EE447 Project

Final Report

Hakan Emre Gedik

2304657

Introduction

In this project, concepts studied in the lectures such as timers, interrupts, ADC, etc. We are to drive a step motor and adjust its rotation speed by calculating the frequency of the input sound signal. We have a user interface which consists of onboard LEDs and an LCD screen. Main project parameters are low frequency threshold, high frequency threshold, and amplitude threshold. The input signal is only considered for the adjustment of LEDs and the rotation speed of the step motor if the most dominant frequency of it exceeds the amplitude threshold. LCD displays the project parameters, and the current dominant frequency and its amplitude. If the dominant frequency is lower than the lower frequency threshold, red LED lights up. If it exceeds the maximum frequency threshold, the blue led lights up. Otherwise, the green LED is turned on.

Physical Construction

The project consists of a microphone, step motor, step motor driver, keyboard, potentiometer, LCD screen, and a TI launchpad. With keyboard, low and high frequency thresholds can be adjusted. Potentiometer is used for amplitude threshold adjustment. Microphone is used for acquiring sound signals, which later FFT of this sound signal will be taken and the dominant frequency will be determined.

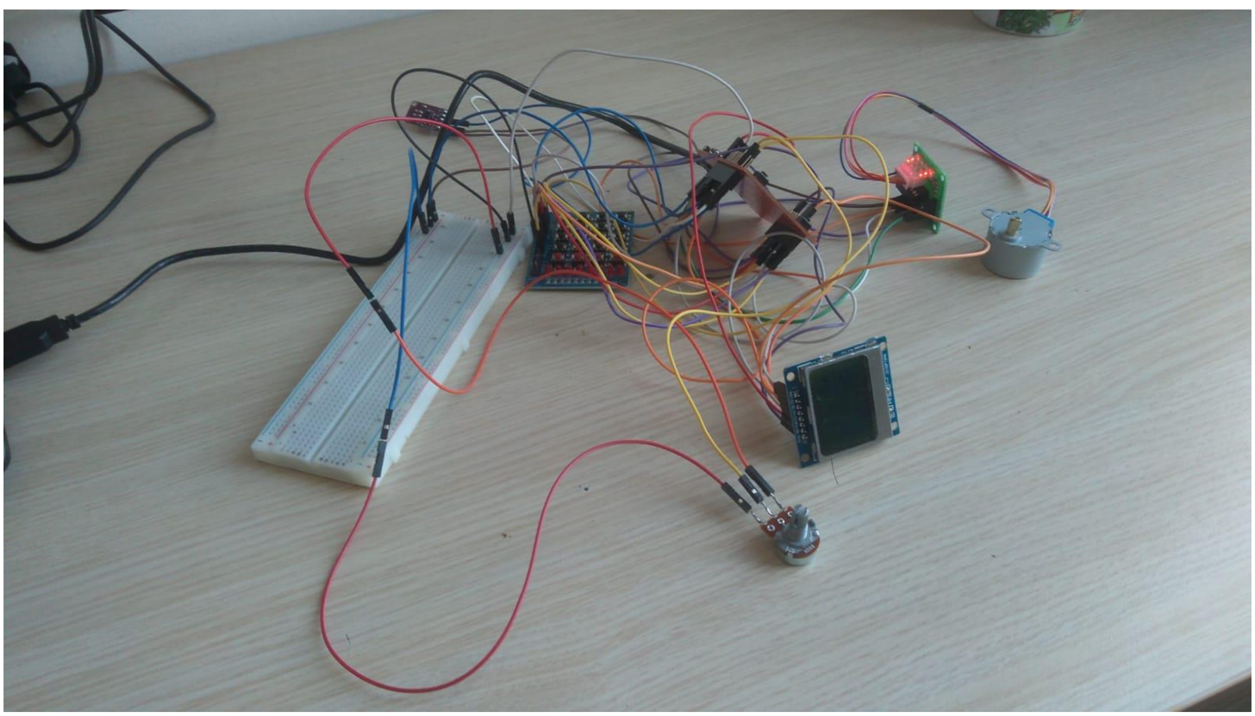

Figure 1 Physical Construction of the Project

In figure 1, the wiring and physical outlook of the project can be observed.

State Machine

In the preliminary report, the state machine in figure 2 is suggested for operation.

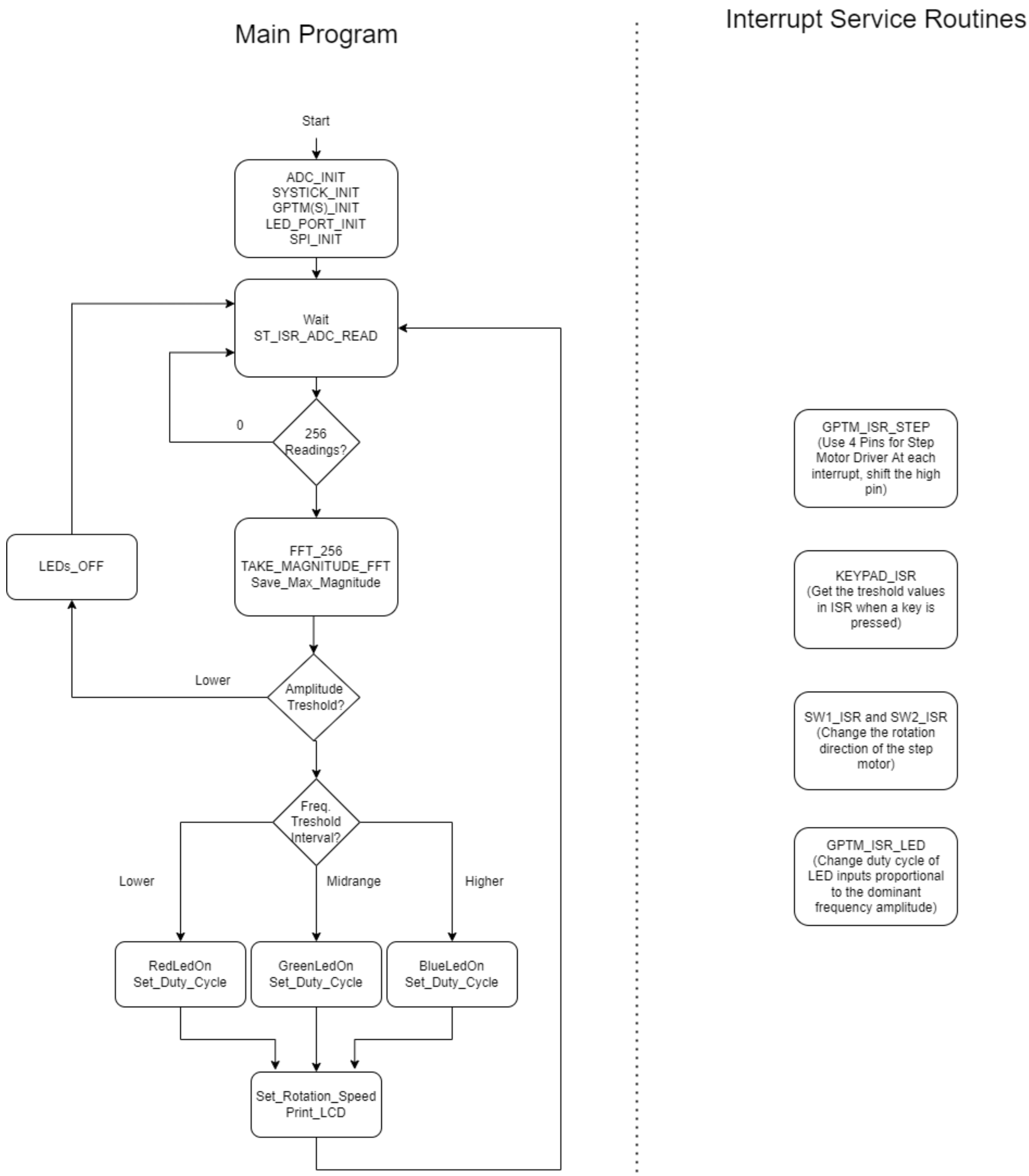

Figure 2 State Machine

Only difference in implementation here is that we use polling for the keyboard. For the case it is implemented with an ISR, we see that rotation of the step motor must halt since it is also driven with an ISR, and nested interrupts are not allowed. With our implementation, step motor does not halt in threshold setting mode, and only LCD screen is not updated.

Critical Submodules

Our main code consists of only few lines. It makes initializations of submodules, and constructs the state machine.

```
__main          PROC
                CPSIE       I
                BL PROJECT_PARAMS_INIT
                BL ADC_INIT
                BL SYSTICK_INIT
                BL TIMERS_INIT
                BL LED_PINS_INIT
                BL STEP_OUT_PINS_INIT
                BL ONBOARDSW_INIT
                BL SPI_INIT
                BL SCREEN_INIT
                BL KEYBOARD_PINS_INIT
                BL MAGNITUDE_ADC_INIT

wait_st_isr     LDR                 R1, =READ_COMPLETE
                LDR                 R0, [R1]
                CMP         R0, #1
                BNE         wait_st_isr
                BL FFT_MAGNITUDE
                ; R0 AND R1 ARE RETURN PARAMETERS OF FFT_MAGNITUDE
                BL ADJUST_STEP_FREQUENCY_LED_PWM
                BL UPDATE_SCREEN
                BL POLL_KEYBOARD
                BL MAGNITUDE_SET_T
                LDR         R2, =READ_COMPLETE
                MOV         R3, #0
                STR         R3, [R2]
                B wait_st_isr
                ENDP
```

We can identify the most critical subsystems as FFT_MAGNITUDE, ADJUST_STEP_FREQUENCY_LED_PWM, UPDATE_SCREEN, MAGNITUDE_SET_T, and ST_ISR. In the operation, microphone is analog read with Systick interrupts configured to trigger with 2kHz frequency. When 256 samples are obtained with this frequency, READ_COMPLETE is set to 1. Then the remaining functions are called in order to update the screen, adjust the brightness of the LED that is on, decide which LED should be on, and determine whether the specific key is pressed in the keyboard. In the reading phase with ST_ISR, data must be shifted by 4 to left not to lose precision. Otherwise, the performance in determining the most dominant frequency drops critically. We also need to account for DC offset on the microphone, otherwise the magnitude values in the FFT becomes upscaled significantly. The microphone sampling code is provided as following:

```
check                   LDR             R0, [R3]
                        ANDS    R0, #0x08
                        BEQ             check
                        ; READ THE DATA
                        LDR             R0, [R4]
                        SUB             R0, #1475 ; SUBTRACT THE 1.25V MIC OFFSET
                        LSL     R0, #4 ; SHIFT THE RESULT BY 16 BITS
                                                  ; FOR THE FFT FUNCTION INPUT
                                                  ; FORMAT (RE=M[31:16] +
;IM=M[15:0])

                        LDR     R1, =MIC_READINGS
                        LDR             R2, =CURRENT_PTR
                        LDR             R3, [R2]
                        CMP             R3, #512 ; OVERFLOW FOR THE MEMORY
                        BEQ             all_read

                        ADD             R1, R3 ; FIND THE CURRENT LOCATION OF THE
;READINGS
                        STR             R0, [R1] ; STORE THE READING
                        ADD             R3, #4 ; INCREASE THE CURRENT PTR BY ONE
                        STR             R3, [R2] ; STORE THE CURRENT PTR

                        LDR             R0, [R6]
                        ORR             R0, #0x08 ; CLEAR THE INTERRUPT USING
;ADC_ISC REGISTER
                        STR             R0, [R6]
                         .

                         .
```

FFT_MAGNITUDE subroutine is called right after the reading is complete. It calculates FFT of the input audio signal, calculates its FFT using ARM CMSIS Signal Processing Library, it calculates the magnitude of each frequency component, and determines the frequency with highest magnitude. Its return values are the dominant frequency, and the magnitude of the dominant frequency. The magnitude at each frequency is calculated as following:

```
FFT_MAGNITUDE_ PROC
                    LDR           R0, =arm_cfft_sR_q15_len256
                    LDR           R1, =MIC_READINGS ; ARRAY OF THE READINGS
                    MOV           R2, #0 ; IFFT FLAG
                    MOV           R3, #1

                    PUSH{R1, LR}
                    BL arm_cfft_q15
                    POP {R1, LR}

                    ; IN THE TABLE, WE HAVE 256 SAMPLES FROM THE FFT OF INPUT
                    ; SIGNAL. EACH 4-BYTE MEMORY LOCATION HOLDS THE REAL PART
                    ; OF THE DFT AT SIGNIFICANT 2-BYTES AND THE IMAGINARY
PART
                    ; IN THE LEAST SIGNIFICANT TWO-BYTES. USING THIS
INFORMATION,
                    ; CALCULATE THE FFT MAGNITUDE AT EACH POINT AND STORE THE
                    ; RESULT TO THE SAME TABLE THAT THE INITIAL READINGS ARE
TAKEN TO
                    MOV           R5, #0
                    MOV           R0, #0 ; USE AS A POINTER ON THE TABLE
                    LDR           R4, =0xFFFF
loop                LDR           R2, [R1, R0] ; WILL HOLD THE IMAGINARY PART
                    MOV           R3, R2 ; WILL HOLD THE IMAGINARY PART
                    LSR           R2, #16 ;
                    SXTAH  R2, R5, R2; SIGN EXTEND THE VALUE
                    MOVS   R2, R2
                    MVNMI  R2, R2
                    AND           R3, R4 ; TAKE THE LEAST SIGNIFICANT HALF
                    SXTAH  R3, R5, R3
                    MOVS   R3, R3
                    MVNMI  R3, R3
                    MUL           R2, R2 ; TAKE THE SQUARE OF THE IMAGINARY
PART
                    LSR           R2, #16 ; LOSE ONE DIGIT PRECISION TO
PREVENT OVERFLOW
                    MUL           R3, R3 ; TAKE THE SQUARE OF THE IMAGINARY
PART
                    LSR           R3, #16 ;
                    ADD           R2, R3 ; MAGNITUDE SQUARE IS IN R2
                    STR           R2, [R1, R0]
                    ADD           R0, #4
                    CMP    R0, #1024
                    BNE           loop
                    .
                    .
```

In this function, we also find the frequency value with maximum magnitude and convert the index to a frequency value using the equation provided in project description.

After determining the maximum magnitude frequency value and the magnitude of it, we call the subroutine ADJUST_STEP_FREQUENCY_LED_PWM. In this subroutine, if the magnitude is lower than the magnitude threshold, no adjustments are made, and LEDs are set to off. Then depending on the determined frequency and magnitude values, LED decision, their brightness, and step motor rotation speed is set by altering the respective timer countdown values.

```
ADJUST_STEP_FREQUENCY_LED_PWM_ PROC
                     ; FIRSTLY, DETERMINE WHETHER THE DOMINANT FREQUENCY IS
GREATER THAN THE
                     ; MAGNITUDE TRESHOLD
                     LDR           R3, =AMPLITUDE_T
                     LDR           R2, [R3]
                     CMP           R1, R2
                     BPL           adjust
                     LDR           R1, =WHICH_LED
                     MOV           R0, #0x03
                     STR           R0, [R1]
                     BXMI LR ; IF MAGNITUDE TRESHOLD IS NOT EXCEEDED DO NOT
MAKE ANY ADJUSTMENTS

                     ; DECIDE WHICH LED SHOULD BE ON WITH THE FREQUENCY
INFORMATION
adjust        LDR           R2, =LOW_FREQ_T
                     LDR           R3, [R2]
                     CMP    R0, R3
                     MOVMI  R4, #0 ; RED LED
                     BMI    save_which_led
                     LDR           R2, =HIGH_FREQ_T
                     LDR           R3, [R2]
                     CMP    R0, R3
                     MOVMI  R4, #1 ; GREEN LED
                     MOVPL  R4, #2 ; BLUE LED
                     .
                     .
```

Then we update the screen with the numbers with parameters we obtained. Aside from the printing the letters, we use the subroutine CONVRT which converts numbers to decimal digits and stores in the memory. The code segment to print the numbers is as following:

```
                    PUSH {R0-R12, LR}
                    LDR             R1, =HIGH_FREQ_T
                    LDR             R4, [R1]
                    LDR             R5, =HIGH_FREQ_DIGITS
                    BL              CONVRT
                    POP     {R0-R12, LR}

                    LDR             R1, =HIGH_FREQ_DIGITS
                    PUSH {R0-R5, LR}
                    BL              SCREEN_FIND_WHICH_NUMBERS
                    POP     {R0-R5, LR}

                    MOV             R3, #6
fnum30      LDRB    R0, [R6], #1
                    PUSH {R0-R12, LR}
                    BL              SCREEN_SEND_CHAR
                    POP     {R0-R12, LR}
                    SUBS    R3, #1
                    BNE     fnum30

                    MOV             R3, #6
fnum31      LDRB    R0, [R7], #1
                    PUSH {R0-R12, LR}
                    BL              SCREEN_SEND_CHAR
                    POP     {R0-R12, LR}
                    SUBS    R3, #1
                    BNE     fnum31

                    MOV             R3, #6
fnum32      LDRB    R0, [R8], #1
                    PUSH {R0-R12, LR}
                    BL              SCREEN_SEND_CHAR
                    POP     {R0-R12, LR}
                    SUBS    R3, #1
                    BNE     fnum32

; NEW LINE
                    MOV             R0, #0x44
                    PUSH {R0-R12, LR}
                    BL              SCREEN_SEND_COMMAND
                    POP     {R0-R12, LR}

                    PUSH {R0-R12, LR}
                    LDR             R5, =40000
                    BL              DELAY
                    POP     {R0-R12, LR}

                    MOV             R0, #0x80
                    PUSH {R0-R12, LR}
                    BL              SCREEN_SEND_COMMAND
                    POP     {R0-R12, LR}

                    PUSH {R0-R12, LR}
                    LDR             R5, =40000
                    BL              DELAY
                    POP     {R0-R12, LR}
                    .
```

The polling for keyboard looks as following:

```
readall                 LDR             R1, =GPIO_PORTC_DATA
                        LDR             R0, [R1]
                        BIC             R0, #0xF0
                        STR             R0, [R1]
                        PUSH {R1, R2, R3, R4, LR}
                        BL              DEBOUNCE_A
                        POP {R1, R2, R3, R4, LR}

                        MVN             R0, R0
                        ANDS    R0, #0xF0; CHECK IF ANY BUTTON PRESSED
                        BXEQ    LR
                        BNE     determiner

determiner
                        LSR     R0, #4
                        MVN     R0, R0
                        ; R2 WILL HOLD THE R VALUE
                        MOV             R2, #0
loopr           ANDS    R3, R0, #0x01
                        BEQ     determinel
                        ADDNE   R2, #1
                        LSR             R0, #1
                        BNE             loopr

determinel      MOV             R3, #0  ; R3 WILL HOLD THE L VALUE
                        LDR             R1, =GPIO_PORTC_DATA
                        LDR             R4, =0x10  ; OUTPUT HIGH TO LINE 1

output          MVN             R4, R4
                        STR             R4, [R1]  ; GIVE THE OUTPUT
                        MVN     R4, R4

                        PUSH {R1, R2, R3, R4, LR}
                        BL              DEBOUNCE_A
                        POP {R1, R2, R3, R4, LR}

                        LSR             R0, #4
                        MVN             R0, R0
                        ANDS    R0, #0x0F; CHECK WHETHER THE BUTTON PRESSED IN THIS
LINE
                        BNE             obtain
                        ADDEQ   R3, #1
                        LSLEQ   R4, #1  ; CHANGE THE LINE
                        BEQ             output


                        ; AT THIS POINT R VALUE IS IN R2 AND L VALUE IS IN R3
```

Overall time until this process is significantly low, around 140ms, which is enough for us to poll whether the specific key in our keyboard is pressed or not. If it is pressed, we go into the threshold selection mode, and expect three more inputs from the user. These three numbers, (0s can be added in front of them) sets the high or low frequency thresholds in order. For example, the first entrance to the threshold setting mode sets the low frequency threshold, second high third, low, etc

Lastly using the other ADC channel of the TI board, we read the analog voltage on a potentiometer. Using the read value, which depends on the angle of its probe, we update the magnitude threshold. Similar to the frequency to timer countdown procedure, we compress the readings between 100-300.

```
sample                  LDR             R0, [R2]
                        ORR             R0, R0, 0x08 ; SET BIT 3 FOR PSSI
                        STR             R0, [R2]
check           LDR             R0, [R3]
                        ANDS    R0, #0x08
                        BEQ             check
                        ; READ THE DATA
                        LDR             R0, [R4]
                        ; SCALE IT BETWEEN 100-300
                        LDR             R1, =200
                        MUL             R0, R1
                        LDR             R1, =4095
                        UDIV    R0, R1
                        LDR             R1, =100
                        ADD             R0, R1

                        LDR             R1, =AMPLITUDE_T
                        STR             R0, [R1]
                         .

                         .
```

With this, we conclude the main critical subsystems. Provided codes are not the full codes, but rather parts of them to illustrate the main idea behind each subsystem. There are also very important subroutines used in updating screen, and ISR to detect whether any of onboard switches are pressed, which changes the rotation direction of the step motor such as ONBOARD_SWTICH_ISR, SCREEN_SEND_COMMAND, SCREEN_SEND_CHAR, etc.

Requirements and Solutions

The first requirements is satisfied since we use three constants for thresholds, which could be altered using the potentiometer and the keypad.

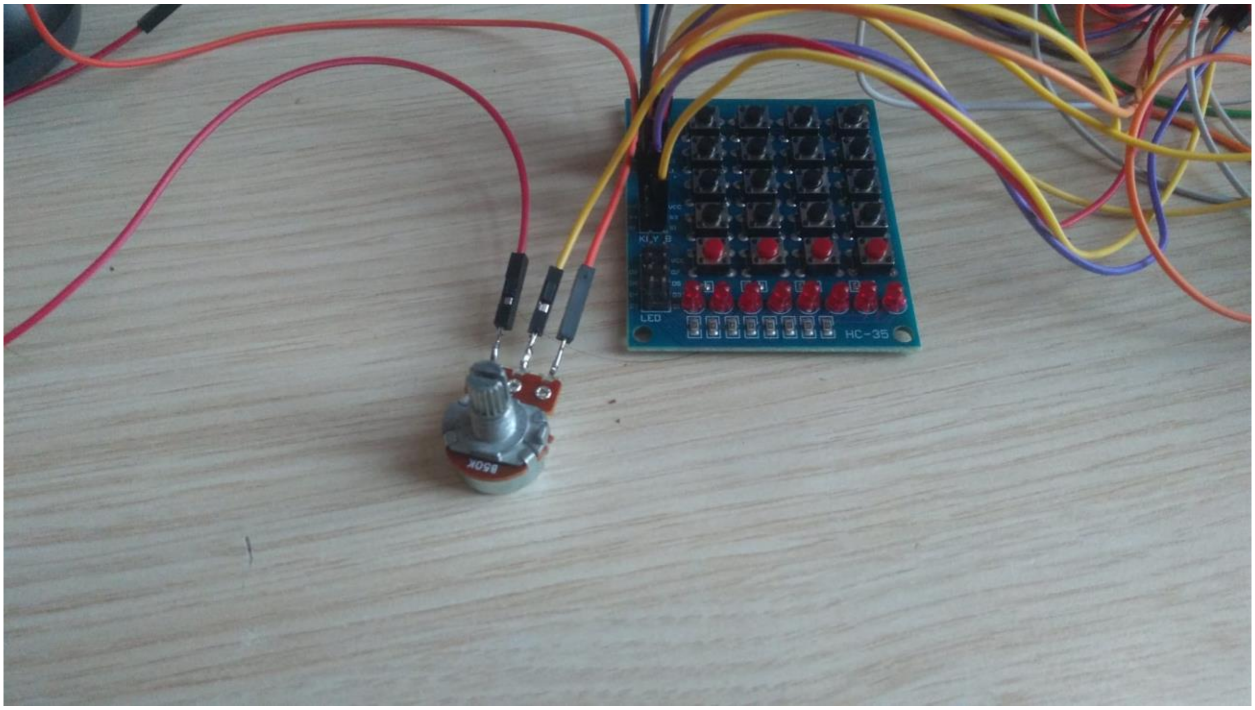

Figure 3 Threshold Setting Equipment

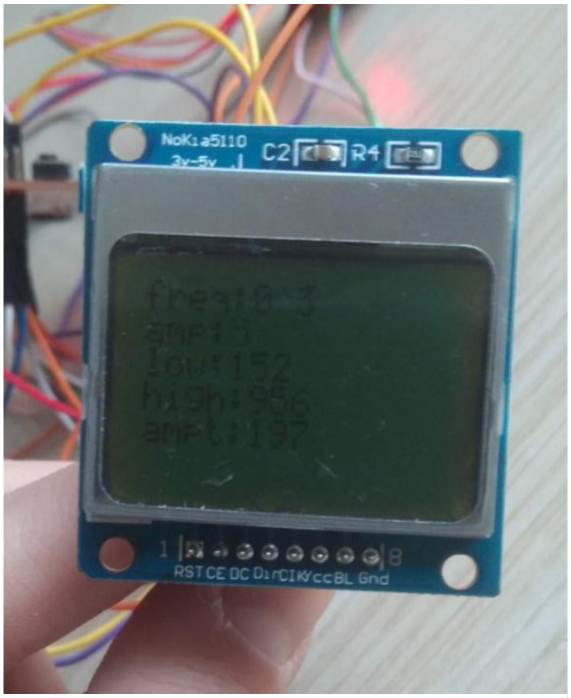

Figure 4 Using the Equipment Low Threshold:152, High Threshold: 956, Amplitude Threshold: 197

Second requirement is satisfied since we do not make any modifications of timer countdown value of the step motor and turn the LEDs off if magnitude threshold is not exceeded.

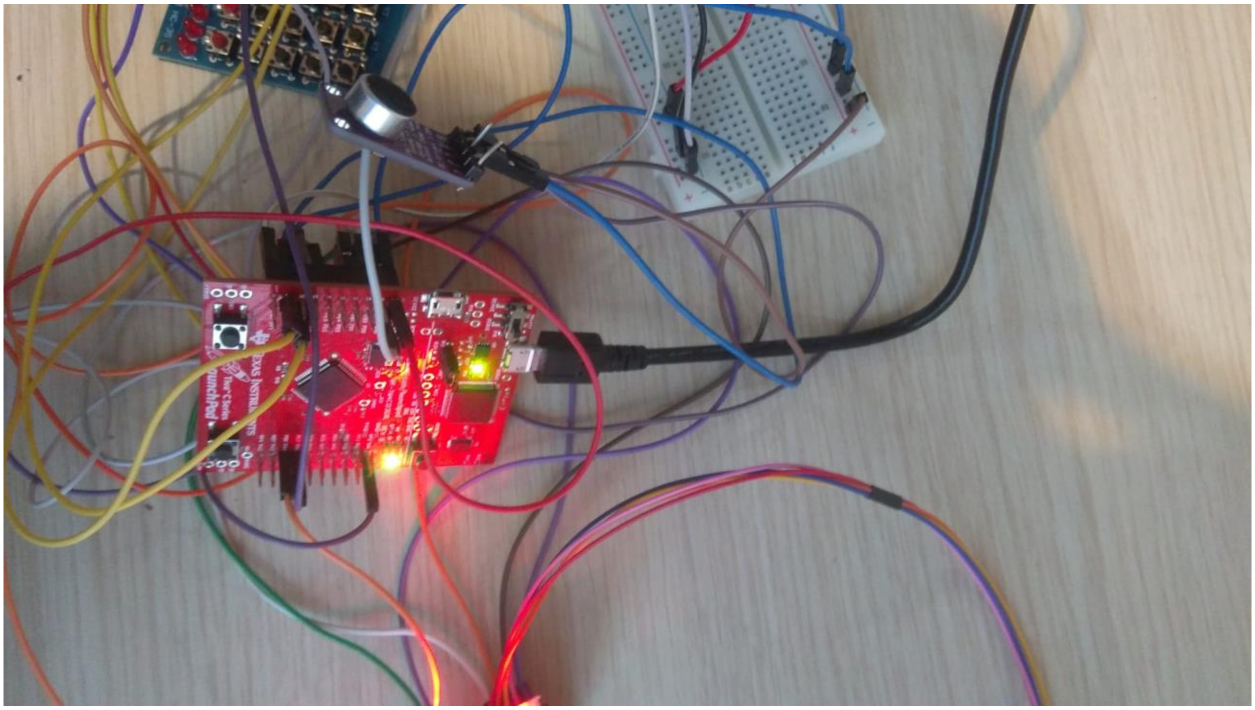

Figure 5 For Midrange Frequencies, Green LED is on

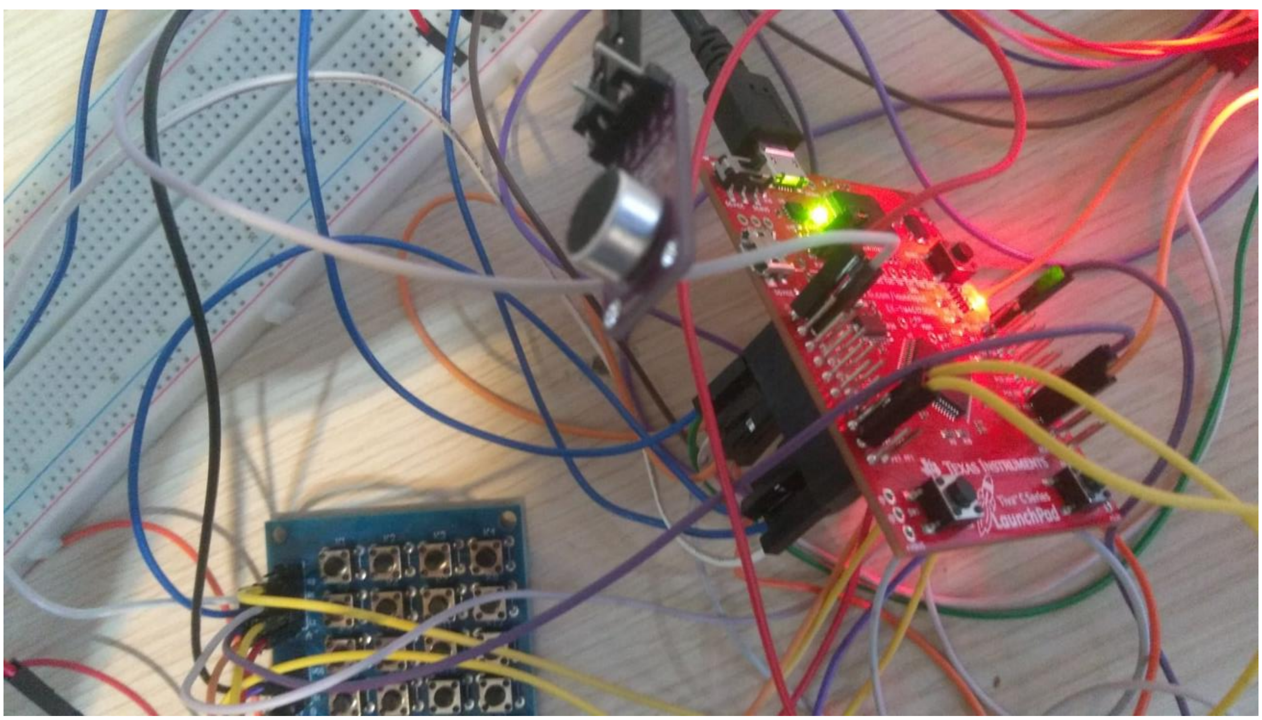

Figure 6 For Lower Frequencies, Red LED is on

The user is able to see the configured thresholds using the LCD screen, which could be seen in figure 4.

The last requirement can also be seen to satisfied in figure 4 since we can update the thresholds using the keyboard and the potentiometer.

Supplementary Subroutines and ISRs

We have a subroutine called PROJECT_PARAMS. It contains all the variables, such as threshold values and initializes them. Its data part can be seen as following:

```
                            AREA        PROJECT_PARAMS, DATA, READWRITE
                            THUMB
WHICH_LED                   DCD           2
DOMINANT_FREQUENCY          FILL          15
DOMINANT_MAGNITUDE          FILL          15
HIGH                        DCD           10000
LOW                         DCD           10000
STEP_DIR                    DCD           0
LOW_FREQ_T                  DCD           0
HIGH_FREQ_T                 DCD           0
LOW_FREQ_DIGITS             DCD           0
HIGH_FREQ_DIGITS            DCD           0
AMPLITUDE_T                 DCD           0
AMPLITUDE_DIGITS            DCD           0
TRESHOLD_SETTING_MODE       DCD           0
SETTING_MODE                DCD           0
KEYBOARD_IN_OUT             DCD           0
                       .
```

These parameters and their correct initialization are critical for the state machine we construct.

We have ADC_INIT subroutine. It initializes the ADC0 for pin PE3, which is used for the microphone. The sample sequence three is used with 125ksps rate. Yet, this rate is meaningless, since we perform the reading in Systick ISR by polling. The Systick countdown value becomes determinant in terms of the maximum sampling frequency we can obtain. The countdown value is chosen to be 1999, which corresponds to 2000 clock cycles. Since the clock Systick uses is 4Mhz, this countdown value sets the sampling rate to 2kHz, as instructed. Its initialization is as following:

```
; SETUP THE TIMER
                              LDR           R1, =NVIC_ST_CTRL ; DISABLE
TIMER DURING THE CONFIGURATION
                              LDR     R0, =0
                              STR           R0, [R1]

                              LDR           R1, =NVIC_ST_RELOAD ; TAKE
THE RELOAD VALUE TO BE 2000-1 FOR
                              LDR           R0, =1999 ; 2 KHZ OF ADC
SAMPLING FREQUENCY
                              STR           R0, [R1]


                              ; SET THE PRIORITY LEVEL
                              LDR           R1, =SHP_SYSPRI3
                              LDR           R0, =0x40000000
                              STR           R0, [R1]

                              LDR           R1, =NVIC_ST_CTRL
                              LDR           R0, =0x03
                              STR           R0, [R1]

                              BX LR
```

We configured timer1 for step motor driving and timer0 for LED power modulation. They are both used in periodic countdown mode, and their countdown value can be altered by either user, or the magnitude of the input dominant frequency.

In step motor driving, we have a critical function FULL_STEP, which creates four signals using the lower order PORTB pins, depending on the current state of rotation. Its code is as following:

```
FULL_STEP_     PROC
               LDR    R1, =GPIO_PORTB_DATA
               LDR    R0, [R1]
               AND        R0, #0x0F
               LDR        R2, =STEP_DIR
               LDR        R3, [R2]
               CMP        R3, #1
               BEQ        counter_clock_wise
               BNE        clock_wise

clock_wise
               CMP        R0, #0x01
               MOVEQ  R0, #0x08
               BEQ        store
               LSR        R0, #1
               B          store
counter_clock_wise
               CMP        R0, #0x08
               MOVEQ  R0, #0x01
               BEQ        store
               LSL        R0, #1
               B          store

store          STR        R0, [R1]
               BX LR
```

Screen initialization is quite critical. We have two main subroutines for the screen initializations. First one, SCREEN_SEND_CHAR, sends a character to the screen by setting D/C port high. Second one, SCREEN_SEND_COMMAND, sends a command signal to the screen by setting D/C port low. A specific sequence for display mode selection, temperature settings, etc. Their source code is omitted, due to them being quite extensive.

We modulate the brightness of the LEDs using the magnitude threshold value. At each cycle of complete read, the HIGH parameter is adjusted, which corresponds the high duration of the duty cycle on the LEDs. A part from LED power countdown ISR is as following:

```
                                        .
make_high_blue
                                        LDR             R0, [R1]
                                        BIC             R0, #0x04 ; CLOSE THE OTHER
LEDS
                                        ORR             R0, #0x04
                                        STR             R0, [R1]
                                        ; CHANGE THE COUNTDOWN VALUE
                                        LDR     R1, =TIMER0_TAILR
                                        LDR     R2, =HIGH
                                        LDR             R0, [R2]
                                        STR     R0, [R1]
                                        B exit
                                        .
                                        .
make_low
                                        LDR             R0, [R1]
                                        BIC     R0, #0x0E
                                        STR             R0, [R1]
                                        ; CHANGE THE COUNTDOWN VALUE
                                        LDR             R1, =TIMER0_TAILR
                                        LDR     R2, =LOW
                                        LDR             R0, [R2]
                                        STR     R0, [R1]
                                       .

                                       .
```

We only provided parts from the source code to illustrate the main idea of the state machine we constructed. The remaining of the code can be found in the project folder.

Conclusion

Low level programming, such as using the assembly language, gives us quite an extensive freedom, and the opportunity of creating very efficient algorithms. In this project, using assembly language, we created algorithms for calculating the most dominant frequency in an audio sample and using that information, we drove a step motor. Many peripherals, such as ADC, timers, GPIO, are used to both driving the step motor, taking input from the microphone, driving the screen, and other user interface modules. We initialized these peripherals, read or write data to them, or created interrupts when necessary. With this project, we practiced using almost all peripherals of the TI board, and created a fully functional, user friendly product.