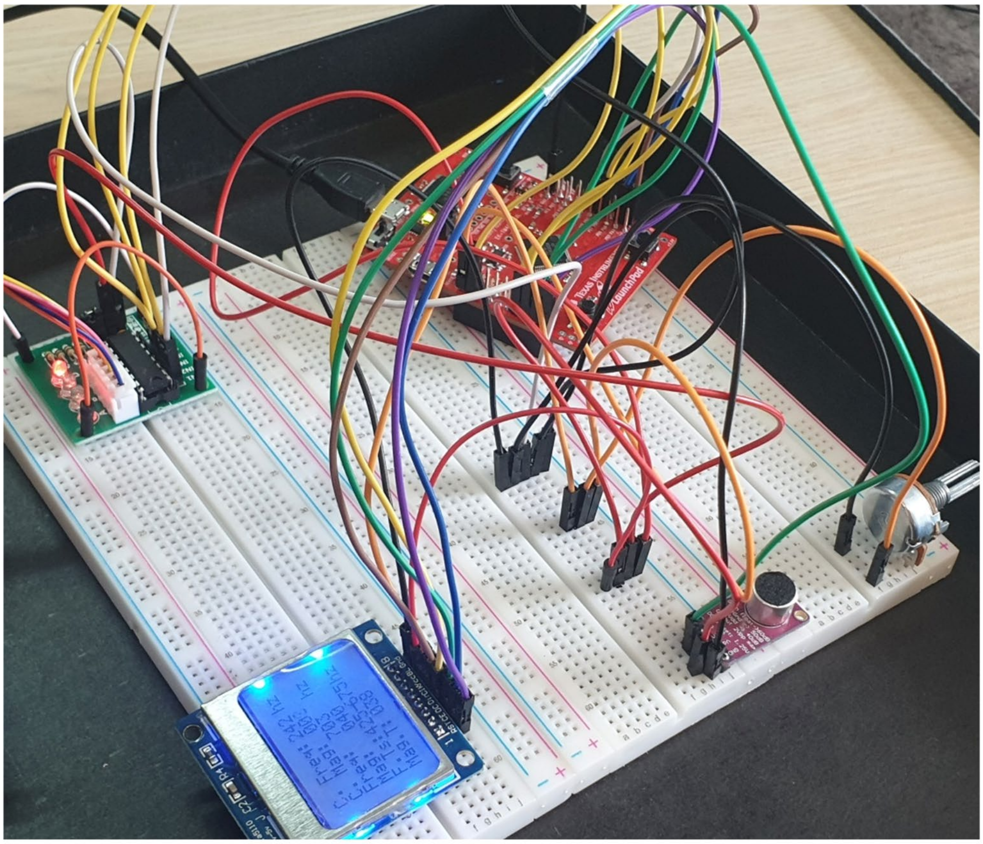


Figure 3: Photo of the project setup

Flowchart

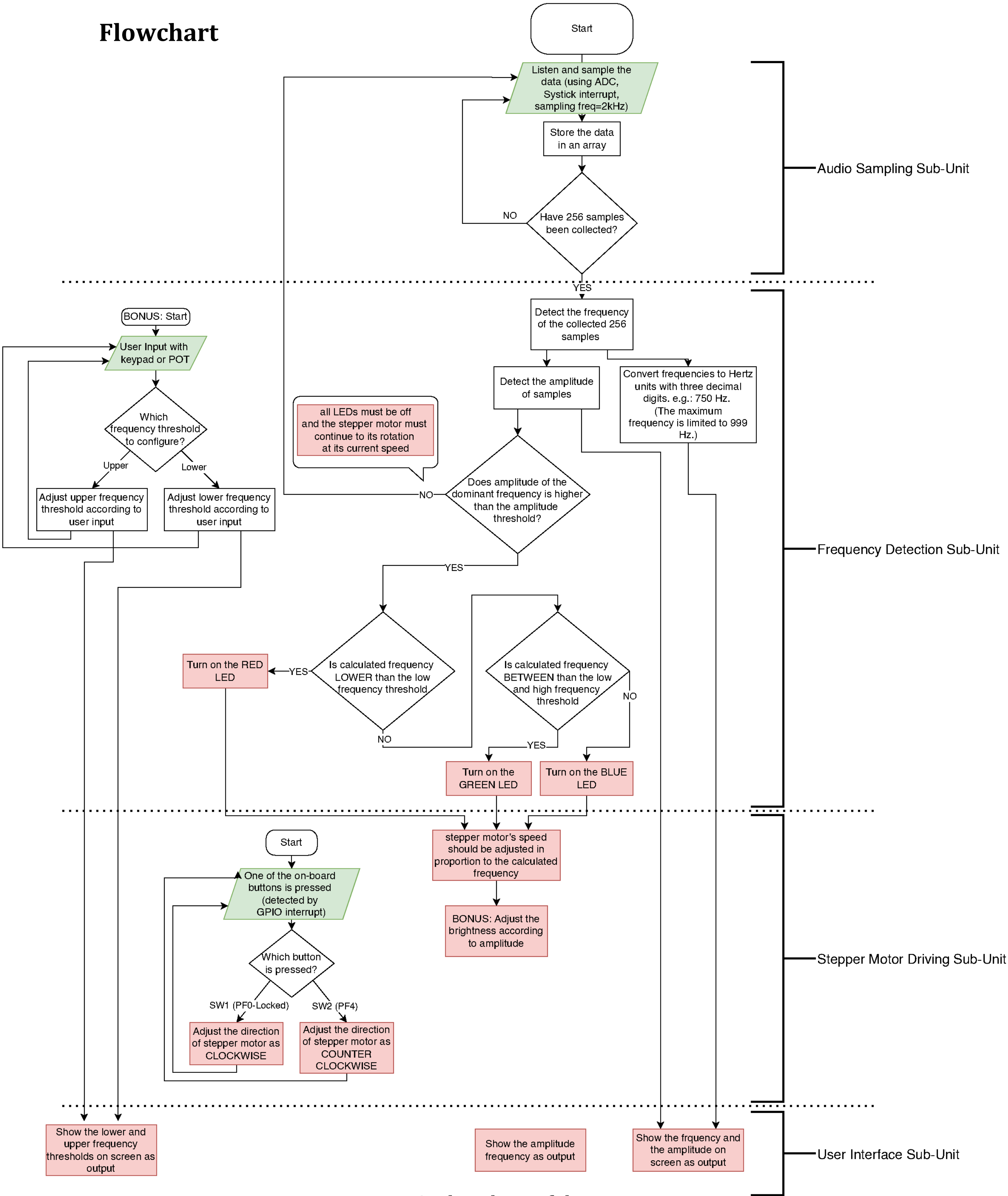


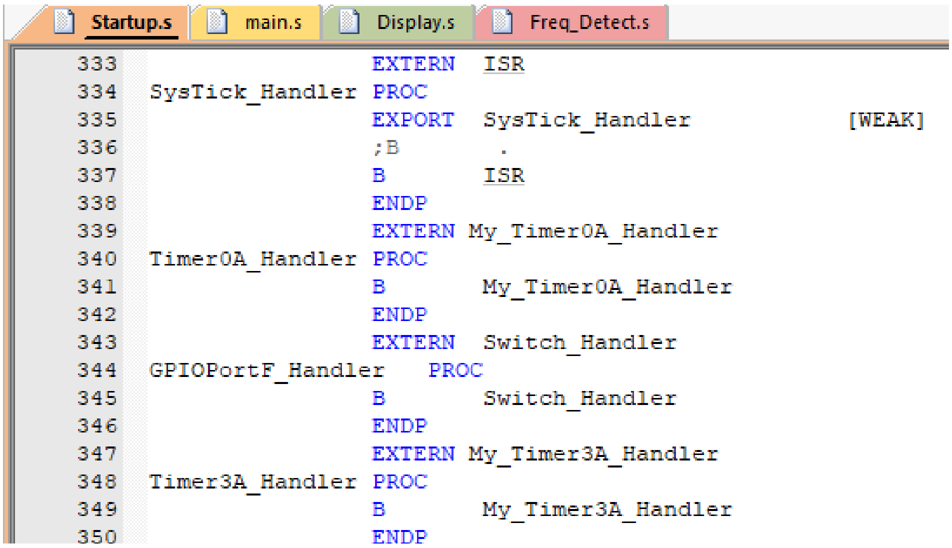
Figure 4: Flowchart of the project

There are some minor changes on projects after the preliminary report. So I decided the update the flowchart I shared in the preliminary report. The above figure shows the flowchart of the Project. The functionalities of the sub-units are well defined and separated from each other. The red rectangles indicate the output, and the green rectangles indicate the inputs.

Subsystems

I decided to divide this project into four subsystems for better insight. All the subsystems focus on specific components, which are stated inside them below. I will not go into detail about the initializations or subroutines. Instead, I will be discussing the solution or the algorithm applied with them.

There are four interrupts created in the program. They will be mentioned under the related subsystems.




```
                EXTERN  ISR
SysTick_Handler PROC
                EXPORT  SysTick_Handler            [WEAK]
                ;B       .
                B       ISR
                ENDP
                EXTERN My_Timer0A_Handler
Timer0A_Handler PROC
                B       My_Timer0A_Handler
                ENDP
                EXTERN  Switch_Handler
GPIOPortF_Handler    PROC
                B       Switch_Handler
                ENDP
                EXTERN My_Timer3A_Handler
Timer3A_Handler PROC
                B       My_Timer3A_Handler
                ENDP
```

Figure 5: Screenshot of the Startup.s showing the interrupt handlers

Audio Sampling Subsystem

GY-MAX9814 Microphone Module is used in this subsystem. PortE_Init subroutine is used to initialize PE3. ADC0 is used for analog sampling. Also, the systick timer is used to create a simple timer. InitSysTick is used to initialize systick interrupts. All the procedures in this subsystem have been handled with the systick interrupt subroutine ISR.s. The backbone of that subsystem is the ARM CMSIS DSP library.

In an endless loop, in ISR, the system listens to the audio signal at a constant sampling frequency which is 2kHz. Due to sampling frequency, only the signals below 2kHz/2=1kHz can be sensed. The samples are kept in a 256-element array in the memory, as shown in figure 5. The data address starts from 0x20000400. Just for you to notice, the imaginary parts are zero. Also, the number of current data in the array stored in the address 0x20000350 to avoid losing the numbers samples currently in the array. Because the system operates lots of things between two calls of ISR, this number cannot be transferred with a register. As can be seen in figure 6, it counts up to 0xFE from 0x00.

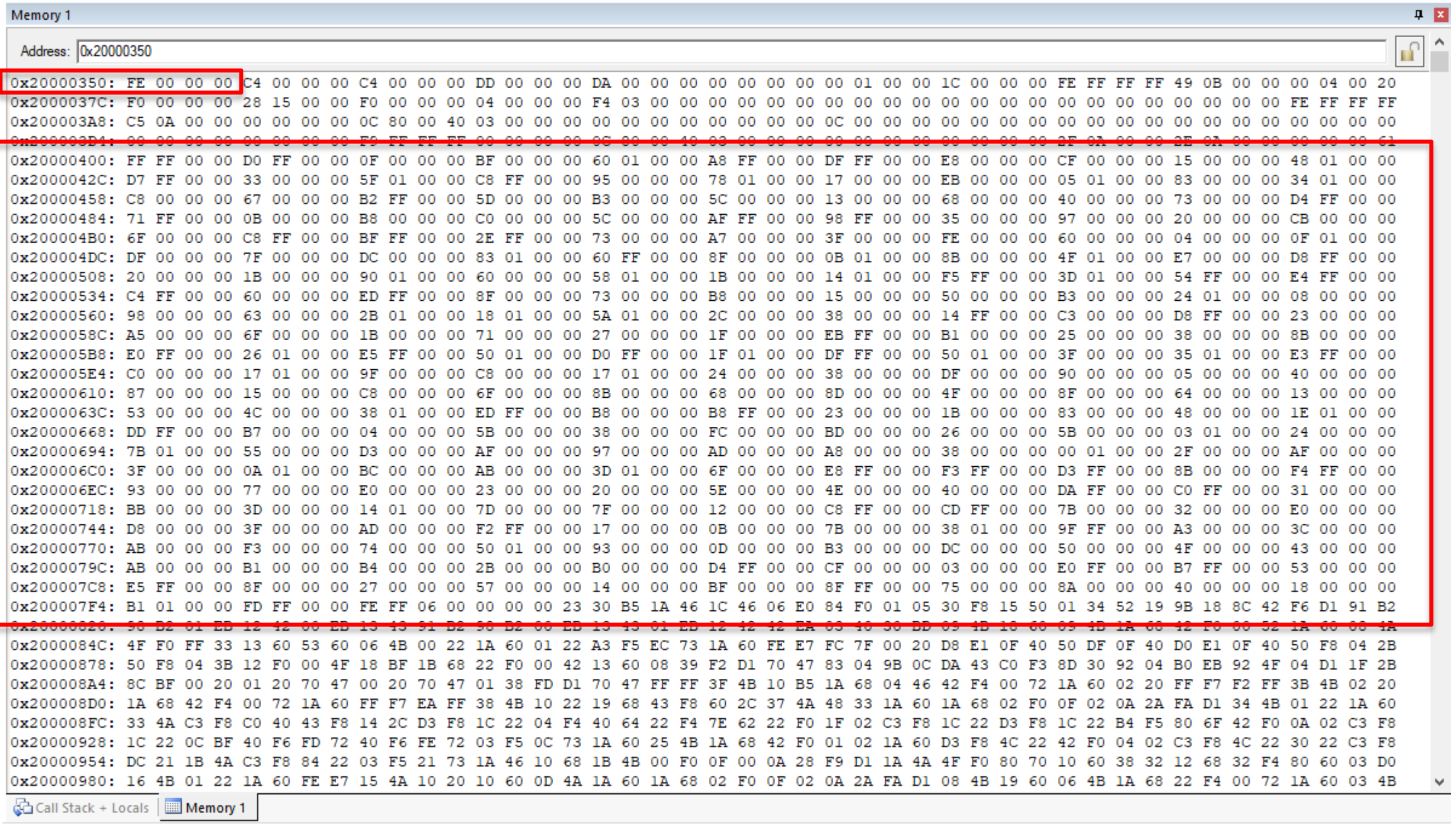


Figure 6: Stored raw (real) audio signals

The system calls the FFT function from the DSP library when the array is filled. It calculates the FFT of given data and writes the results back to the same locations in memory (The returned data includes the frequencies obtained from the sampled signals). So, the screenshot of the FFT results is shown in figure 6. Also, zero is written to address 0x20000350 so that the sampling begins later again.

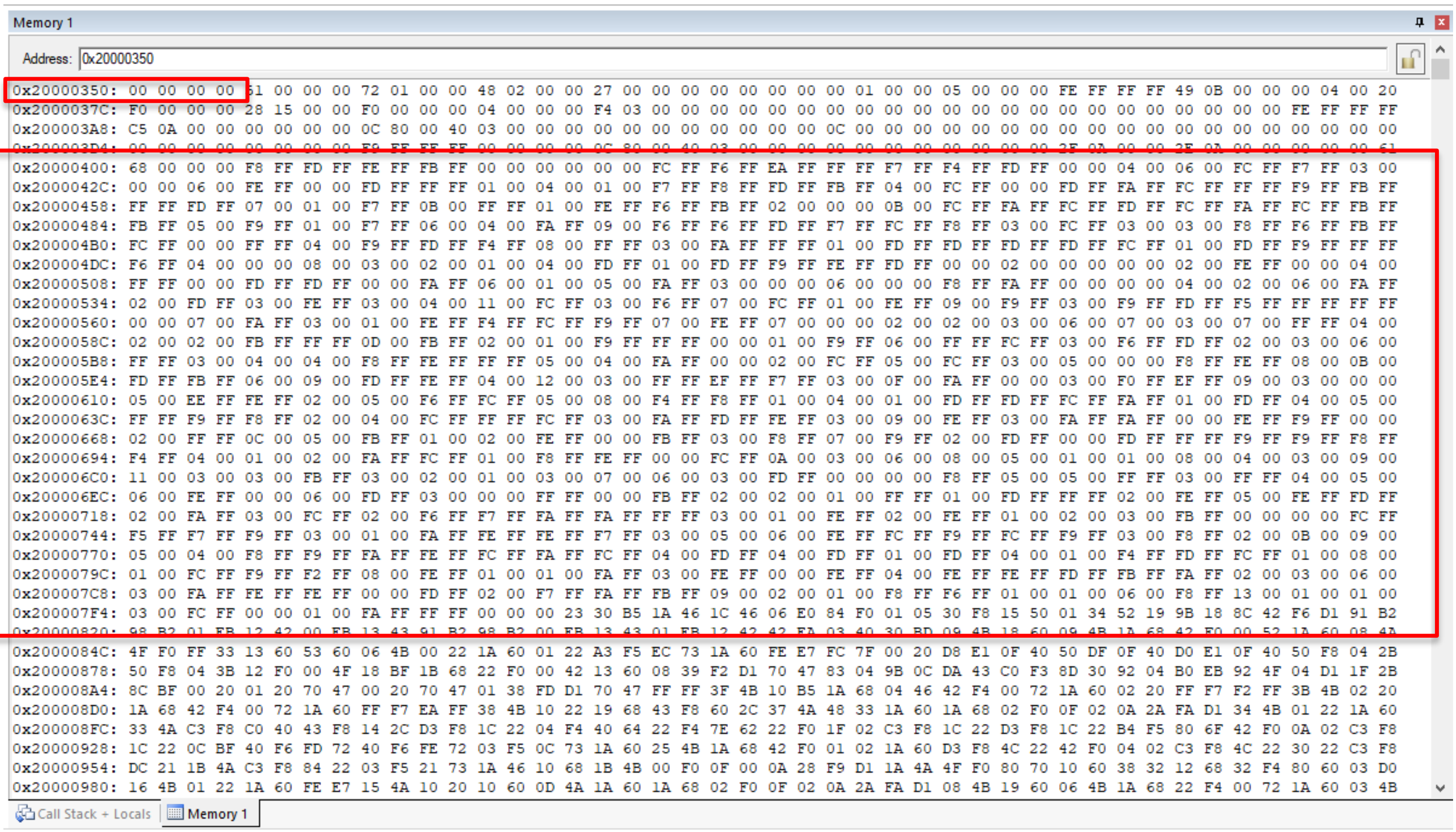


Figure 7: After taking the FFT of stored audio signals

Figure 8: ISR subroutine which is called with 2khz

After that point, the Freq_Detect subroutine has called. So, that brings us to the frequency detection subsystem.

Frequency Detection Subsystem

RGB LEDs Placed on the TM4C123G Board are used in this subsystem. PortF_Init is called to initialize LEDs and the buttons. PF0 must be unlocked to use the SW2 button. Also, the PF1 must be adjusted as the pull-up resistor. So that, when it is not pressed, it stays as high; however, when it is pressed, it becomes zero, and the system detects that it is pressed.

This frequency detection sub-unit detects the frequency of samples obtained in the audio sampling sub-unit. This subsystem is applied in the Freq_Detect subroutine. In this subroutine, the addresses starting from the 0x20000400 are visited one by one, and the higher magnitude is found by simply taking squares of the real and imaginary parts and adding them. Then the dominant frequency and its magnitude are stored in the memory addresses 0x20000200 & 0x20000204, respectively. So when the Timer3A creates interrupt, they are sent to the current_display subroutine to print them on screen.

Timer3A is initialized to call the current_display subroutine every 1 second. TAILR register is loaded with 62500, and TAPR is loaded with 256 to get a 15μs count interval. So, we get 62500*256=16M -> 1 second timer. Therefore, the detected frequency and magnitude are printed on the screen every second, even if the signal's magnitude is below the magnitude threshold.

Then, this frequency's magnitude is compared with the predefined magnitude. If the new signal's magnitude is greater than the predefined magnitude threshold, the LEDs and the motor speed are adjusted according to the new signal. The Timer0A's reload value is changed proportional to the frequency for adjusting the motor speed. The frequency is compared with lower (425 Hz) and upper (675 Hz) frequency thresholds for the LEDs. Then the related LED is turned on. Also, the screen is updated with the new frequency value and its magnitude.

RED	f < 425 hz
GREEN	425 hz < f < 675 hz
BLUE	675 hz < f

If the magnitude of the signal is lower than the magnitude threshold, then no LED will be turned on, and the motor rotates with its previous speed.

Startup.s | main.s | Display.s | Freq_Detect.s

```
                LDR R1,=GPIO_PORTF_DATA
                BIC R2,#0xE

                MOV R3,#low_freq_thres
                CMP R11,R3
                BLT red_led

                MOV R3,#high_freq_thres
                CMP R11,R3
                BGT blue_led
                ORR R2,#0x8
                STR R2,[R1]
                B   terminate

red_led         ORR R2,#0x2
                STR R2,[R1]
                B   terminate

blue_led        ORR R2,#0x4
                STR R2,[R1]
                B   terminate

no_led          LDR R1,=GPIO_PORTF_DATA
                BIC R2,#0xE
                STR R2,[R1]


terminate       POP {R0,R1,R2,R3,R4,R5,R6,R7,R8,R9,R11,R12,LR}
                BX LR
                ENDP
                ALIGN
                END
```

Figure 8: A part of the Freq_Detect subroutine which handles the LED selection after FFT calculation

Stepper Motor Driving Subsystem

Stepper Motor and 2 Buttons placed on the TM4C123G Board are used in this subsystem. PortB_Init is called to initialize the stepper motor, and PortF_Init is called to initialize LEDs and the buttons. The interrupts for port f are activated. Also, the Timer0A is set up for the stepper motor. The timer creates an interrupt after every cycle to rotate the motor.

This subsystem is not complex as the others. The only responsibility of that subsystem is to adjust the motor's direction. When one of the buttons, SW1 or SW2, is pressed, the software will cause an interrupt. The motor's direction will then be altered accordingly.

SW1	ClockWise
SW2	CounterClockWise

Startup.s | main.s | Display.s | Freq_Detect.s | PortF_Init.s

```
;*******************************************************************
Switch_Handler  PROC
                PUSH {R0,R1,R2,R3,R4,R5,R6,R7,R8,R9,R11,R12,LR}

                LDR R1,=GPIO_PORTF_DATA
                LDR R0, [R1]

                BIC R0,#0xFFFFFFEE
                MOV R3,#0x10
                CMP R0,R3
                BEQ cw

                LDR R1,=dir_addr     ; SW2 is pressed -> ccw
                MOV R2,#1
                STR R2, [R1]
                B   quit

cw              LDR R1,=dir_addr     ; SW1 is pressed -> cw
                MOV R2,#0
                STR R2, [R1]


quit            LDR R1,=GPIO_PORTF_ICR  ;clear interrupt flag
                MOV R0,#0x11
                STR R0, [R1]

                POP {R0,R1,R2,R3,R4,R5,R6,R7,R8,R9,R11,R12,LR}
                BX LR
                ENDP
            ALIGN
        END
```

Figure 9: A part of the Switch_Handler (PortF Handler) which handles rotation direction of the motor according to pressed button

User Interface Subsystem

NOKIA 5110 LCD Screen is used in this subsystem. PortA_Init is called to initialize it. The SPI protocol is used to communicate with the screen.

The outputs will be shown in the screen and updated continuously in this part. All the values are passed with registers. Timer3A handler calls the current_display subroutine as discussed above. Also, the Freq_Detect subroutine calls display subroutine to print currently used frequency and magnitude and their thresholds. The photo of the display while working is shown in figure 7. The outputs are as follows:

-C. Freq: ---> Current frequency
-C. Mag: ---> Current amplitude and their thresholds
-Freq: ---> Frequency of the signal (the motor rotates accordingly with it)
-Mag: ---> Amplitude of the signal (the motor rotates accordingly with it)
-F.Ts: ---> Frequency Thresholds (Lower- Upper)
-Mag.T: ---> Magnitude Threshold

When Timer3A handler subroutine calls the current_display subroutine, this subroutine creates the first two screen lines, showing the environment's instant frequency and amplitude. However, if the instant signal's magnitude is lower than the magnitude threshold, the system will not update the following two lines showing the previous dominant frequency (which determines the motor's current rotation speed). If the instant signal's magnitude is higher than the threshold, then the two lines in the middle are updated according to its values.

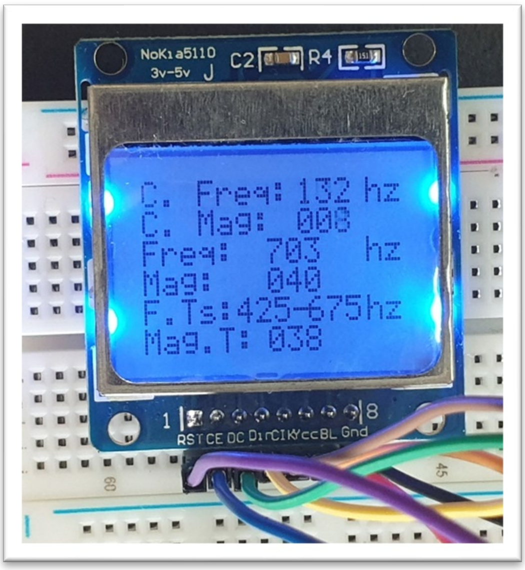


Figure 10: Working LCD Display and Shown Outputs

The letters and words on LCD are predefined Display.s, and they are printed every time one of its subroutines is called. However, the numbers (as discussed) passed by the registers as follows:

- The current frequency is passed with R7
- The current magnitude is passed with R8

for current_display subroutine

- Below frequency threshold is passed with R2
- Above frequency threshold is passed with R3
- Frequency is passed with R4
- Magnitude threshold is passed with R5
- Magnitude is passed with R6

for display subroutine

If the number is higher than "999", then the system automatically prints "999". That is, only 3-digits numbers are supported.

Startup.s ISR.s Display.s

```
current_display
            PUSH {R0,R1,R2,R3,R4,R5,R6,R7,R8,R9,R11,R12,LR}
            ; Current frequency is passed with R7
            ; Current magnitude is passed with R8
        ; first row
                MOV R0,#0       ;X
                MOV R1,#0       ;Y
                BL set_coords
                LDR R5,=c_freq_msg
                BL print_str_nokia

            ; coordinates of current frequency
                MOV R0,#50      ;X
                MOV R1,#0       ;Y
                BL set_coords
            ; print frequency
                MOV R5,R7
                BL print_number

        ; second row
                MOV R0,#0       ;X
                MOV R1,#1       ;Y
                BL set_coords
                LDR R5,=c_mag_msg
                BL print_str_nokia

            ; coordinates of current magnitude
                MOV R0,#50      ;X
                MOV R1,#1       ;Y
                BL set_coords
            ; print  magnitude
                MOV R5,R8
                BL print_number
            POP {R0,R1,R2,R3,R4,R5,R6,R7,R8,R9,R11,R12,LR}
                BX LR
```

Firstly, set coordinates and print the predefined frequency message. Then, again set coordinates and print R7(current frequency)

Figure 8: A part of the current_display subroutine shown as an example

Conclusion

To conclude, in this project, an audio frequency-based stepper motor driver is implemented. All the obligatory restrictions stated in the project definition are satisfied.

- The mic is read using the ADC module
- ADC is sampled using SysTick interrupt handler (2kHz sampling)
- 256 point FFT is used to detect the frequency
- Onboard LEDs are adjusted according to the frequency
- Stepper motor is driven by GPTM interrupts
- SW1-SW2 buttons are used to adjust rotation direction (Using interrupt)
- The frequency and magnitude of the current signal are printed on the screen as well as the thresholds(frequency and magnitude).

Finally, this project gave me a chance to practice working with multiple complex sub units and understand complex setups and cooperation ıf utility modules. It gives an understanding of serial communication as well as the ARM CMSIS DPS library. Also, I had a chance to practice ADC, GPIO, GPTM, and SysTick again.