ISLAMIC UNIVERSITY OF TECHNOLOGY

# PROJECT REPORT

## PULSE WIDTH AND FREQUENCY MEASUREMENT

## GROUP-2

**Course No : 4706**

**Course Name : (Microcontroller Based System Design Lab)**

**Submitted by**

**STUDENT ID : 190021312 ( Mainul Islam )**

**190021310 ( Md. Ahsanul Adib )**

**190021318 ( Abrar Fahim )**

**190021336 ( Sam An Saif )**

**190021344 ( Safa Omar Moussa )**

**Department : EEE**

# Contents

## Objective:

The objective of the project is to get us familiarized in using 8051 family microcontrollers as well as interfacing with some other electronic components with it.

Our goal is to make the 8052 microcontroller able to measure frequency of a signals coming from outside as well as generating a frequency.

Measure frequency also includes measuring the duty cycle of the frequency which can be useful in measuring pwm signals.

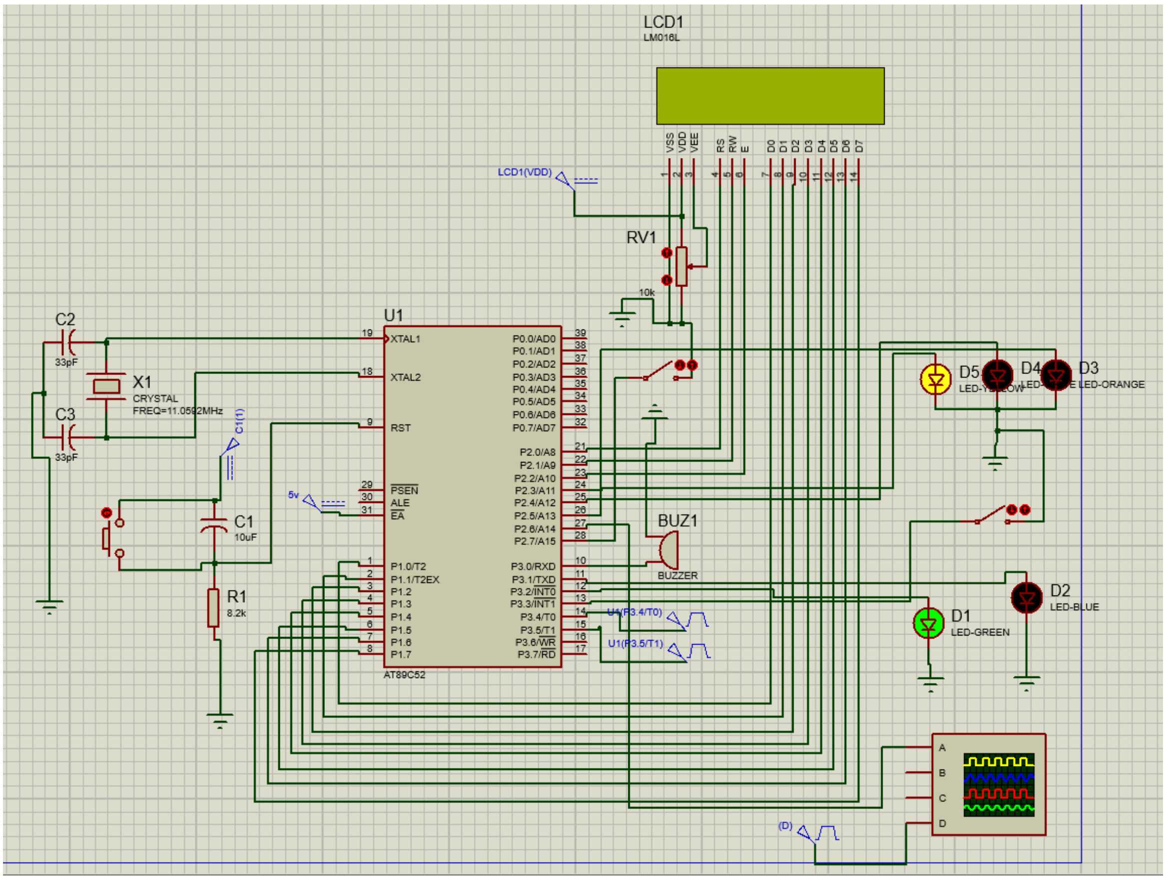
Generating frequency is a generalized approach that means it can make any frequency provided instantaneously, not provided earlier.

Other priorities include making the microcontroller halt in 2 ways. Either measuring frequency will halt, or both measuring frequency as well as generating frequency will halt too.

## Required Components:

| No. | Components | Quantity | Price (TK) |
|---|---|---|---|
| 1 | 8052 Microcontroller Board | 1 | 8000 |
| 2 | Jumper Wires | 2 (sets) | 40 |
| 3 | Two SPST (Single Pole Single Throw) Switch | 2 | 3 |
| 4 | Arduino Mega | 1 | 0 |
| 5 | Frequency Analyzer | 1 | 0 |

## Circuit Diagram:




Here, 1st external signal comes to Port P3.4. 2nd external signal comes to Port P3.5. Generated signal is passed out through Port P2.6. A buzzer is connected to Port P3.0.

The port that is used to halt measuring frequencies is P3.3. Whereas the port that is used to halt everything is port P2.7.

In simulation LED D0-D7 connected to port P1 but in hardware simulation, its Port P0. P2.0,P2.1,P2.2 are used for Controlling LED's by connecting them to RS, RW and Enable pin of LED.

LED's are connected in P2.3, P2.4, P2.5 and P3.1, P3.2. The first 3 LEDs indicates whether one frequency is greater , equal or smaller than the other one. The later two LEDs indicate the task 3 of mandatory features.

A oscilloscope was used to check whether the generated signal matches with the expected signal.

The remaining connections are used to make the microcontroller running properly, like connecting oscillators with proper capacitor, power etc.

## Features:

Mandatory Features:

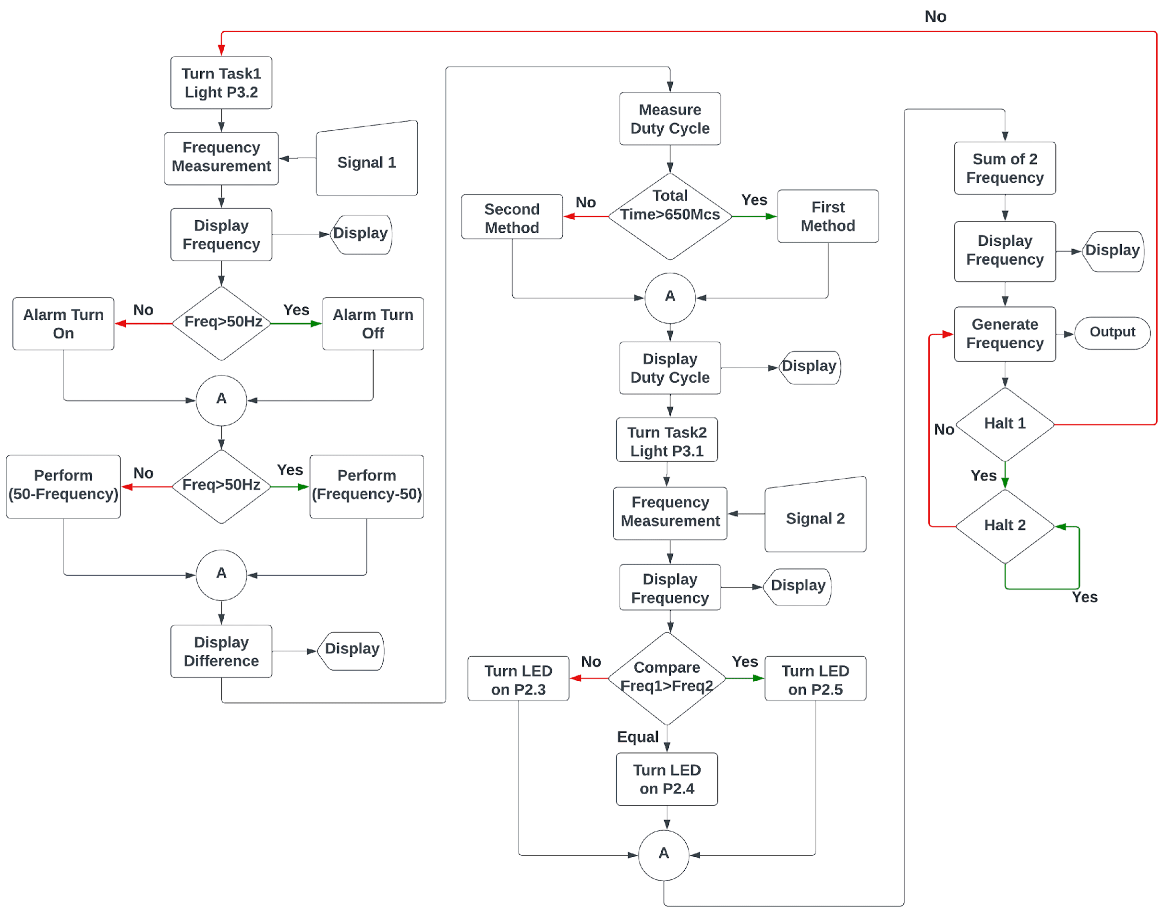
1. Measure the frequency and duty cycle of a signal and display it on the LCD. If it is less/more than 50Hz sound an alarm and specify the difference on the LCD.
2. Take two signals, measure their frequencies and generate a new signal equal to the sum of those frequencies.
3. Continuously carry out the above tasks and LED 1 should be turned on when the first task is being carried out, and LED 2 should be turned on when the second task is being carried on.

Optional Features:

1. The Microcontroller will run in 2 modes. One of them is trigger mode. This feature can halt or pause the microcontroller from executing when needed using external pins. This is done using 2 pins with spdt switches. One switch halt the measuring of frequencies only but will be generating frequency. The other switch will halt all process of the microcontroller by infinitely looping.
2. 3 Led's will blink depending upon the frequency of the second signal being larger, equal or less than the frequency of the first signal.

# Working Principle:

Algorithm Flow Chart:




## **Mandatory tasks:**

Preparatory Stage:

Before starting task-1, we have to allocate the interrupt location of timer -2. We will complement the port in that location and will clear timer overflow flag.

We will label all the necessary memories. For the first time we will turn all the LED’s off, buzzer off and will make some ports as input to take the signals as well as detecting halt operation.

Task-1:

Task-1 light is turned on. Out of the three timers, one timer was used to measure 1 sec and other timer was used as a counter to count the number of cycles within that 1 sec coming from a external signal. The number of cycle is displayed.

In the next part, it was checked the frequency was 50Hz or not. First, the high byte of frequency was checked. If 0 the lower byte was checked that if it is 50 or not. If it is less than 50Hz then it will buzz the alarm or otherwise it will not buzz.

In the next part, the difference between the frequency and 50Hz. To figure that out, we check weather the high byte is 0 or not. If not 0 then the frequency will be higher than 50 so we will subtract 50 from the frequency. If the higher byte is 0 then we will check the lower byte. if the lower byte is less than 50, we will subtract frequency from 50Hz. Then we will display the difference.

In the following part, we will figure out the time period in **OFF** cycle. First, we will lose one cycle to sync with the upcoming cycle until the signal goes high, a timer will measure the time. Same goes for the time period calculation of **ON** cycle. Only the difference is the timer will measure until the signal goes low.

In the upcoming section, the TON and TOFF was summed (24 bit addition) to get the total time for each cycle to take place. Then we checked weather total time is less than 650 machine cycle or not. If it is less than 650 machine cycles then we will first multiply 100 with TON then we will divide the product with total time. The resultant is the duty cycle.

If it's more than 650 machine cycle, first we will divide 100 from total time and then we will divide TON with the previous quotient found. The resultant is the duty cycle. Then we displayed the duty cycle in LCD.

Here task- 1 ends. Task-2 starts and Task-2 LED turned on and Task-1 LED turned off.

<u>Task-2:</u>

Another signal was measured in a similar way like the previous one except the timer 1 was acting as counter and timer 0 was acting as timer. Frequency was displayed in LCD.

In the next part, the two frequencies was compared. First the higher byte of the two frequency was compared then the lower byte. Depending upon the relation between the frequencies 3 different LED's are turned on individually indicating whether frequency 1 greater or equal or smaller than frequency 2.

In the following part, both the frequencies were summed. Using a 24 bit division subroutine the required value was found out to be subtracted from the corresponding higher byte and lower byte of timer 2 that was used to generate the frequency.

In the last part, two halt operation were conducted. One halt operation only pauses the measurement of frequency but frequency generation was continuing. Using both halt operation both the frequency generation and frequency measurement can be paused that is nothing but infinite looping until the conditions become reverse.

Task-3:

Finally, the code loops back to the vary beginning indicating a continuous operation. We assigned P3.1 for task 1 LED and P3.2 for task 2 LED. So, for our task 1 we need the LED for task 1. At first we had to clear the task 2 LED. After clearing the task 2 LED, we just used SETB P3.1 and turned on the LED for task 1. That's how we carried out task 3 in the entire code.

### Duty Cycle (24 bit addition):

Storing at R3 after addition with carry of R5 and R7. If a carry comes up that will be added with R4 and R6 then the value will be stored at R2. Here, if another carry comes up that will be added with Car_TON and Car_TOFF and the value will be stored at R1. After that it will return to duty cycle. Here, the value of R1, R2 and R3 will be stored at a different location named TOT0, TOT1 and TOT2 because R1,R2 and R3 can be free to store any other variable. After this we are adding TON1 withTOFF1 because they are high byte and TON2 with TOFF2 because they are low byte. That's how we are adding 24 bit addition here.

### Print (8 bit):

Taking FD1, FD2 and FD3 variable as empty variable. Whatever value are in A that will be divided by 10 which will be stored in B. After division the remainder will be stored in FD3. Again 10 will be stored in B and quotient will be stored in A. Then remainder will be stored at FD2. Then the quotient will be stored in FD1. Direct value can not be placed for displaying digits. So the conversion will be done using OR operation. Moving every value at A then OR 30 with variable FD1, FD2, FD3. This is how 8 bit operation is done.

### Print (16 bit):

The subroutine takes five memory locations (FD1 to FD5) to store the digits of the decimal number. It then performs a series of divisions by 10 to extract each digit of the 16-bit decimal number for a total of 5 times. Each time we reset the values in R4 to avoid error.

The code converts each digit to ASCII representation and calls a DISPLAY subroutine to show it on the output device. After each digit is displayed the next significant digit is goes through the process and gets displayed. The subroutine returns after displaying all five digits.

### Division (24 bit):

This subroutine takes 3 byte dividend input as well as 3 byte divisor as input as the resultant is 2 byte Quotient and the Last byte of reminder. The dividend is subtracted by the divisor in a conditional loop. In each iteration a register is incremented. When the register is overflowed, another register gets incremented and the previous register is cleared. These 2 registers are nothing but the resultant quotient high byte and low byte. The loop breaks when the dividend high byte goes to FF from 0 also providing a carry , indicating dividend is less than zero. The reminder is the value from 0 to 9 found in the last subtraction of divisor value from dividend. This reminder value is used for printing digit. And the resultant quotient is used to measure duty cycle as well as printing and other supporting tasks.

## Code:

```
;GROUP-2
;190021312, 310, 318, 336 ,344

    ORG 00H
    LJMP MAINN

    ORG 002BH               ; Interrupt Location
    CPL P2.6
```

```
    CLR T2CON.7
    RETI

    ORG 0030H
    MAINN:
    MOV SP, #70H
    MOV PSW, #00H

    RS EQU P2.0
    RW EQU P2.1
    ENBL EQU P2.2

    T2CON EQU 0C8H          ; Had to name memory locations of  timer2
components as missing in MIDE
    T2MOD EQU 0C9H
    TR2 EQU T2CON.2
    RCAP2L EQU 0CAH
    RCAP2H EQU 0CBH

    DTC EQU 67H
    FD1 EQU 51H
    FD2 EQU 52H
    FD3 EQU 53H
    FD4 EQU 54H
    FD5 EQU 55H

    CAR_TOFF EQU 61H
    TOFF1 EQU 58H
    TOFF2 EQU 59H
    CAR_TON EQU 62H
    TONE1 EQU 5AH
    TONE2 EQU 5BH

    TOT0 EQU 66H
    TOT1 EQU 65H
    TOT2 EQU 64H

    REM  EQU 63H

    HIH EQU 6EH
    LWO EQU 6FH

    CLR P3.0
    CLR P3.1
    CLR P3.2

    SETB P3.3
    SETB P2.7
    CLR P2.3
    CLR P2.4
    CLR P2.5
    MOV A,#0d
```

```
    MOV CAR_TOFF,A
    MOV CAR_TON,A

    LCALL LCD_SET_UP
    LCALL   DELAY

    MOV DPTR, #MSG_1
    LCALL STR_DISP

    MOV DPTR, #MSG_2
    MOV A, #0C0H
    LCALL COMMAND
    LCALL DELAY
    LCALL STR_DISP

    SETB P3.4           ;set the port to input mode (F1) COUNTER 0 as EXTERNAL
INPUT
    SETB P3.5           ;set the port to input mode (F2) COUNTER 1 as EXTERNAL
INPUT
;-----------------------------------------------------------------------------
----------------------------------------TASK 1
;-------------------------------------------------------------------------FRE
QUENCY MEASUREMENT of 1st Input Signal
AT_START:
    CLR P3.2            ; task 2 light off
    SETB P3.1           ; task 1 light on

    MOV TMOD, #15H          ; Timer 1 as timer and Timer 0 as counter
    MOV TCON, #00H

    MOV TL0,#00H
    MOV TH0,#00H
    F1_low1: JNB P3.4,F1_low1   ; Dropping two cycle to sync with the 1st
signal input
    F1_high1: JB P3.4,F1_high1
    SETB TR0

    MOV R0,#15          ; Make 1 SECOND
AGN:    MOV TL1, #0FBH
    MOV TH1, #0FH
    SETB TR1
BACK:   JNB TF1, BACK
    CLR TF1
    CLR TR1
    DJNZ R0, AGN
    CLR TR0
;--------------------------------------------------------------------------DI
SPLAY FREQUENCY 1
    LCALL LCD_SET_UP
    MOV DPTR, #MSG_3
    LCALL STR_DISP
```

```
    MOV A, #0C0H
    LCALL COMMAND
    LCALL DELAY

    CLR A
    MOV R1,A
    MOV R0,A
    MOV R2,A
    MOV R3,A
    MOV R5,A

    MOV R3, TH0
    MOV R2, TL0

    PUSH 03             ; keep TH0 of F1 in stack pointer
    PUSH 02             ; keep TL0 of F1 in stack pointer

    LCALL PRINT_DIGIT_16

    MOV DPTR, #HZ
    LCALL STR_DISP

;-----------------------------------------------------------------------Buzz
er Alarm
    MOV A,TH0
    JNZ NALRM
    MOV A,TL0
    MOV B,#50d
    CJNE A,B,ALRM           ; if TL0 < 50hz
ALRM:   JNC NALRM
    SETB P3.0
    LJMP OKK
NALRM:  CLR P3.0
OKK:
;------------------------------------------------------------------------Dis
play difference from 50Hz

LCALL LCD_SET_UP
MOV DPTR, #MSG_4
LCALL STR_DISP

MOV A, #0C0H
LCALL COMMAND
LCALL DELAY

CLR A
CLR C
MOV A , TL0
SUBB A, #50d
MOV R2, A
MOV A, TH0
SUBB A, #0
```

```
MOV R3, A

JNC NEEE                         ; if carry not found F1>50hz , go to NEEE

NEE:
MOV A, #50d
SUBB A, TL0
MOV R2, A
MOV R3, #0

NEEE:
MOV R5,#0d
LCALL PRINT_DIGIT_16        ; show difference
MOV DPTR, #HZ
LCALL STR_DISP
;-----------------------------------------------------------------------------
--TOFF CALCULATION
MOV TL1, #00H
MOV TH1, #00H
TOFF_low1: JNB P3.4,TOFF_low1       ; Dropping two cycle to sync with the
signal input
TOFF_high1: JB P3.4,TOFF_high1
SETB TR1                ; start timer when pulse is low.
TOFFL_1: JNB P3.4 , TOFFL_2 ;
    SJMP TOFFL_3
TOFFL_2: JNB TF1  , TOFFL_1 ;
    INC CAR_TOFF            ; increment MSB of TOFF
    CLR TF1
    SJMP TOFFL_1
TOFFL_3: CLR TR1
    MOV TOFF1, TH1
    MOV TOFF2, TL1
;-----------------------------------------------------------------------------
-TON CALCULATION
MOV TL1, #00H
MOV TH1, #00H
TON_low1:  JB P3.4 ,TON_low1        ; Dropping two cycle to sync with the
signal input
TON_high1: JNB P3.4,TON_high1
SETB TR1                ; start timer when pulse is high.
TONL_1: JB P3.4 , TONL_2 ;
    SJMP TONL_3
TONL_2: JNB TF1  , TONL_1 ;
    INC CAR_TON         ; increment MSB of TONE
    CLR TF1
    SJMP TONL_1
TONL_3: CLR TR1
    MOV TONE1, TH1
    MOV TONE2, TL1
;-----------------------------------------------------------------------------
--FIND DUTY CYCLE
```

```
;--------------------------------------------------------------------MAKE
Ttotal
;CAR_TON
MOV R6,TONE1
MOV R7,TONE2
;CAR_TOFF
MOV R4,TOFF1
MOV R5,TOFF2

LCALL ADD_24bit             ; output stored in TOT0(highest) TOT1(high byte)
TOT2
MOV TOT0, R1
MOV TOT1, R2
MOV TOT2, R3

;-----------------------------------------------------------------------------
---Two Ttotal secnario
CLR C
MOV A, TOT2
SUBB A,#10001010b
MOV A, TOT1
SUBB A,#00000010b
MOV A, TOT0
SUBB A, #0d              ; whether Total < 650 or note
JC SEC_CASEE
;-----------------------------------------------------------------------------
---First case where Ttotal>100
FIR_CASEE:
                    ;1st part
    MOV R2, TOT2
    MOV R3, TOT1
    MOV R4, TOT0
    MOV R0, #100d
    MOV R1, #0d
    MOV A,#0d               ;clear
    MOV R6,A
    MOV R7,A
    MOV R5,A
    LCALL DIV_24bit
    MOV TOT2, R6
    MOV TOT1, R7
                    ;2nd part
    MOV R2, TONE2
    MOV R3, TONE1
    MOV R4, CAR_TON;
    MOV R0, TOT2
    MOV R1, TOT1
    MOV A,#0d               ;clear
    MOV R6,A
    MOV R7,A
    MOV R5,A;
    LCALL DIV_24bit
```

```
    LJMP EXITTT
;-----------------------------------------------------------------------------
----Second case where Ttotal<=100
SEC_CASEE:
    CLR C            ;1st part ,output will be Ton*100
; First number in R6(H) and R7
; Second number in R4(H) and R5.
; Result oin R0(MSB), R1, R2 and R3(LSB).
    MOV R6, TONE1
    MOV R7, TONE2

    MOV R4,#0d
    MOV R5,#100d

    LCALL MUL16_bit

    MOV TONE2 , R3
    MOV TONE1 , R2
                    ;2nd part
    MOV R2, TONE2
    MOV R3, TONE1
    MOV R4, #0d             ;msb as TON can't be more than 16 bit
    MOV R0, TOT2
    MOV R1, TOT1
    MOV A,#0d           ;clear
    MOV R6,A
    MOV R7,A
    MOV R5,A
    LCALL DIV_24bit

EXITTT:
    MOV LWO, R6
    MOV HIH, R7
;-----------------------------------------------------------------------------
-------Display duty cycle
MOV DTC, LWO

MOV A, LWO;-------redundency
SUBB A, #01100100b
JC RED

MOV DTC, #01011111b
CLR C
RED:
    LCALL LCD_SET_UP
    MOV DPTR, #MSG_7
    LCALL STR_DISP

    MOV A, #0C0H
    LCALL COMMAND
    LCALL DELAY
```

```
    CLR A
    MOV A,DTC            ; TL0 CONTAINS Duty cycle COUNT
    LCALL PRINT_DIGIT

    CLR P3.1
    SETB P3.2
;-----------------------------------------------------------------------------
-------------------------------TASK2
;-------------------------------------------------------------------FREQUENCY
MEASUREMENT of 2nt Input Signal
MOV TMOD, #51H               ; Timer 1 as counter and Timer 0 as timer
MOV TCON, #00H

MOV TL1,#00H
MOV TH1,#00H
F2_low1: JNB P3.5,F2_low1
F2_high1: JB P3.5,F2_high1
                    ; done to sync with the 2nd Input signal from start
SETB TR1

MOV R0,#15               ; MAKE 1 SECOND
AGN1:   MOV TL0, #0FBH
    MOV TH0, #0FH
    SETB TR0
BACK1:   JNB TF0, BACK1
    CLR TF0
    CLR TR0
    DJNZ R0, AGN1
    CLR TR1
;--------------------------------------------------------------------DISPLAY
FREQUENCY 2
    LCALL LCD_SET_UP
    MOV DPTR, #MSG_8
    LCALL STR_DISP

    MOV A, #0C0H
    LCALL COMMAND
    LCALL DELAY

    CLR A                ;Clear
    MOV R1,A
    MOV R0,A
    MOV R2,A
    MOV R3,A
    MOV R5,A
    MOV R3, TH1
    MOV R2, TL1

    PUSH 03                  ; Push TH1
    PUSH 02              ; Push TL1
    LCALL PRINT_DIGIT_16
```

```
    MOV DPTR, #HZ
    LCALL STR_DISP

    CLR P2.3 ;clear LED's
    CLR P2.4
    CLR P2.5
;-----------------------------------------------------------------------------
---Compare 2 input frequencies
    CLR C
    POP 00              ; pop TL1
    POP 01              ; pop TH1
    POP 02              ; Pop TL0
    POP 03              ; pop TH0
    MOV A, R1           ; freq 2
    MOV B, R3           ; freq 1
    CJNE A,B , STG1
    MOV A,R0            ; since TH0 = TH1
    MOV B,R2
    CJNE A,B,  STG2
    SETB P2.4           ; F2 = F1
    LJMP STG0
STG1:   JC p23
    SETB P2.5           ; freq1 < freq2
    LJMP STG0
p23:    SETB P2.3           ; freq1 > freq2
    LJMP STG0
STG2:   JC p23
    SETB P2.5           ; freq1 < freq2
    LJMP STG0
STG0:
;-----------------------------------------------------------------------Sum of
2 input signals' frequencies
    MOV A, R0
    ADD A, R2
    MOV R0, A           ; R0=R0+R2 (TL0 of 2 freqs)
    MOV A, R1
    ADDC A, R3
    MOV R1,A            ; R1=R1+R3 (TH0 of 2 freqs)

    PUSH 00
    PUSH 01
    LCALL LCD_SET_UP
    MOV DPTR, #MSG_9
    LCALL STR_DISP

    MOV A, #0C0H
    LCALL COMMAND
    LCALL DELAY

    POP 03 ;H
    POP 02
```

```
    PUSH 02
    PUSH 03
    LCALL PRINT_DIGIT_16

    MOV DPTR, #HZ
    LCALL STR_DISP
    POP 01
    POP 00
;----------------------------------------------------------------------Find the time period of the summed frequency
    MOV R2, #00111010B      ;LSB dividend
    MOV R3, #00010000B
    MOV R4, #00001110B      ;MSB dividend 921658
    MOV R5, #0B
    MOV R6, #0B             ;LSB loop counter== quotient
    MOV R7, #0B             ;MSB loop counter==quotient
    LCALL DIV_24bit         ;Final output to be stored in LWO (R6 also) and HIH (R7 also)
;---------------------------------------------------------------------Make the output half (for each half cycle)
    CLR C
    MOV R2, LWO
    MOV R3, HIH
    MOV R4, #0d
    MOV R5, #0d
    MOV R6, #0d
    MOV R7, #0d
    MOV R0, #2d
    MOV R1, #0d
    LCALL DIV_24bit
    ;INC R6
    ;MOV LWO, R6
    ;MOV HIH, R7



;----------------------------------------------------------------------Start Timer2
    CLR C
    CLR TR0
    MOV T2CON , #00000000B
    MOV T2MOD , #00000000B
    MOV A, #0FFH
    SUBB A, LWO
    MOV LWO,A
    MOV RCAP2L, LWO         ;Timer initial value = FF- quotient
    CLR C

    MOV A,#0FFH
    SUBB A, HIH
    MOV HIH, A
    MOV RCAP2H, HIH
```

```
    CLR C

    MOV IE, #10100000B
    SETB TR2
;---------------------------------------------------------------------------HA
LT
HLT:    JNB P3.3 ,HLTT ; halt input if grounded
LJMP LAST
HLTT:   JB P2.7 ,HLT  ; halt generation if grounded
    CLR TR2
LJMP HLT
;---------------------------------------------------------------------------RE
SET
LAST:   MOV TL1,#00H
    MOV TH1,#00H
    MOV TL0,#00H
    MOV TH0,#00H
    CLR TR1
    CLR TR0

    LJMP AT_START           ; RETURN TO THE Beginning
;---------------------------------------------------------------------------SU
BROUTINES
;---------------------------------------------------------------------------PR
INT_DIGIT_16 Sub-Routine
PRINT_DIGIT_16:
    ;Clearing memory space
    MOV FD1, #0H    ;MSD
    MOV FD2, #0H
    MOV FD3, #0H
    MOV FD4, #0H
    MOV FD5, #0H    ;LSD
    ; Taking out each digit
    ;1
    ;DIVISOR = 10
    MOV R1, #00H
    MOV R0, #0AH
    MOV R4,#0d
    LCALL DIV_24BIT
    MOV FD5, 63H    ; LOW BYTE OF REMAINDER -- LSD 63H
    ;2
    MOV R3,07H  ; QOUTIENT HIGH TO DIVIDEND
    MOV R2,06H  ; QOUTIENT LOW  TO DIVIDEND
    ;DIVISOR = 10
    MOV R1, #00H
    MOV R0, #0AH
    MOV R4,#0d
    LCALL DIV_24BIT
    MOV FD4, 63H    ; LOW BYTE OF REMAINDER
    ;3
    MOV R3,07H  ; QOUTIENT HIGH TO DIVIDEND
    MOV R2,06H  ; QOUTIENT LOW  TO DIVIDEND
```

```
    ;DIVISOR = 10
    MOV R1, #00H
    MOV R0, #0AH
    MOV R4,#0d
    LCALL DIV_24BIT
    MOV FD3, 63H    ; LOW BYTE OF REMAINDER
    ;4
    MOV R3,07H  ; QOUTIENT HIGH TO DIVIDEND
    MOV R2,06H  ; QOUTIENT LOW  TO DIVIDEND
    ;DIVISOR = 10
    MOV R1, #00H
    MOV R0, #0AH
    MOV R4,#0d
    MOV R5,#0d
    LCALL DIV_24BIT
    MOV FD2, 63H    ; LOW BYTE OF REMAINDER
    ;5
    MOV FD1, 06H    ; LOW BYTE OF QUOTIENT -- MSD
    ; printing
    ;ASCII
    CLR A
    MOV A, FD1
    ORL A, #30H
    LCALL DISPLAY
    ;ASCII
    CLR A
    MOV A, FD2
    ORL A, #30H
    LCALL DISPLAY
    ;ASCII
    CLR A
    MOV A, FD3
    ORL A, #30H
    LCALL DISPLAY
    ;ASCII
    CLR A
    MOV A, FD4
    ORL A, #30H
    LCALL DISPLAY
    ;ASCII
    CLR A
    MOV A, FD5
    ORL A, #30H
    LCALL DISPLAY
    RET
;-----------------------------------------------------------------------PRIN
T_DIGIT Sub-Routine
PRINT_DIGIT:
    MOV FD1, #0H    ;MSD
    MOV FD2, #0H
    MOV FD3, #0H    ;LSD
    MOV FD4, #0H
```

```
    MOV FD5, #0H

    MOV B,#10
    DIV AB
    MOV FD3, B  ; LSD
    MOV B,#10
    DIV AB
    MOV FD2, B
    MOV FD1, A  ; MSD

    MOV A, FD1
    ORL A, #30H
    LCALL DISPLAY

    MOV A, FD2
    ORL A, #30H
    LCALL DISPLAY

    MOV A, FD3
    ORL A, #30H
    LCALL DISPLAY
    RET
;-------------------------------------------------------------------------STR_
DISP Sub-Routine
STR_DISP:
    CLR A
    MOVC A,@A+DPTR
    JZ FINISH_1
    LCALL DISPLAY
    LCALL DELAY
    INC DPTR
    LJMP STR_DISP
FINISH_1: RET
;---------------------------------------------------------------------------LC
D SETUP Sub-Routine
LCD_SET_UP:

    ; DISPLAY COMMANDS
    MOV     A, #38H
    LCALL   COMMAND
    LCALL   DELAY
    MOV     A, #0EH ;display on, cursor on
    LCALL   COMMAND
    LCALL   DELAY
    MOV     A, #01  ;clear LCD
    LCALL   COMMAND
    LCALL   DELAY
    MOV     A, #06H ;shift cursor right
    LCALL   COMMAND
    LCALL   DELAY
    MOV     A, #80H ;cursor at line 1 postion 1
    LCALL   COMMAND
```

```
    LCALL   DELAY
    RET

COMMAND:LCALL READY
    MOV P1, A
    CLR RS
    CLR RW
    SETB ENBL
    LCALL DELAY
    CLR ENBL
    RET

DISPLAY: LCALL READY
    MOV P1, A
    SETB RS
    CLR RW
    SETB ENBL
    LCALL DELAY
    CLR ENBL
    RET

READY:  SETB P1.7
    CLR RS
    SETB RW
WAIT:   CLR ENBL
    LCALL DELAY
    SETB ENBL
    JB P1.7, WAIT
    RET

;-------------------------------------------------------------------------DELA
Y Sub-Routine (Large)
DELAY:  MOV R3, #50
AGAIN_2:MOV R4, #255
AGAIN:  DJNZ R4, AGAIN
    DJNZ R3, AGAIN_2
    RET
;-------------------------------------------------------------------------DELA
Y2 Sub-Routine (Small)
DELAY_2:MOV R5, #100
AGAIN_3:LCALL DELAY
    DJNZ R5, AGAIN_3
    RET
;-------------------------------------------------------------------------ADD_
24bit Sub-Routine
ADD_24bit:
    MOV A,R7
    ADD A,R5
    MOV R3,A

    MOV A,R6
    ADDC A,R4
```

```
    MOV R2,A

    MOV A,CAR_TON
    ADDC A,CAR_TOFF
    MOV  R1,A       ;OUTPUT in R1(MSB) , R2, and R3.
    RET
;-------------------------------------------------------------------------DIV
_24bit Sub-Routine
DIV_24bit:
    ;R0= LSB divisor
    ;R1 =2nd MSB divisor
    ;R5 =MSB divisor
    ;R2 =LSB dividend
    ;R3 =2nd MSB dividend
    ;R4 =MSB dividend 921658

    ;R6 =LSB quotient = LWO
    ;R7 =MSB quotient = HIH

    ;REM =Remainder lower byte

    CLR A    ;clear
    CLR C
    MOV R6,A
    MOV R7,A
    MOV REM,A

LOOP_SUB:

    CLR C             ;updates
    MOV A,R2
    MOV B,#10d
    CJNE A,B,LSST
LSST:   JNC LSSST
    MOV A,R2
    MOV REM,A
LSSST:

    CLR C
    MOV A,R2
    SUBB A,R0   ;R2=R2-R0-C
    MOV R2, A

    MOV A,R3
    SUBB A,R1   ;R3=R3-R1-C
    MOV R3,A

    MOV A,R4
    SUBB A,R5   ;R4=R4-0-C
    MOV R4,A
```

```
    MOV B,#0FEH     ;carry overflow to msb of dividend ends program
    CJNE A,B,LL
    LJMP EXITT

LL:
    JNC EXITT
    INC R6
    MOV A,R6
    MOV B,#0B
    CJNE A,B,FNNN   ;if R6 resets itself , r7 increases
    INC R7



FNNN:
    CLR C
    LJMP LOOP_SUB
EXITT:
    MOV LWO, R6
    MOV HIH, R7
RET

MUL16_bit:
; First number in R6(H) and R7
; Second number in R4(H) and R5.
; Result oin R0(MSB), R1, R2 and R3(LSB).


 ;Multiply R5 by R7
 MOV A,R5
 MOV B,R7
 MUL AB
 MOV R2,B
 MOV R3,A

 ;Multiply R5 by R6
 MOV A,R5
 MOV B,R6
 MUL AB
 ADD A,R2
 MOV R2,A
 MOV A,B
 ADDC A,#00h
 MOV R1,A
 MOV A,#00h
 ADDC A,#00h
 MOV R0,A

 ;Multiply R4 by R7
 MOV A,R4
 MOV B,R7
 MUL AB
```

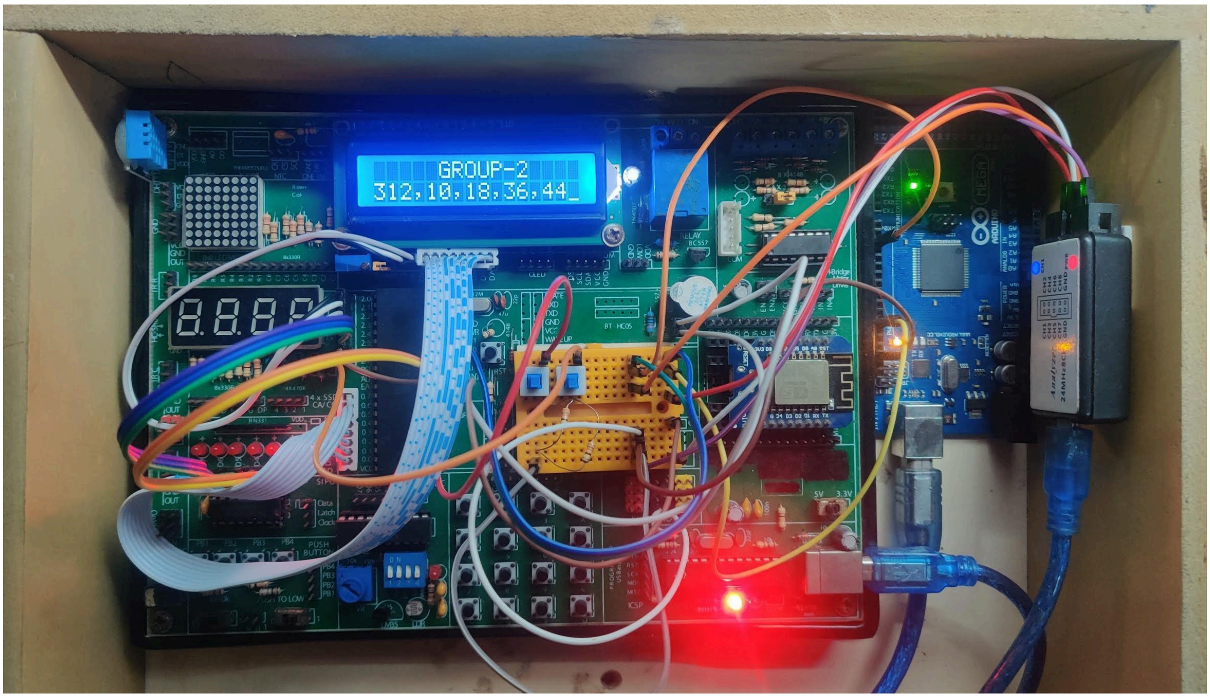
```
 ADD A,R2
 MOV R2,A
 MOV A,B
 ADDC A,R1
 MOV R1,A
 MOV A,#00h
 ADDC A,R0
 MOV R0,A

 ;Multiply R4 by R6
 MOV A,R4
 MOV B,R6
 MUL AB
 ADD A,R1
 MOV R1,A
 MOV A,B
 ADDC A,R0
 MOV R0,A

 ;Return - answer is now in R0, R1, R2, and R3
 RET

;------------------------------------------------------------------------------
--------Messages
MSG_1:  DB "     GROUP-2    ",0 ; Messages that were shown
MSG_2:  DB "312,10,18,36,44",0
MSG_3:  DB "FREQ-1:",0
HZ:     DB "HZ",0
MSG_4:  DB "DIFF:",0
MSG_7:  DB "DC:",0
MSG_8:  DB "FREQ-2:",0
MSG_9:  DB "NEW FREQ:",0
    END
```

## Hardware Implementation:




An Arduino Mega is used to generate two signals of different frequencies and duty cycles and provided to the 8052 microcontroller board . The generated signal from 8052 microcontroller is observed using a logic analyzer. Also the signal generated from Arduino mega is also monitored using the logic analyzer.

## Problems Faced:

1) Had to figure out the memory location of the timer 2 , manually labelled the memory locations as mide doesn't recognize their names; and used it accordingly. For this, we had to go through INTEL and ATMEL datasheet of 8052 microcontroller.
2) When working with high frequencies (>10KHz) , using machine cycles for other purposes becomes significant in calculations (which we used to ignore but they do consume some machine cycles) and when calculations contain very less number of machine cycle , those unconsidered machine cycles ends up creating about 5% error in worst case scenario.
   Eg. (10000hz shows 10010hz – 0.1 percent error)
3) When proteus crashes in middle of the work, the whole project files gets corrupted. Its better to make multiple iterations while working to be able to start working from the previous checkpoints.
4) The closest machine cycle to represent one second using loop is 921660 machine cycles (14d*F004h) which is 2 machine cycle more than the

actual 921658 machine cycles. So for large frequencies, creates about 3% error in worst case scenario.

5) Hardware LEDs and alarms are active low. We used to forget that and had to debug for hours. It is very important to remember.
6) Improper stack PUSH POP made debug a lot harder.

## Conclusion:

The project's goal was to familiarize the team with the use of 8051 family microcontrollers and interfacing with various electronic components. The goals were to generate frequencies, measure the frequency and duty cycle of external signals, and implement features like the ability to stop the microcontroller in various modes. The required features of measuring and displaying frequencies, creating a new signal based on input frequencies, and controlling LEDs in accordance with tasks completed were all successfully completed by the project. In addition, we added optional features like LED indicators for frequency comparison and trigger mode.

We had to deal with difficulties in controlling machine cycles for high-frequency computations, troubleshooting hardware components, and making sure stack management was done correctly. Despite these challenges, the project was successfully implemented, and the team gained valuable experience in working with microcontrollers and electronic components.