EE447 Project-Report

https://drive.google.com/file/d/1uB22u9FD-8futuUmVE-I_9wIkttgyLXa/view?usp=drivesdk

The link of the video that showing the project is working correctly was shared above.

Introduction

In this project, I build a stepper motor driver system that is responsive to the frequency of an audio input. I use the ADC module to sample an audio signal from a microphone, the ARM CMSIS DSP library to compute its frequency, and the GPTM module to drive a stepper motor. I use a Nokia 5110 LCD with an SPI module to display the current configuration and measurements. GPIO is also used for the on-board RGB LED and pushbuttons. The device continuously gathers audio signals by using a microphone and a constant sample frequency. The audio samples are saved in a 256-element array. My system calculates the audio frequency and updates the states of the output components when the array is complete. The system responds to the frequency of the sampled audio. The ARM CMSIS DSP library is used to detect the frequency. There are three output elements in the system. First, the detected frequency and its amplitude, as well as the threshold values, are presented on the LCD screen. Second, based on the measured frequency, on-board LEDs are switched on or off. The LEDs are turned off if no frequency is detected. If the detected frequency is low, the red LED illuminates. Green LED is turned on in the medium frequencies, while blue LED is turned on in the high frequencies. Finally, the stepper motor should rotate at a rate corresponding to the frequency that was last detected. The on-board switches is used to change the rotation direction.

Flow Charts & Explanations

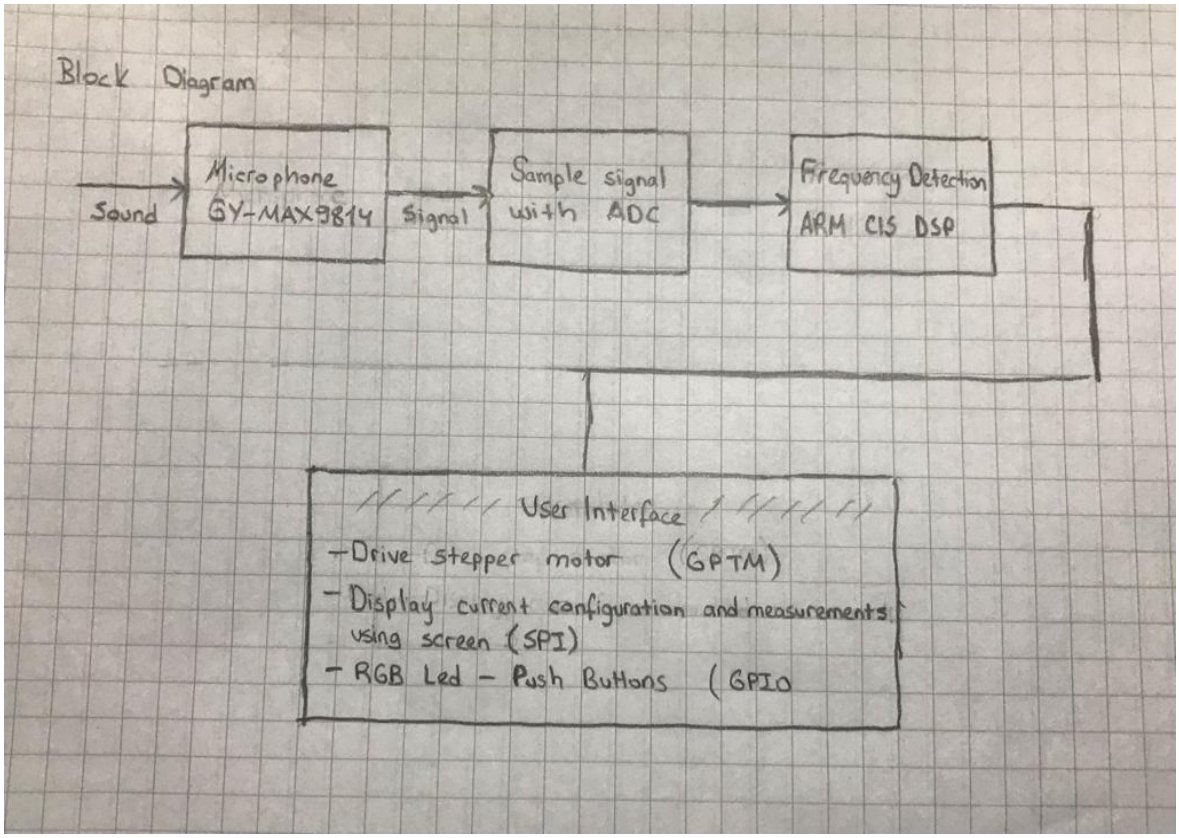


Figure 1. Block Diagram

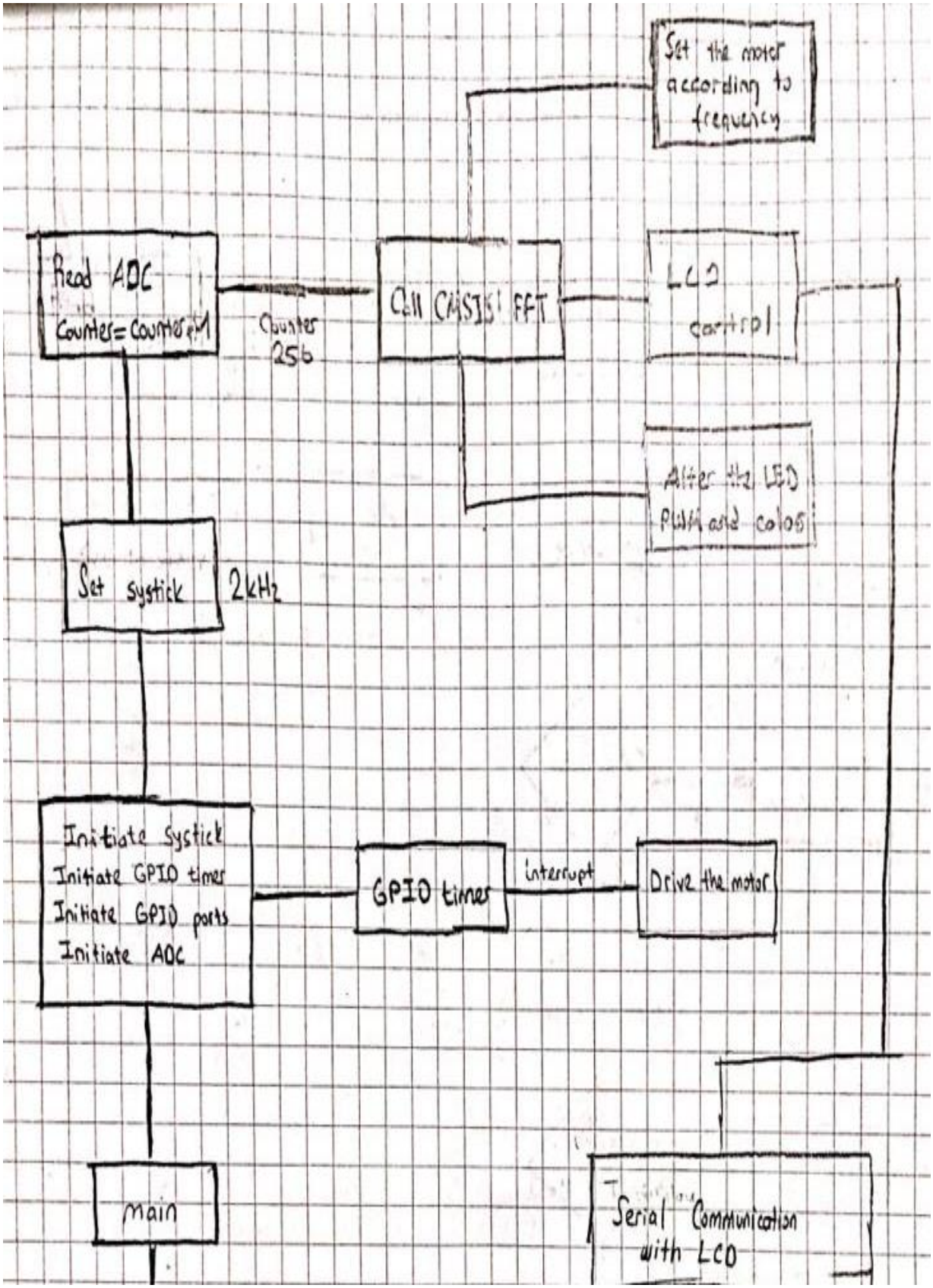
As the user interface, Iutilize a GY-MAX9814 microphone, a Nokia 5110 LCD screen, on-board LEDs, and a stepper motor. For threshold adjustment, I also use a potentiometer or a keypad. The system contains three constant thresholds: low and high frequency thresholds, as well as an amplitude threshold. If the amplitude of the dominant frequency is less than the amplitude threshold, all LEDs are turned off and the stepper motor continues to rotate at its present speed. If the amplitude of the dominant frequency is greater than the amplitude threshold, the stepper motor's speed is modified in accordance to the computed frequency. That instance, the motor rotates slowly at low frequencies and quickly at high frequencies. Furthermore, if the computed frequency is lower than the low frequency threshold, the red LED illuminates. If the estimated frequency falls between the low and high frequency thresholds, the green LED illuminates. If the estimated frequency exceeds the high frequency threshold, the blue LED will illuminates.The stepper motor has the ability to rotate in both directions.

On the screen, the user is able to view the specified higher and lower frequency and amplitude thresholds. Also, the user is able to observe the current frequency and amplitude on the screen.

The ADC module is used to read the microphone. The SysTick interrupt handler sample the microphone ADC. The sampling rate is 2 kHz. The frequency of the input will be detected using a 256-point FFT. GPTM interrupts are used to operate the stepper motor. That is, in a GPTM interrupt handler that I create, the motor signals advance one step in the selected direction each time. By deactivating the timer, I can halt the motor and alter

the speed by adjusting the reload register value. I advance steps no faster than 5 milliseconds to keep the stepper motor running smoothly.

In addition, I utilize interrupts and set the priority of the interrupts. I remember that in order to setup an interrupt, I know the interrupt number of the interrupt source I intend to utilize in order to determine which NVIC registers to modify. I read the ADC using a SysTick interrupt handler and run the stepper motor with a GPTM interrupt handler under the constraints. As a result, I conduct additional repetitious chores inside main in an unending cycle. The followings are the flow charts about what I did.


Set the motor according to frequency
Read ADC
Counter=Counter+1
Counter 256
Call CMSIS FFT
LCD control
Alter the LED Pulse and colors
Set systick
2kHz
Initiate Systick
Initiate GPIO timer
Initiate GPIO ports
Initiate ADC
GPIO timer
interrupt
Drive the motor
main
Serial Communication with LCD

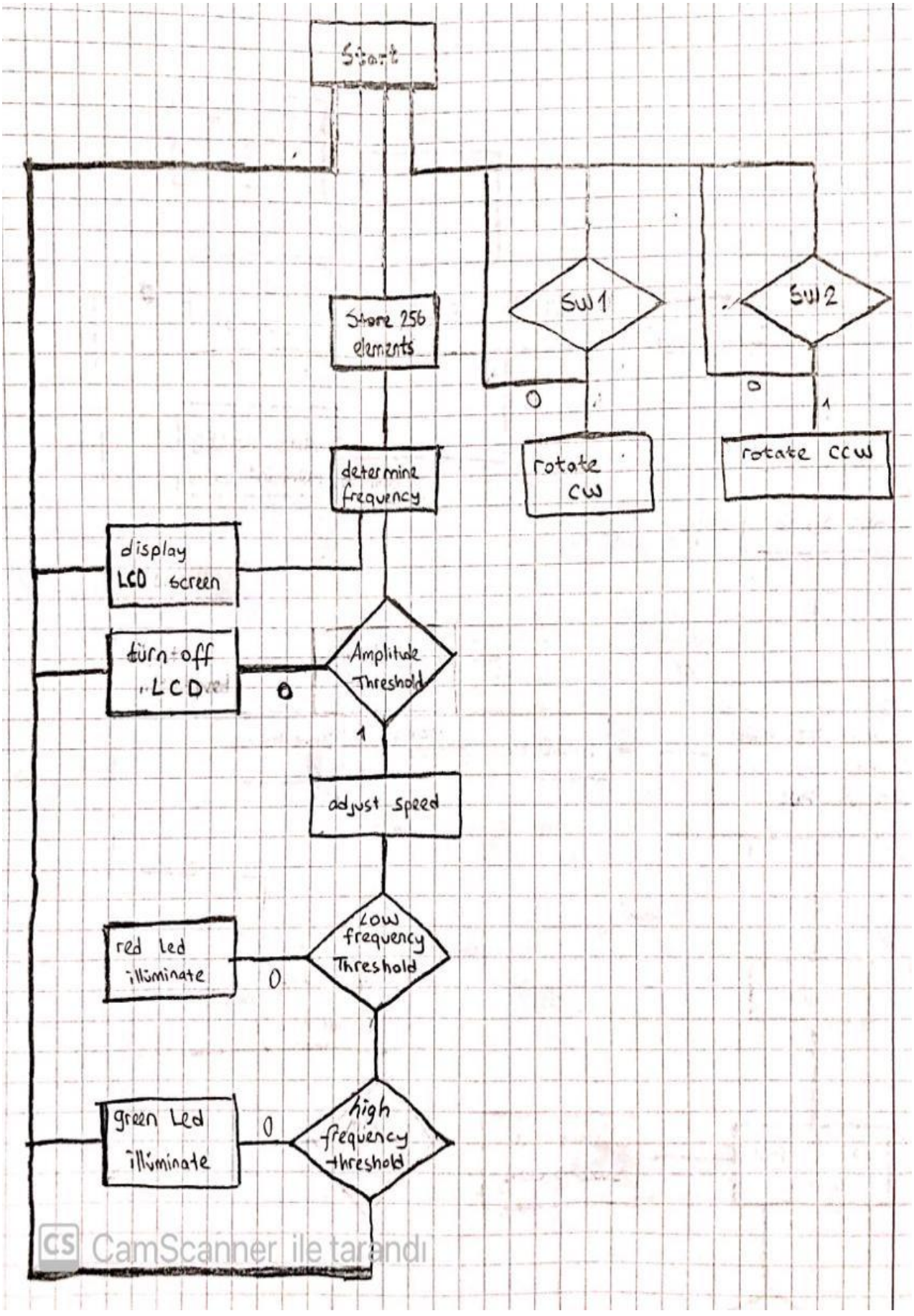


Figure 2.The Flowcharts

Sub-Modules

Microcontroller:

The TM4C123GXL board is utilized in this project. It is made up of a TM4C123GH6PM microcontroller. The TM4CGH6PM's GPIO, ADC, timer, and SSI functionalities are utilized.

Microphone:

The MAX9814 is utilized in the project. The MAX9814 is a low-cost, high-quality microphone amplifier with AGC and low-noise microphone bias. This microphone is recommended because of its qualities. It transforms electrical signals from sound waves. Its output is wired directly to the microcontroller's PE3 pin.

Timer:

TM4C123GH6PM has 6 general purpose timers. These timers can be configured as 16 bit or 32 bits. Moreover, ARM Cortex-M4F includes an integrated system timer, SysTick. SysTick provides a simple, 24-bit, clear-on-write, decrementing, wrap-on-zero counter with a flexible control mechanism. The counter can be used in several different ways. For example, in this project Microphone ADC sampling is done in SysTick interrupt handler. Sampling frequency is 2 kHz. The timer consists of three registers:

STCTRL: A control and status counter to configure its clock, enable the counter, enable the SysTick interrupt, and determine counter status.

STRELOAD: The reload value for the counter, used to provide the counter's wrap value.

STCURRENT: The current value of the counter.
My load value is 0x000007D0 which is 2000 in decimal. So, I can set the sampling frequency as 2kHz.

250ns * (Load_Value) = 1/2kHz

The board is equipped with six 16/32 bit General Purpose Timer Modules named TIMERn (They are called as GPTM in the datasheet, which refer to the same thing). Each timer module consists of two 16 bit timers (A and B). They can be used seperately as two 16-bit counters or together as a 32-bit counter. In this project, I use them seperately as 16-bit counters. TM4C Timer Modules can be used for various purposes. In this project it is used to drive the stepper motor. PB6 pin of TM4C123GH6PM is used for this purpose. Timer0 is used.

Pin Name	Pin Number	Pin Mux / Pin Assignment	Pin Type	Buffer Type[a]	Description
T0CCP0	1 28	PB6 (7) PF0 (7)	I/O	TTL	16/32-Bit Timer 0 Capture/Compare/PWM 0.
T0CCP1	4 29	PB7 (7) PF1 (7)	I/O	TTL	16/32-Bit Timer 0 Capture/Compare/PWM 1.

Figure 3. General-Purpose Timers Signals

Stepper Motor:

In this project, I build a stepper motor driver system based on an applied audio signal's frequency. After I sample an audio signal with the ADC module using a microphone and calculate that signal's frequency using ARM CMSIS DSP library, I drive a stepper motor using the GPTM module.

If the dominant frequency's amplitude is lower than the amplitude threshold, the stepper motor continues to its rotation at its current speed. If the dominant frequency's amplitude is higher than the amplitude threshold, stepper motor's speed is adjusted in proportion to the calculated frequency. That is, the motor rotates slowly in low frequencies and fast in high frequencies. Stepper motor is driven using GPTM interrupts. That is, the motor signals advance one step in the determined direction every time in a GPTM interrupt handler that I configure.

A stepper motor is a type of electromagnetic device that translates digital pulses into mechanical shaft rotation. Step motors have the advantages of low cost, excellent dependability, high torque at low speeds, and a simple, sturdy architecture that can function in practically any environment. The primary drawbacks of utilizing a stepper motor are the resonance effects that are frequently observed at low speeds and the decreased torque with rising speed.

Stepper motors are treated as black boxes in this project, converting digital pulses into mechanical shaft rotation. The methodology for this conversion is conventional and straightforward. The stepper motor, like most other types of motors, is built with coils that convert electrical current to magnetic force to rotate the motor shaft.

In terms of driving circuitry, the motor may demand higher currents than other interface devices. This need may not be met by the current output of standard voltage outputs. Separate motor driver ICs are utilized for similar purposes. I'm using ULN2003A for this project. The ULN200xA devices are Darlington transistor arrays with high voltage and current. The ULN2003A is commonly used in driver circuits for relays, lamp and LED displays, stepper motors, logic buffers, and line drivers.

ADC:

Microphone is read using the ADC module. 2. Microphone ADC sampling is done in SysTick interrupt handler. Sampling frequency is 2 kHz. Under the given restrictions, I read the ADC using SysTick interrupt handler. Then, I give an array of 256 complex numbers. My input sequence is real valued. I store the 16-bit ADC readings in the real parts of the complex numbers, and the imaginary parts are zero.

A digital-to-analog converter (ADC) is a device that transforms a continuous analog voltage to a discrete digital number. Two identical converter modules with a total of 12 input channels are supplied. The TM4C123GH6PM ADC module offers 12 input channels and has a 12-bit conversion resolution, as well as an inbuilt temperature sensor. Each ADC module comprises four programmable sequencers that enable the sampling of various analog input sources without the involvement of a controller. Each sample sequencer has

completely adjustable input sources, trigger events, interrupt generation, and sequencer priority.

Led:

I utilize the LEDs that were present on the TM4C123GH6PM. This procedure makes use of the PF1, PF2, and PF3 pins. As mentioned in project manual:

If the dominant frequency's amplitude is lower than the amplitude threshold, all LEDs are off.

If the calculated frequency is lower than the low frequency threshold, red LED is on.

If the calculated frequency is between the low and high frequency thresholds, green LED is on.

If the calculated frequency is higher than the high frequency threshold, blue LED is on.

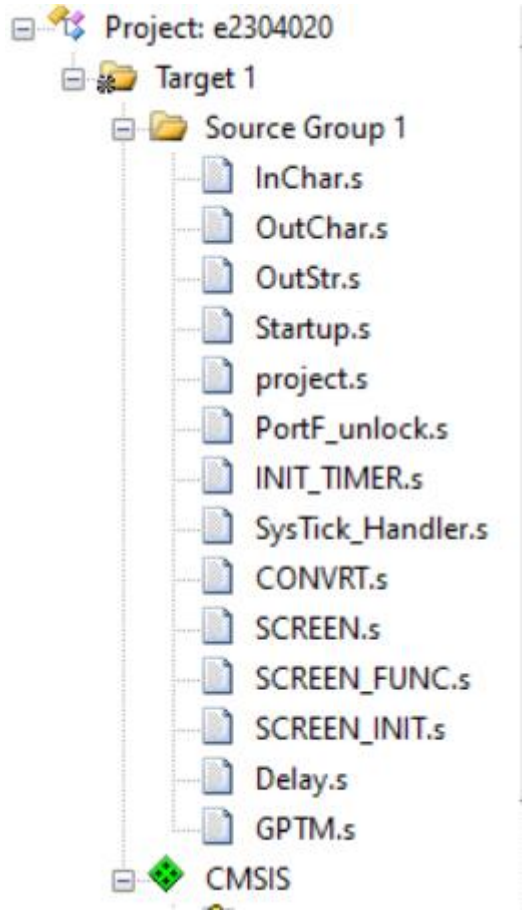
Screen:

The user is able to see the configured upper and lower frequency and amplitude thresholds on the screen. Also, the user is able to see the current frequency and amplitude on the screen.

For GPIO and SPI configuration parts, I refer to page 965 of the [2] for details and 967 for register map. For Nokia 5110 LCD configuration I refer to [4] provided along with the project manual, where a detailed version of the instruction set and data transmission graphs are provided for clarification.

Four Synchronous Serial Interface (SSI) modules are included with the TM4C123GH6PM microcontroller. Each SSI module serves as a master or slave interface for synchronous serial communication with peripheral devices that use SPI, MICROWIRE, or Texas Instruments synchronous serial interfaces. To get the Nokia 5110 to operate, three distinct configurations are necessary. The first is GPIO, where the relevant I/O pins are set to work as SPI pins. The SPI module on the TM4C123 is set to be LCD compatible in the second half. Finally, the display is initialized in the NOKIA 5110 setup section allowing communicating and receiving data to be shown. The Nokia 5110 receives commands, text, and graphics to display through SPI signals. The SPI protocol indicates that there is only one data output signal. The LCD must somehow differentiate whether the data being provided is data to be shown or a command to operate the panel. This is accomplished "manually" via a GPIO pin rather than using the SPI module. A Data/Command pin is located on the Nokia screen (DC). The Nokia interprets the incoming SPI bits as a command when the DC signal is low. Similarly, if the DC signal is high, the SPI bits are read as display data. Long before the last bit is delivered, the DC signal might be made high or low.

Important Codes



Project.s includes my main and microphone initilizations are done here.

PortF_unlock.s is used to SW1 & SW2 & RGB LEDs

INIT_TIMER.s is used to use Systick Timer.

SysTick_Handler is used as a interrupt service routine of Systick Tımer.

Delay.s makes 100ms delay.

GPTM is used to initialize the GPTM and necessary pins for the rotation of the stepper motor.

CONVRT.s converts an m-digit decimal number represented by n bits (n < 32) in register R4 into such a format that the ASCII codes of the digits of its decimal equivalent would listed in the memory starting from the location address of which is stored in register R5.

```
41                  LDR R5,=ADC0_ADCSSFIFO3
42                  LDR R0,[R5] ; fifo val in the R0
43
44
45                  LDR R9,=COUNTER
46                  LDR R8,[R9]
47                  MOV R7,#0
48                  LDR R6,=0x20000800
49                  CMP R8,#512
50                  BEQ fft
51                  STRH R0,[R1,R8]
52                  ADD R8,#2
53                  STRH R7,[R1,R8]
54                  ADD R8,#2
55                  STR R8,[R9]
56
```

```
59  fft
60                      LDR R1,=STCTRL
61                      MOV R2,#0X00
62                      STR R2,[R1]
63                      PUSH{LR}
64                      LDR R0,=arm_cfft_sR_q15_len256
65                      LDR R1,=0x20000400
66                      MOV R2,#0
67                      MOV R3,#1
68                      BL  arm_cfft_q15
69                      POP{LR}
70
```

```
82  square
83                        CMP R7,#128
84                        BEQ ENDOFFFT
85                        ADD R7,#1
86                        LDRSH R10,[R5],#2
87                        MUL R6,R10,R10
88                        LDRSH R10,[R5],#2
89                        MUL R11,R10,R10
90                        ADD R6,R11
91                        CMP R4,R6
92                        movlt R4,R6
93                        movlt R8,R7
94                        LDR R6,=AMPL
95                        STR R4,[R6]
96                        STR R8,[R9]
97                        B square
```

Figure 4. Some piece of code from SysTick_Handler.s

Arm_cfft_q15 function takes the input and gives the output as complex numbers. Real and imaginary parts of input is in q15 and those of output is in q9.7 fixed point formats. In this project, I assume that both q15 and q9.7 formats are the same as 16-bit signed 2's complement numbers. That is, 0x7FFF is the maximum value and 0x8000 is the minimum value. A single complex number in this format takes up 4 bytes. The first 2 bytes are the real part, and the last 2 bytes are the imaginary part. As the input, I give an array of 256 complex numbers. My input sequence is real valued. I store the 16-bit ADC readings in the real parts of the complex numbers, and the imaginary parts ARE zero. Arm_cfft_q15 function operatSe on the array in-place. That is, when the function returns, output data will be placed on the same memory location as the input data and input data will be lost. To be able to use CMSIS DSP functions, I add the source files to my project. The function uses a constant table, arm_cfft_sR_q15_len256, which is stored in the CMSIS DSP library. The function takes 4 input arguments [4]. Before calling the function, I store the table's address in R0, your array's address in R1, '0' in R2 and '1' in R3. I found and store the frequency and amplitude in here.

```
46
47                  cmp     r10,#0
48                  BNE     CCW
49  CW              LDR     R1,=PB_OUT
50                  LDR     R0,[R1]
51                  BIC     r0,#0xf0
52                  LSL     R0,#1
53                  CMP     R0,#0X10
54                  MOVEQ   R0,#0X1
55                  STR     R0,[R1]
56                  B       EXIT
57  CCW             LDR     R1,=PB_OUT
58                  LDR     R0,[R1]
59                  LSR     R0,#1
60                  CMP     R0,#0X0
61                  MOVEQ   R0,#0X08
62                  STR     R0,[R1]
```

Figure 5. Some piece of code from GPTM.s

I rotate the stepper motor in here.

```
150                     BL  PortF_unlock
151                     POP {LR}
152                     CMP R6,#0x7000
153                     BPL LEDRGB
154                     LDR R1,=GPIO_PORTF_DATA
155                     MOV R3,#0x00
156                     STR R3, [R1]
157                     MOV R8,#50
158                     B finish
159 LEDRGB
160                     CMP R4, #0x3A
161                     BPL LEDGB
162                     LDR R1,=GPIO_PORTF_DATA
163                     MOV R3,#0x02
164                     STR R3, [R1]
165                     MOV R8,#8
166                     B finish
167 LEDGB
168                     CMP R4, #0x5A
169                     BPL LEDG
170                     LDR R1,=GPIO_PORTF_DATA
171                     MOV R3,#0x08
172                     STR R3, [R1]
173                     MOV R8,#6
174                     B finish
175 LEDG
176                     LDR R1,=GPIO_PORTF_DATA
177                     MOV R3,#0x04
178                     STR R3, [R1]
179                     MOV R8,#3
```

Figure 6. Some piece of code from SysTick_Handler.s

I illuminate the LEDs in here.

If the calculated frequency is lower than the low frequency threshold, red LED is on.

If the calculated frequency is between the low and high frequency thresholds, green LED is on.

If the calculated frequency is higher than the high frequency threshold, blue LED is on.

```
333                     EXTERN  MY_SysTick_Handler
334  SysTick_Handler    PROC
335                     EXPORT  SysTick_Handler          [WEAK]
336                     B       MY_SysTick_Handler
337                     ENDP
475                     EXTERN  My_Timer1A_Handler
476  Timer0A_Handler    PROC
477                     EXPORT  Timer0A_Handler          [WEAK]
478                     B       My_Timer1A_Handler
479                     ENDP
```

Figure 7. Some piece of code from Startup.s

I am stating that I will use the SysTick and GPTM in here.

```
32                  MOV     R0,#0X0
33                  MOV     R1,#0X0
34                  BL      SCR_XY          ;SET CURSOR TO THE BEGINNING
35
36                  MOV     R5,#0X46        ;f
37                  BL      SCR_CHAR
38                  MOV     R5,#0X72        ;r
39                  BL      SCR_CHAR
40                  MOV     R5,#0X3A        ;:
41                  BL      SCR_CHAR
42                  MOV     R5,#0X34        ;4
43                  BL      SCR_CHAR
44                  MOV     R5,#0X35        ;5
45                  BL      SCR_CHAR
46                  MOV     R5,#0X30        ;0
47                  BL      SCR_CHAR
48                  MOV     R5,#0X2D        ;-
49                  BL      SCR_CHAR
50                  MOV     R5,#0X37        ;7
51                  BL      SCR_CHAR
52                  MOV     R5,#0X30        ;0
53                  BL      SCR_CHAR
54                  MOV     R5,#0X30        ;0
55                  BL      SCR_CHAR
56                  MOV     R5,#0X20        ;SPACE
57                  BL      SCR_CHAR
58
59                  MOV     R0,#0X49
60                  MOV     R1,#0X0
61                  BL      SCR_XY          ;SET CURSOR TO THE LEFT OF THE ROW
62
63                  MOV     R5,#0X48        ;H
64                  BL      SCR_CHAR
65                  MOV     R5,#0X7A        ;Z
```

Figure 8. Some piece of code from SCREEN.s

In SCREEN.s, as you can see, first I specify the coordinate and then according to ASCII values I write the outputs on the screen.

PIN Assignments

TM4C123GH6PM GND - Nokia 5110 GND – Microphone GND – Stepper Motor GND

TM4C123GH6PM 3.3V - Nokia 5110 Vcc - Nokia 5110 BL – Microphone Vcc

TM4C123GH6PM Vbus - Stepper Motor Vcc

TM4C123GH6PM PA7 - Nokia 5110 RST

TM4C123GH6PM PA3 - Nokia 5110 CE

TM4C123GH6PM PA6 - Nokia 5110 DC

TM4C123GH6PM PA5 - Nokia 5110 DIN

TM4C123GH6PM PA2 - Nokia 5110 CLK

TM4C123GH6PM PB0 - Stepper Motor IN1

TM4C123GH6PM PB1 - Stepper Motor IN2

TM4C123GH6PM PB2 - Stepper Motor IN3

TM4C123GH6PM PB3 - Stepper Motor IN4

TM4C123GH6PM PE3 – Microphone Out

Overall Setup and LCD Screen Output

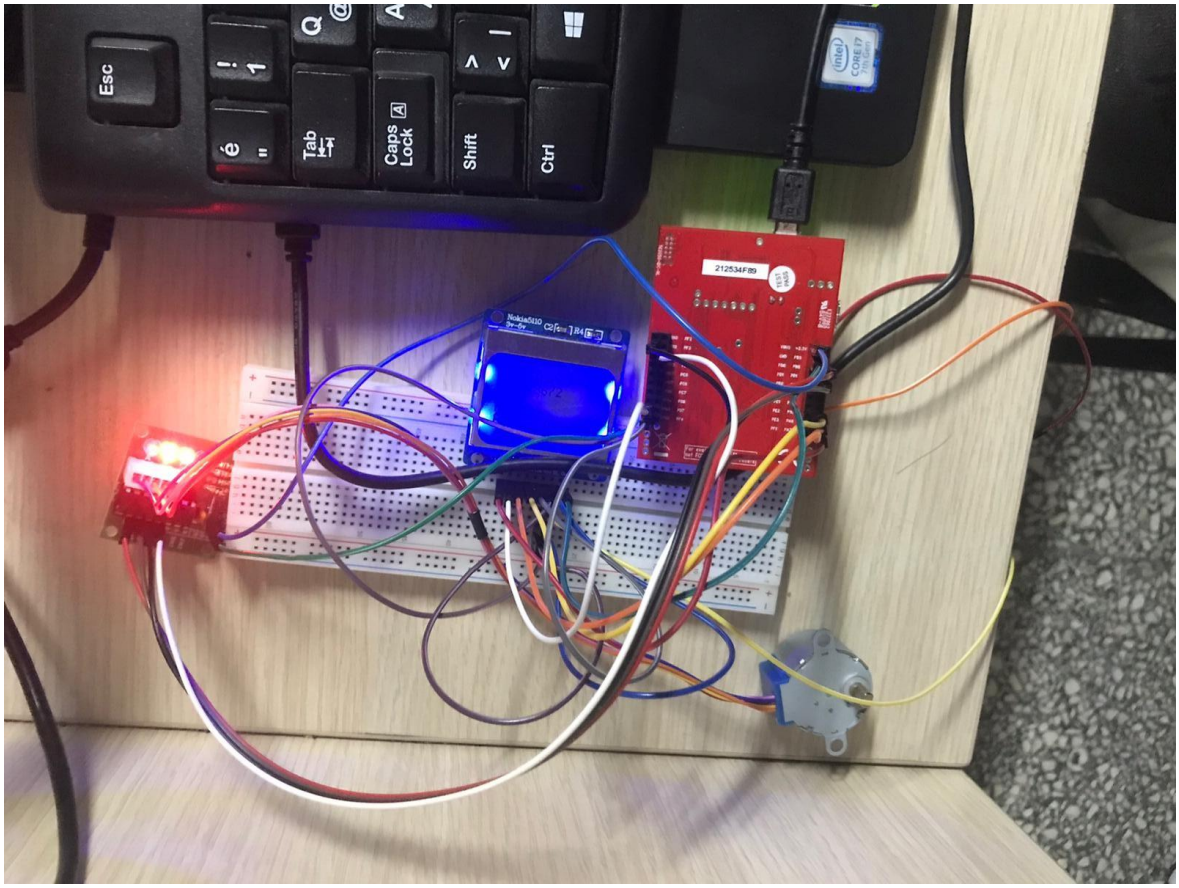


Figure 9. Overall Setup

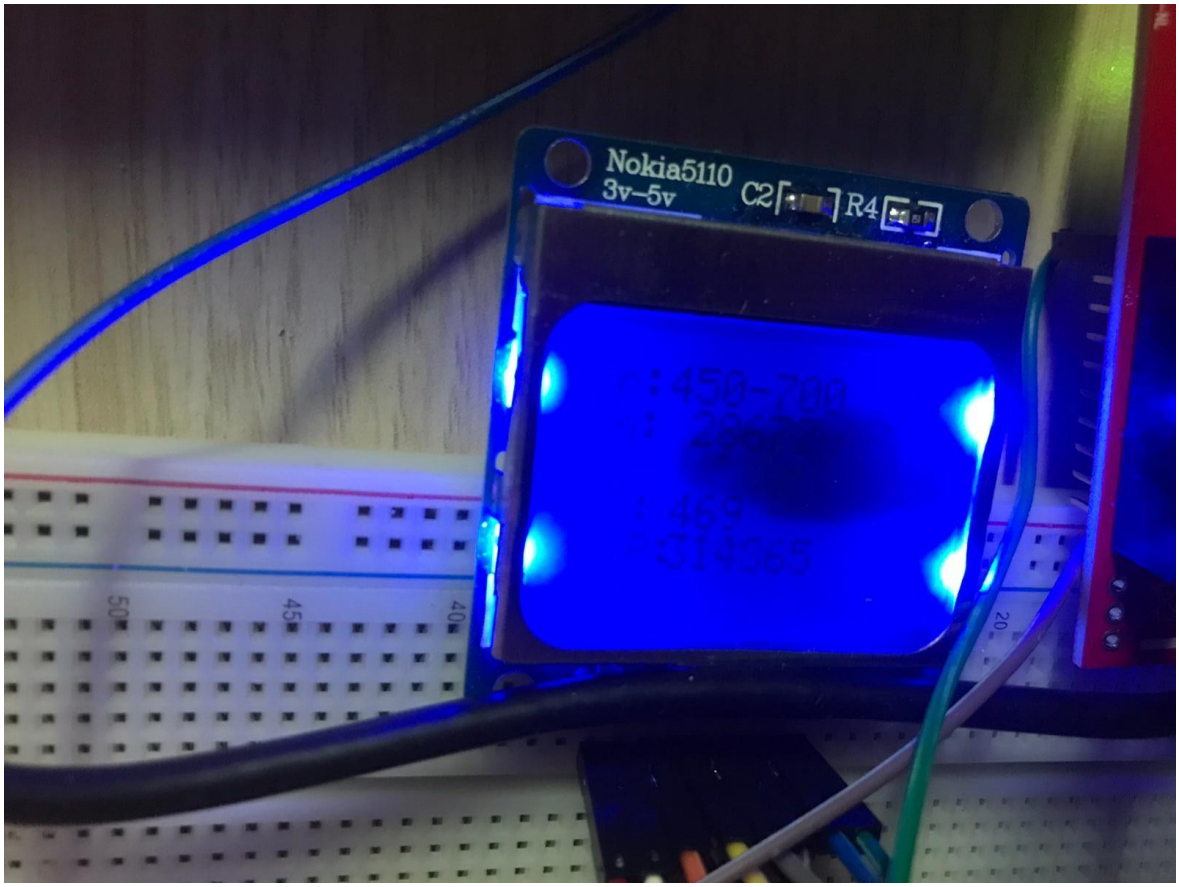


Figure 10. LCD Screen Output

Conclusion

I learned how to sample audio signal using a microphone, at a constant sampling frequency and store the audio samples in an array of 256 elements. I understood the concepts of stepper motor, LCD screen, LED and realized what was desired by using them. I discovered to usages of timers. I learned about the ARM CMSIS DSP Library and the Serial Peripheral Interface. I became acquainted with the CMSIS FFT Implementation and comprehended the Input and Output Format. I learnt how to set up the LCD and SPI on a Nokia 5110. The ideas of Vertical Addressing Mode and Horizontal Addressing Mode are investigated.

References

[1] EE447 Lecture Notes

[2] TM4C123GH6PM_Datasheet

[3] TM4C123GXL_Users_Manual

[4] Nokia5110Datasheet

[5] https://www.youtube.com/channel/UC40tcn7Ezg603M9ZZt4wQqg

[6] https://github.com/efeyitim/EE447-Introduction-to-Microprocessors

Appendix I

Total time spent on/during Report : 7 hours

https://drive.google.com/file/d/1uB22u9FD-8futuUmVE-I_9wIkttgyLXa/view?usp=drivesdk