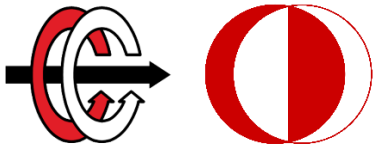


MIDDLE EAST TECHNICAL UNIVERSITY

DEPARTMENT OF ELECTRICAL AND ELECTRONICS ENGINEERING

EE447 INTRODUCTION TO MICROPROCESSORS

LABORATORY PROJECT

AUDIO FREQUENCY BASED STEPPER MOTOR DRIVER

BAHRİ BATUHAN BİLECEN

2374635

6 February 2022

Table of Contents

1. Introduction

In this project, a frequency-based step motor controller is designed using EK-TM4C123GXL LaunchPad Tiva board, Nokia 5510 LCD, stepper motor and its controller, and a microphone module. Programming is done with ARM assembly, using Thumb2 instruction set.

This report summarizes the critical design points for each requirement and restriction given in the project manual. It also provides visual test results. Since this report is a summary, a well-commented source code is provided alongside for more detail.

2. Requirements & Restrictions

In this section, requirement and restrictions given in the project description are summarized. Their solution approaches are to be explained in detail in the following sections.

Requirements of the project are as follows:

1. Three threshold values should be set: low-high frequency thresholds and a magnitude threshold. LEDs should behave accordingly to the magnitude and frequency thresholds **(see 3.2).**
2. Stepper motor should be able to rotate in two directions, toggled by two on-board buttons. Motor speed should be determined by the current frequency value and keep its speed when frequency magnitude drops below the threshold **(3.3).**
3. LCD should display threshold values and current magnitude & frequency values **(3.4 and 4).**

Restrictions of the implementation are as follows:

1. Microphone reading must be done by ADC module, using SysTick handler, with 2kHz sampling frequency **(3.2.1).**
2. 256-point DFT of the input signal must be calculated by FFT **(3.2.2).**
3. Stepper motor control must be done by Timer handler **(3.3.1).**
4. Button control must be done by GPIO interrupt flags **(3.3.2)**.

3. Implementation Details

3.1. Code Hierarchy and Pin Layout

Main routine is kept simple and most of the operations are done in nested subroutines and interrupt subroutines.

As a design choice, registers from R4 to R10 are used as “global” variables, meaning that they never get used for another purpose or get pushed to stack. Interrupt subroutines do not push R4-R10 to the stack, either. R0-R3 are used as temporary registers inside the subroutines.

```
MOV     R4,#0           ;Motor flag register.
                        ;R4[0] = Motor rotation flag. SW1 pressed -> 1, SW2 pressed -> 0
                        ;R4[31:16] = Motor rotation mask (16 bit)
MOVT    R4,#0x0001      ;Move initial mask to R4[31:16]

MOV     R5,#0           ;samplecount
                        ;When 256 samples are taken, FFT is called

MOV     R6,#63000       ;TIMER0A initial interval load value
                        ;During the program execution, R6 will hold the load value and get updated
                        ;depending on the microphone input frequency inside findFundFreq subroutine

MOV     R7,#0           ;Fundamental frequency magnitude

MOV     R8,#0           ;Corresponding index to the fundamental frequency magnitude.
                        ;In findFundFreq subroutine, it is then changed to the actual frequency value (7-992Hz).

MOV     R9,#0           ;LCD-related register.
                        ;R9[8] = D/C flag, 1 if D
                        ;R9[3:0] = Data to be written to SSI_DR

LDR     R10,=0x20000804 ;String addresses for R7 and R8 after passing through CONVRT subroutine.
```

Figure 1. Initializations and descriptions of R4-R10.

```
contSample  CMP     R5,#1020        ;At each measurement write, samplecount += 4 (+2 for real, +2 for imag)
            BHS     computeFFT      ;Then for 256 meausurements, 256*4 = 1028
            B       contSample      ;R5 starts from 0, so jump to fft when 1028-8=1020

computeFFT  LDR     R0,=arm_cfft_sR_q15_len256
            LDR     R1,=0x20000400
            MOV     R2,#0
            MOV     R3,#1

            CPSID   I               ;Interrupts inside cfft subroutine cause issues
            BL      arm_cfft_q15
            CPSIE   I

            BL      findFundFreq
            BL      updateScreen
            BL      updateLED

            MOV32   R2,#1200000     ;Delay
loop        SUBS    R2,#1
            BNE     loop

            B       contSample

            ENDP
            ALIGN
            END
```

Figure 2. Main routine, without the initializations.

Table 1. Planned pin layout.

	Board		Peripheral	Description/Notes
	Pin mux	Pin function	Pin name	
LCD Screen	PA2	SSI0Clk	CLK	SPI0 clock
	PA3	SSI0Fss	CE	SPI0 frame signal
	PA5	SSTI0Tx	DIN	SPI0 transmit
	PA6	GPIO	DC	Data/command bar
	PA7	GPIO	RST	External reset
	3.3V		VCC	
	GND		GND	
Stepper Motor Controller	PB0	GPIO	IN1	
	PB1	GPIO	IN2	
	PB2	GPIO	IN3	
	PB3	GPIO	IN4	
	VBUS		+	
	GND		-	
Tiva Board Components	PF0	GPIO		Button, unlock the pin
	PF1	GPIO		Red LED
	PF2	GPIO		Blue LED
	PF3	GPIO		Green LED
	PF4	GPIO		Button
Microphone	GND		GND	
	3.3V		Vdd	
	PE3	AIN0	Out	Remove 1.25V offset

3.2. Microphone, Sampling and Thresholds

3.2.1. SysTick

ADC0 & SS3 with FIFO size 1 is used for the sampling procedure. Lowest sample rate ADC module allows is 125ksps despite we require 2ksps. Hence, we use SysTick with $f_{sampling} =$ 2kHz frequency to generate our own desired sampling rate.

Considering that PIOSC/4 = 4MHz is used for SysTick CLK, $reload$ value of SysTick should be:

$$\frac{reload + 1}{4 * 10^6} = \frac{1}{2 * 10^3} \rightarrow reload = 1999$$

Whenever reload=0, the program branches to SysTick handler ISR. Inside the ISR, data from the FIFO is gathered, 1.25V offset (0x60F = 1551) is removed and written to the real part of the memory location, with imaginary parts 0x00. This is repeated for 256 times since we will calculate 256-point FFT later.

3.2.2. FFT

After gathering all required data from the microphone, 256-point FFT of the sound signal is calculated by arm_cfft_sR_q15_len256. Interrupts are temporarily disabled before branching to FFT subroutine to avoid hard faults.

Then, linear search is performed for the first 128 results to find fundamental frequency and its magnitude. Since the input signal is real, FFT will be conjugate symmetric. This means only the bins [1,127] will have unique information contributing to the magnitude response. 0th

bin will indicate $0Hz$, 1st bin will indicate $7Hz$ and 127th bin will indicate $\frac{f_{sampling}}{2} - 1 - 7 = \frac{2kHz}{2} - 8 = 992Hz$.

3.2.3. Thresholds

Low and high frequency thresholds are determined as 256Hz and 768Hz, respectively.

Magnitude is found by $\mathcal{F}(Re(input)^2 + Im(input)^2)$, without the square root. Threshold is empirically determined as 1000.

In a separate subroutine, LED colors are updated according to the current frequency & magnitude value and thresholds.

3.3. Stepper Motor Control

3.3.1. TIMER Module

TIMER0A with 4MHz CLK (3-bit prescaling) with its GPTM ISR is used for step motor control. Bit shifting according to the rotation direction, updating rotation direction according to button presses, and reloading TAILR register according to the frequency value is done in TIMER0A ISR.

TAILR maximum (slowest rotation) and minimum (fastest rotation) values are determined as 63000 and 8000, empirically. Considering that frequency values range from 7-992Hz, a linear mapping is used for in-between TAILR values:

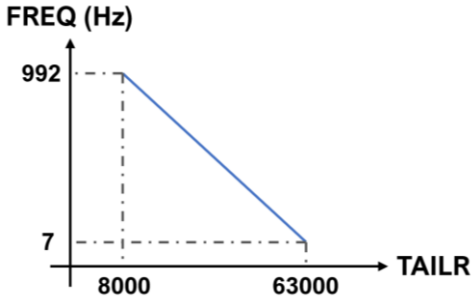


Figure 3. Linear relationship between FREQ and TAILR.

With several approximations, $TAILR = FREQ * \frac{1000}{18} + 63500$ is used as the mapping.

3.3.2. On-Board Buttons

Two buttons are used as on-board buttons, where one of them had to be unlocked before using. To understand whether buttons are pressed or not, relevant GPIO interrupt registers are used (IS, IBE, IV, IM, etc.). Edge sensitive & falling edge options are selected, and relevant interrupts are enabled.

Inside the GPIO ISR, to avoid debouncing and distinguish two buttons, data registers of the pins are utilized. If an invalid data register configuration is read inside the ISR, then it is considered as bouncing and program is branched outside the ISR immediately. By this, rotation of the motor is only changed when the button is released and polling the data register in the main routine is avoided.

3.4. LCD Control

LCD is controlled by SSI0 module. In the datasheet of the LCD, it is stated that clock cycle SCLK period needs to be at least 250ns [1]. Hence, SSI0Clk is set to $2.6MHz = \frac{16}{2*(1+2)} MHz$ with prescaler of 2. By this, T = 375ns > 250ns is satisfied with enough error margin.

Most of the initialization procedure is taken from mhuangwm's Nokia5510 library [2] and converted into ARM assembly.

Each character in the LCD is represented by 5 columns, each of a byte. A corresponding lookup table [2] is written into read only memory:

```
                AREA    char_table, DATA, READONLY, ALIGN=2
CHARS           DCB     0x00, 0x00, 0x00, 0x00, 0x00 ; 20 SPACE
                DCB     0x00, 0x00, 0x5f, 0x00, 0x00 ; 21 !
                DCB     0x00, 0x07, 0x00, 0x07, 0x00 ; 22 "
                DCB     0x14, 0x7f, 0x14, 0x7f, 0x14 ; 23 #
                DCB     0x24, 0x2a, 0x7f, 0x2a, 0x12 ; 24 $
                DCB     0x23, 0x13, 0x08, 0x64, 0x62 ; 25 %
                DCB     0x36, 0x49, 0x55, 0x22, 0x50 ; 26 &
                DCB     0x00, 0x05, 0x03, 0x00, 0x00 ; 27 '
                DCB     0x00, 0x1c, 0x22, 0x41, 0x00 ; 28 (
                DCB     0x00, 0x41, 0x22, 0x1c, 0x00 ; 29 )
                DCB     0x14, 0x08, 0x3e, 0x08, 0x14 ; 2a *
                DCB     0x08, 0x08, 0x3e, 0x08, 0x08 ; 2b +
                DCB     0x00, 0x50, 0x30, 0x00, 0x00 ; 2c ,
                DCB     0x08, 0x08, 0x08, 0x08, 0x08 ; 2d -
                DCB     0x00, 0x60, 0x60, 0x00, 0x00 ; 2e .
                DCB     0x20, 0x10, 0x08, 0x04, 0x02 ; 2f /
                DCB     0x3e, 0x51, 0x49, 0x45, 0x3e ; 30 0
                DCB     0x00, 0x42, 0x7f, 0x40, 0x00 ; 31 1
                DCB     0x42, 0x61, 0x51, 0x49, 0x46 ; 32 2
                DCB     0x21, 0x41, 0x45, 0x4b, 0x31 ; 33 3
                DCB     0x18, 0x14, 0x12, 0x7f, 0x10 ; 34 4
```

Figure 4. CHARS lookup table. Each character is of 5 bytes.

When a string of ASCII characters is given (whether it be hard coded by DCB command, or a number inside a register converted into its ASCII characters and stored into memory in run-time) with accompanying end of transmission character (0x04), it can be decoded and displayed in LCD by using suitable subroutines, and the CHARS array with correct offsets.

```
                AREA    char_table, DATA, READONLY, ALIGN=2
FREQ            DCB     "FREQ (HZ)  : "
                DCB     0x04    ;end of transmission

MAG             DCB     "MAG: "
                DCB     0x04

BREAK           DCB     "============"
                DCB     0x04

T_FREQ_L        DCB     "FREQ_LO(HZ): 256"
                DCB     0x04

T_FREQ_H        DCB     "FREQ_HI(HZ): 768"
                DCB     0x04

T_MAG           DCB     "MAG_THRESH :1000"
                DCB     0x04
```

Figure 5. Hard coded strings to be printed on the screen.

Only the sections of the screen where magnitude & frequency values are located are refreshed (cleared & re-written) each time. Hardcoded strings are only printed once.

4. Test Results

An online tone generator [3] is used for the tests. It is observed that sinusoidal waves give the most accurate results compared to square and sawtooth. This is due to the sampling rate.

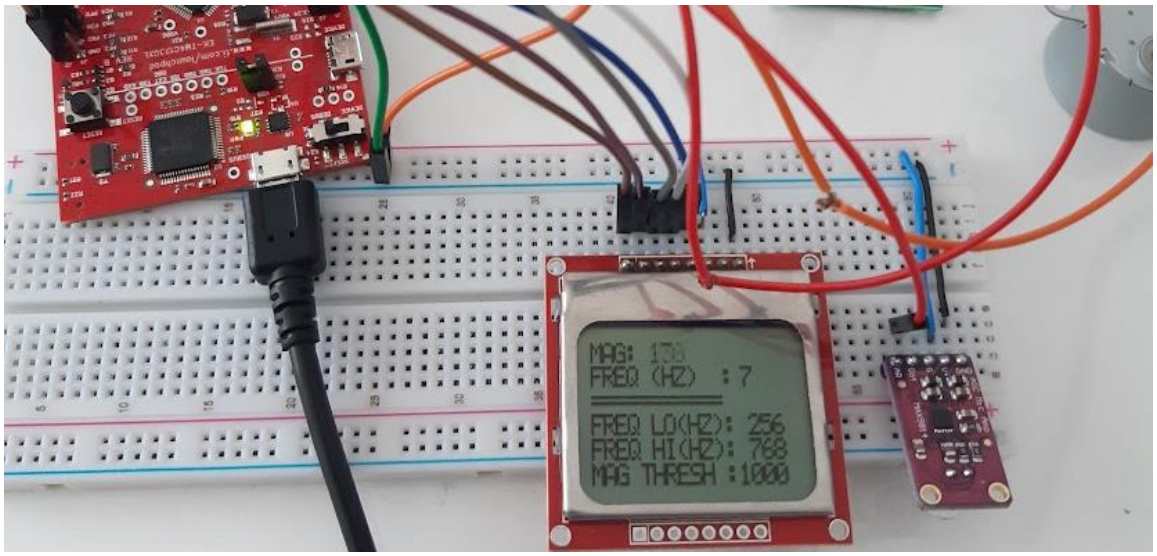
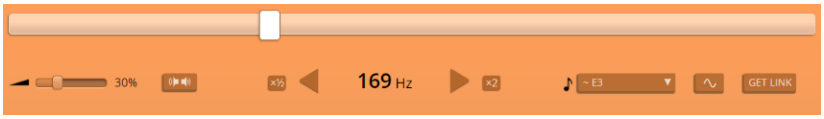


Figure 6. Setup with ambient noise.



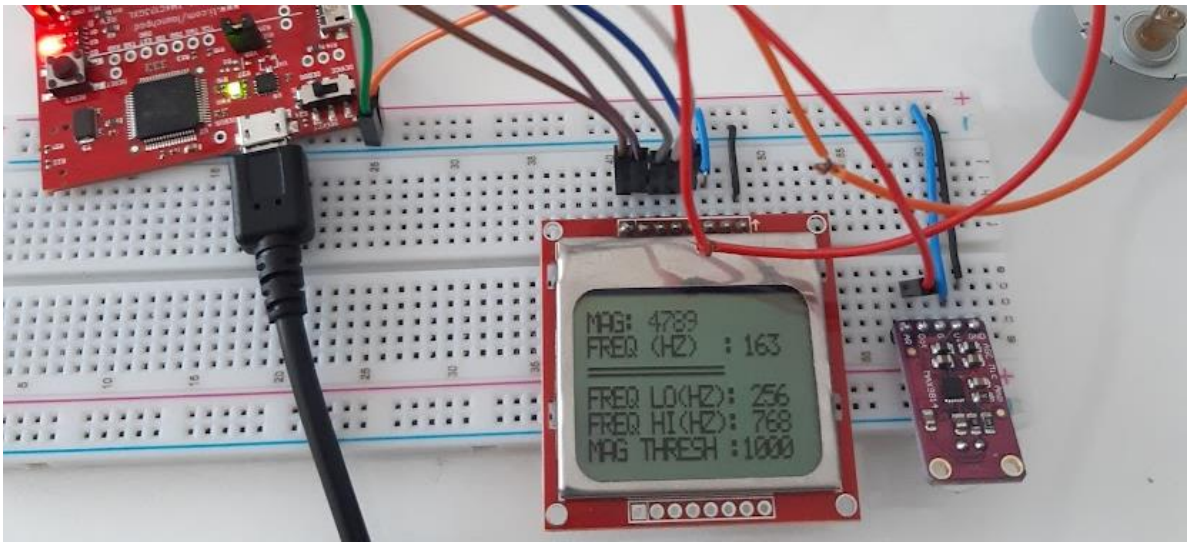


Figure 7. Setup with f=169Hz input. Red LED is on.

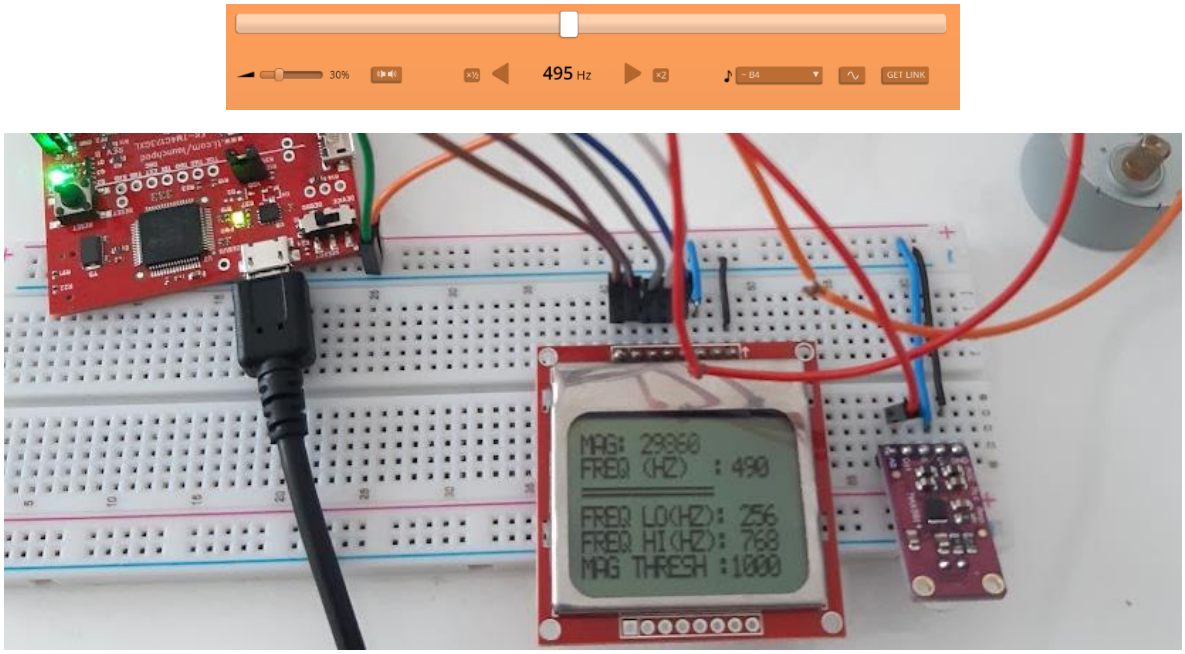


Figure 8. Setup with f=495Hz input. Green LED is on.

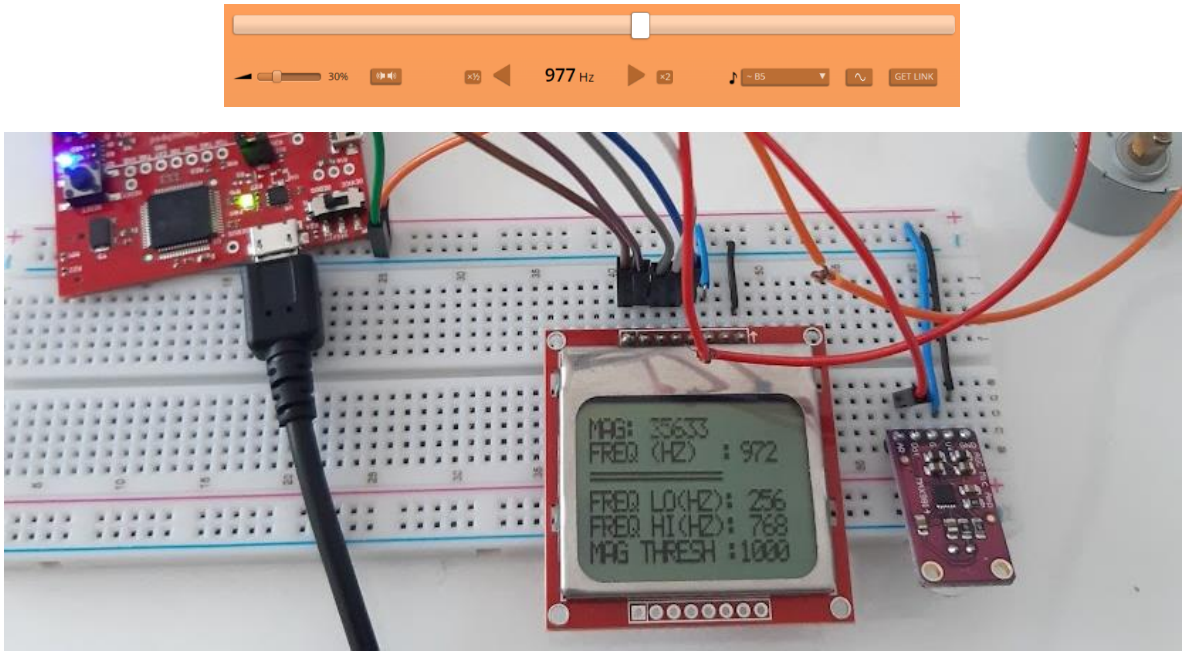


Figure 9. Setup with f=977Hz input. Blue LED is on.

As seen, there are -+5 Hz offsets, which are insignificant.

5. Conclusion

In this term project, EK-TM4C123GXL LaunchPad Tiva board is used with various peripherals, exploring its capabilities and control mechanisms. A frequency-based stepper motor controller was constructed and shown with tests that it satisfies the requirements.

The project was a useful and fun experience to understand how a microprocessor operates and most of its features. It was also a good assembly coding exercise.

6. References

- [1] Nokia 5510 Datasheet (on METUClass)
- [2] https://github.com/mhuangwm/TM4C123-Nokia5110-Library
- [3] https://www.szynalski.com/tone-generator/