UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA
Faculty of Engineering, Built Environment and Information Technology

PRACTICAL REPORT

EMK: 310

HOME ASSIGNMENT: AM I BREATHING?

Name and Surname	Student Number	Signature	% Contribution
			100

February 1, 2023

Contents

1 INTRODUCTION

During the current covid-19 outbreak, the need arose for a detection of the breath rate to help medical staff with the treatment of covid-19 patients. The goal of this project is to be able to make the sensor and micro-controller simple and cheap.In order for this to be effective the sensor needs to non invasive thus not requiring approval from the Health society, the program should also be able to reliable detect the breath rate and warn hospital staff of irregular breathing rates.

1.1 PROJECT DEMO VIDEO LINK

https://drive.google.com/file/d/1IKxdAVF1OsJM3sJiu60VimgDDXMhgLyO/view?usp=sharing

2 RESPIRATORY SENSOR

2.1 IDEA

The idea was to use an ultra sonic distance sensor as seen in figure 1 below to measure changes in distance between it and the patients chest during berating but found during test depicted in the results section below that the sensor was to erratic to be used reliably no matter how close it was positioned to a users chest.

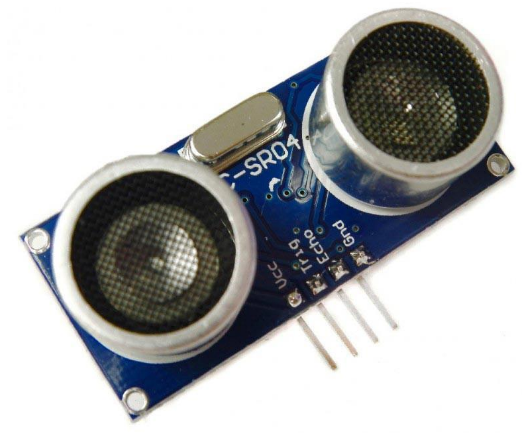


Figure 1: Ultra sonic distance sensor

Thus the final solution was to use a 10kΩ potentiometer connected to VDD and GND with V_{out} connected to pin A1 which feeds into the ADC.

2.2 RESULTS

The results were the the above sensor, even though the data sheet claimed it could detect a change in distance of 2mm did not yield reliable results, thus as mentioned above was to use a 10kΩ potentiometer.

3 ADC IMPLEMENTATION

An ADC is used to convert the analogue signal into a digital signal.

3.1 DESCRIPTION OF IMPLEMENTATION

In order to setup the ADC it was decided that a greater voltage range was desired rather than smaller a smaller voltage rage with greater precision. Thus the following setup was implemented found in the code below

```
;ADC Pin Setup
BSF                 TRISA,1         ;Set RA1 as input
BSF                 ANSELA,1         ;Set RA1 to analog

;ADC Setup (Page 289)
;ADCON0
MOVLW        b'00000101' ; AN1,ADC on
MOVWF        ADCON0
;ADCON1
MOVLW        b'00000000' ; ADC ref = Vdd,Vss
MOVWF        ADCON1
;ADCON 2
MOVLW        b'00101110'; left justify
MOVWF        ADCON2      ; Frc , Acquisition Time
BSF          ADCON0,GO  ;Starts the adc module
```

ADC Setup

As one can see from the above code ADCON0 sets the ADC pin and turns the module on, in this case it is set to port A1 which has been setup as an analog input.

The next register that is configured is the ADCON1 register which is to set the V_{REF+} and V_{REF-} voltage references which for this practical are 5V and 0V respectfully.

As stated above the goal was a larger range thus left justify was chosen for the alignment of the result this means that the 10bit value of the ADC has the most significant 8 bits stored in the ADRESH register while the least significant results of the ADC conversion contained in the ADRESL register are ignored. Thus with

Moving Average Algorithm

The data from the ADC is however very noisy and is subject to sudden fluctuations thus it was decided that a form of filtering needed to be implemented. This is done with a moving average filter which is described by the equations found below,

$$y(n) = bx(n) + bx(n-1) + ...bx(n-M) \tag{1}$$

Above is the general equation of the moving average filter where the average is computed of M samples but in this implementation the average is only taken with 2 samples, thus the value of b can be defined as,

$$b = \frac{1}{(2M)} \tag{2}$$

thus,

$$b = \frac{1}{4} \tag{3}$$

Thus the final equation the describes the coded averaging function is,

$$Y_{new} = \frac{Y_{old}}{4} + \frac{X_{sample}}{4} \tag{4}$$

where X_{sample} is the new value from the ADC and Y_{old} is the previous averaged result. The code implementation of the above equation can be seen below,

```
ADC_FUNC:
   clrf      Count
   MOVLW     .64
   MOVWF     Count
Sample_Buf
;Sample and return the sum
   BSF       ADCON0,GO
   BTFSC     ADCON0,GO
   GOTO      $-2 ;$-2 go to previous instruction
   MOVFF     ADRESL,NumL
   MOVFF     ADRESH,NumH
Average
;Moves value from STORED to STORE_OLD without effecting STORED
   MOVLW     0xFF
   ANDWF     STORED,0
   MOVWF     STORE_OLD
   BCF       STATUS,C         ;Clear the carry flag
    ;Shift is the same divide
   RRCF      STORE_OLD,1         ;Divide by 2
   BCF       STATUS,C
   RRCF      STORE_OLD,0         ;Divide by 2 again
   SUBWF     STORED,1         ;STORE_OLD/4 - STORED
    ;
   BCF       STATUS,C         ;Clear the carry flag
   RRCF      NumH,1
   BCF       STATUS,C
   RRCF      NumH,0
    ;
   ADDWF     STORED,1
   BCF       STATUS,C
   DECFSZ    Count
   goto      Sample_Buf
   MOVFF     STORED,CCPR4L
   RETURN
```

3.2 EVIDENCE OF CORRECT FUNCTIONING

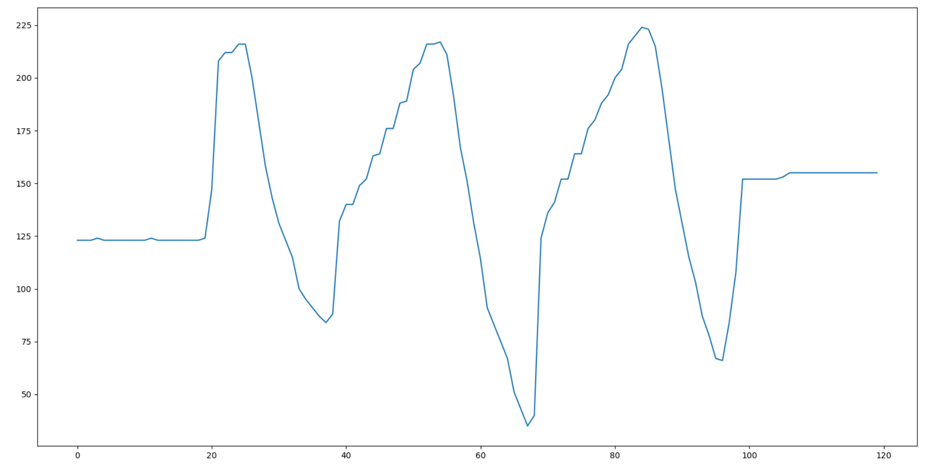


Figure 2: Graph of 3 breaths from the ADC with the moving averaged turned on

4 CALIBRATION

4.1 DESCRIPTION OF IMPLEMENTATION

The calibration is used to implement an offset for the respiratory sensor, which can be tuned with a potentiometer this means that the device will perform to the same standard no matter where and on whom it is being used.

The calibration subroutine is implemented as the first function in the program, after the setup of the micro-controller the lcd will print the message ”Set sensor to 127: xxx” where 127 is the midpoint of the ADC reading and xxx is the current ADC output from the sensor. This needs to be done when the patient is neither berating in or out. Once this is done the operator needs to just push a button to complete the calibration. This can present a problem that a patient unconscious or due to other health concerns may not be able to complete the above calibration process thus this mode can help calibrate in a second way.

Since the breath-rate detection does not require calibration and does not need to centred at an ADC reading of 127. The only requirement is that the ADC reading does not exceed the 0 to 255 range of the ADC module for reasons why see section 5. Thus the calibration mode can be used to set the respiratory sensor so that it does not exceed the bounds stated above since the LCD is outputting the real-time data from the respiratory sensor in this mode thus a second form of calibration can be performed if the previous method is not possible to implement.

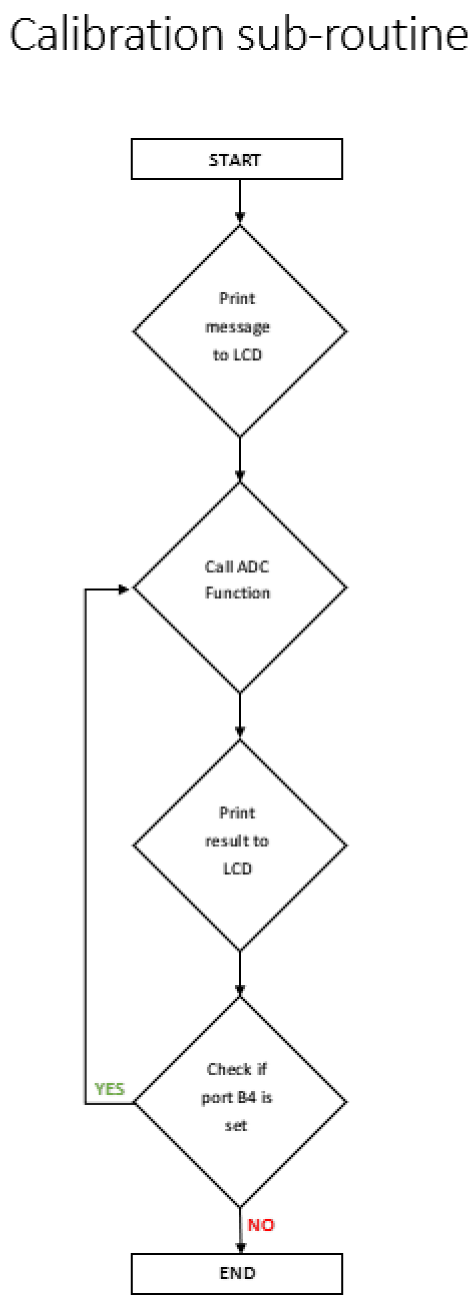


Figure 3: Calibration function flow diagram

4.2 EVIDENCE OF CORRECT FUNCTIONING

For evidence of correct functioning please see the demo video.

5 RESPIRATION RATE DETECTION

The below implementation of respiration rate detection makes use of the ADC and timer interrupts. The reason I chose this approach was because it does not require calibration and can self adjust to to how deep or shallow the patient is breathing.

5.1 DESCRIPTION OF IMPLEMENTATION

The respiration rate makes use of continuously sampling the ADC function and comparing the current result to the previous result while this is happening a timer interrupt is being triggered and increments a counter see section 9. When the current result is a change from a positive value to a negative value (the change from breathing in to breathing out) the function calls the value from the counter and calculates the BPM from the formula below,

$$BPM = \frac{countervalue}{valueof60s} \tag{5}$$

This value is then averaged using the same moving average function implemented on the ADC see section 3.

In order to obtain a more accurate BPM value smaller increments need to be used that is why the counter for final implementation is incremented every 250 ms thus the value of 60s is 240.

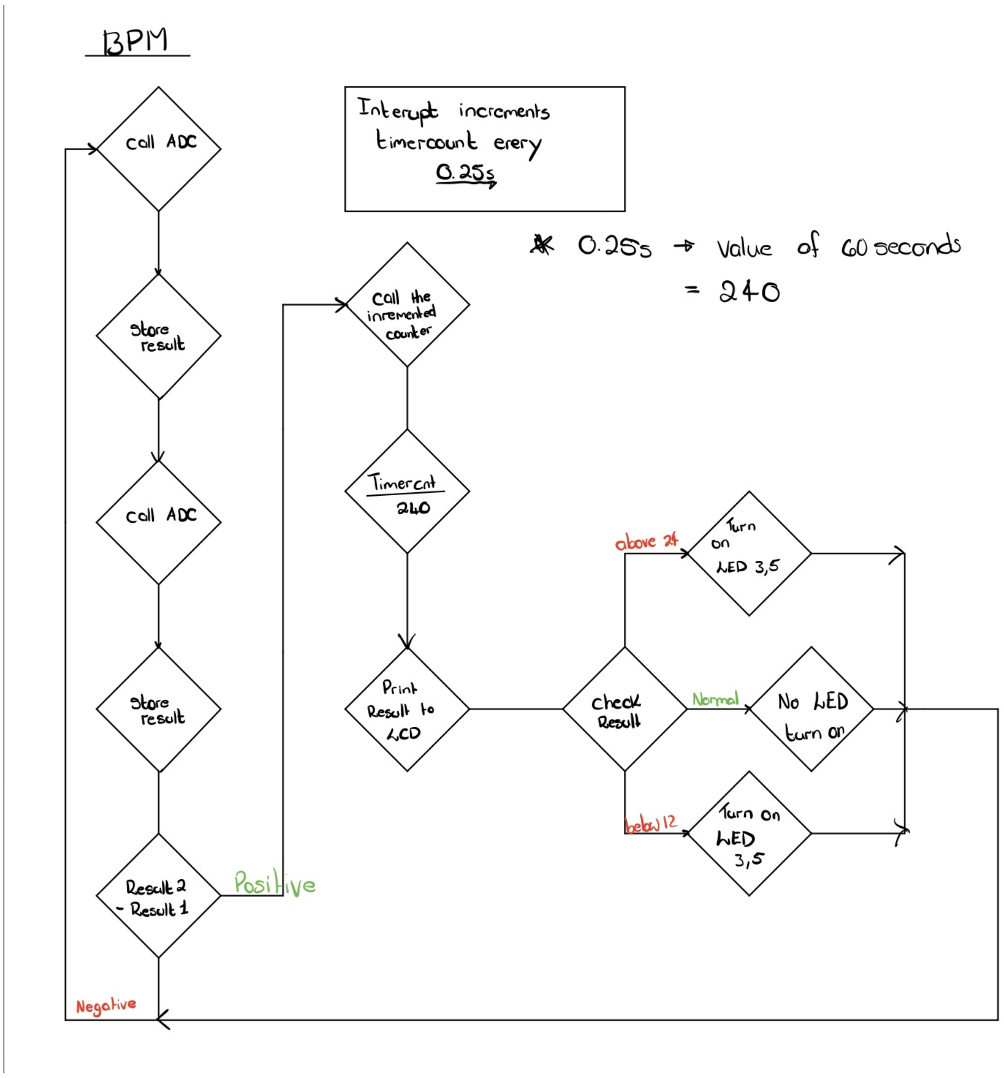


Figure 4: FLow digram showing how the BPM program is implemented

5.2 EVIDENCE OF CORRECT FUNCTIONING

For evidence please see the demo video.

6 BREATHING DIFFICULTY ALERT

6.1 DESCRIPTION OF IMPLEMENTATION

I The breathing difficulty alert is taken from the national health organisation the a breath-rate of below 12 and above 24 are cause for concerns, thus the program takes the BPM obtained in section 5 and compares it with a value for 12 BPM, and 24 BPM using the ’CPFSGT’ and ’CPFSLT’ functions respectfully based on this three possible options are possible

1. BPM to low In this state the programs functions as normal but a berating difficulty warning is displayed on the LCD and the LED’s change see section 7 to indicate what breathing difficulty the patient is experiencing.

   ```
   DifficultyL:
       BCF     PORTA,6
       BCF     PORTA,7
       CALL    Cursor131
       MOVLW   'B'
       CALL    LCD_PRINT
       MOVLW   'D'
       CALL    LCD_PRINT
       MOVLW   '!'
       CALL    LCD_PRINT
       BSF     PORTA,5
       BSF     PORTA,7
       RETURN
   ```

2. BPM normal In the normal BPM the LCD just displays the standard information as described in section 5

3. BPM to high In this state the programs functions as normal but a berating difficulty warning is displayed on the LCD and the LED’s change see section 7 to indicate what breathing difficulty the patient is experiencing.

   ```
   DifficultyF:
       BCF     PORTA,6
       BCF     PORTA,7
       CALL    Cursor131
       MOVLW   'B'
       CALL    LCD_PRINT
       MOVLW   'D'
       CALL    LCD_PRINT
       MOVLW   '!'
       CALL    LCD_PRINT
       BSF     PORTA,5
       BSF     PORTA,6
       RETURN
   ```

6.2 EVIDENCE OF CORRECT FUNCTIONING

See the demo video.

7 LED OUTPUT

In order for the pins to control the LEDs' connected to them they need to be configured as digital outputs this is done by clearing the tristate and the analong select register respectfully as seen the code excerpt below,

```
BCF                 TRISA,5;Set RA4 as output (Curiosity led D3)
BCF                 ANSELA,5;Set RA4 as digital
```

7.1 DESCRIPTION OF IMPLEMENTATION

The LEDS' in the above figure are configured to turn on for the following conditions

- LED1:
  LED 1 is connected to port D1 which is configured as PWM output see section 8 for details.
- LED2:
  LED 2 is connected to port A4 which turns on when the patient is breathing IN and turns off when the patient is breathing OUT.
- LED3:
  LED 3 is turned on when a breathing difficulty alarm is triggered.
- LED4:
  LED 4 indicates with LED 3 that the BPM is above 24 BPM.
- LED5:
  LED 5 indicates with LED 3 that the BPM is below 12 BPM.

7.2 EVIDENCE OF CORRECT FUNCTIONING

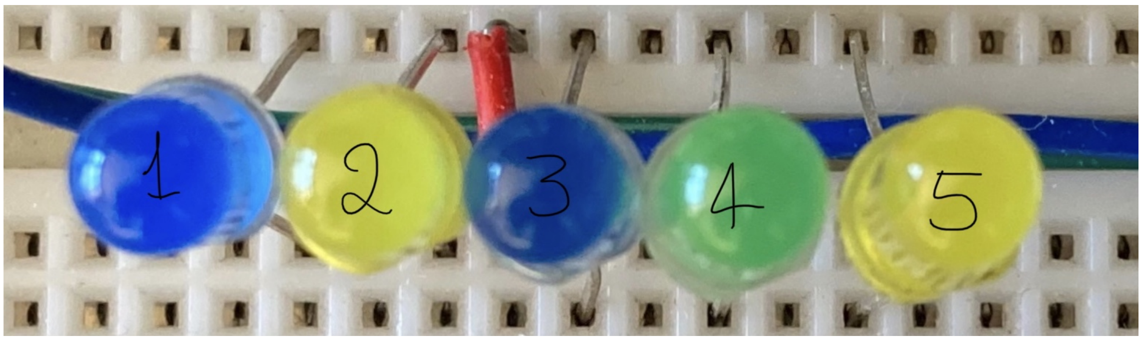


Figure 5: Figure showing LED numbering

For evidence of correct functioning please see the attached demo video

8 PWM OUTPUT

PWM stands for pulse width modulation and is used to control motor speeds and servos it can also be used to control the brightness of a LED by adjusting the duty cycle which is the amount in percentage that the pin is set high for instance a 50% duty cycle means that the pin is 'high' for half of the period and 'low' for the other half.

8.1 DESCRIPTION OF IMPLEMENTATION

The implementation of PWM in this project is on LED 1 and is used to show the intensity of a breath eg the larger the ADC value the larger the duty cycle becomes and the brighter the led shines. Below is the code used to setup the PWM on pin D1

```
;PWM Setup (Page 180)
;PWM period setup (Formula Page 181)
MOVLW   0xFF
MOVWF   PR2
;Timer2 Setup
MOVLW   b'01111010' ;1:16 Prescale there is no Postscale
MOVWF   T2CON;with timer2 and PWM
;Set Which timer to use
MOVLW   b'00000000'
MOVWF   CCPTMRS0
;Setup the compare module
MOVLW   b'00101100'
MOVWF   CCP4CON
;Setup Pulse width/Duty Cycle (Note its a 10bit value)
BCF     CCP4CON,5 ;Controls the PWM decimal
BCF     CCP4CON,;Controls the PWM decimal
MOVLW   .76
MOVWF   CCPR4L
;Start the PWM on PIN D1
BSF     T2CON,2
```

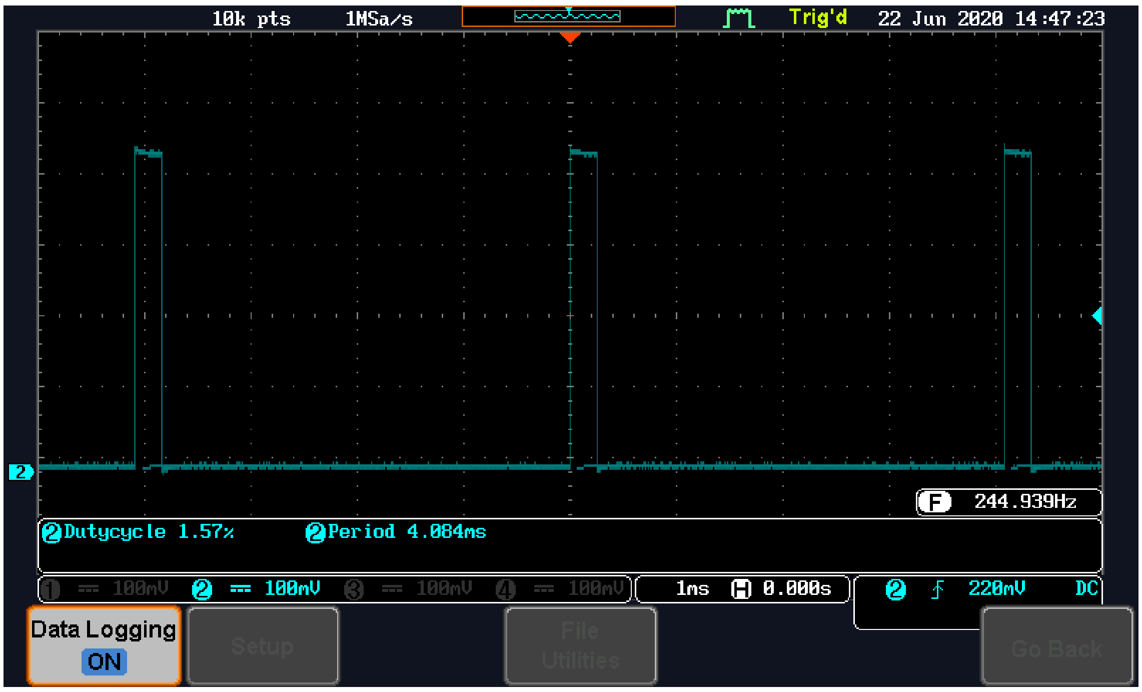
The above code was designed to be able to not only control a brightness of a LED by also control a servo that is why the PWM has a period of 4ms which is 250 Hz which is the required frequency of the servo I will use to control my M.A.R.V. This was configured according to the following formula,

$$PWMPeriod = [(PRx) + 1] \cdot 4 \cdot T_{OSC} \cdot Prescale \tag{6}$$

where $T_{OSC} = 0.25 \cdot 10^{-6}s$ and $Prescale = 16$ thus $PR_x = 255$ thus,

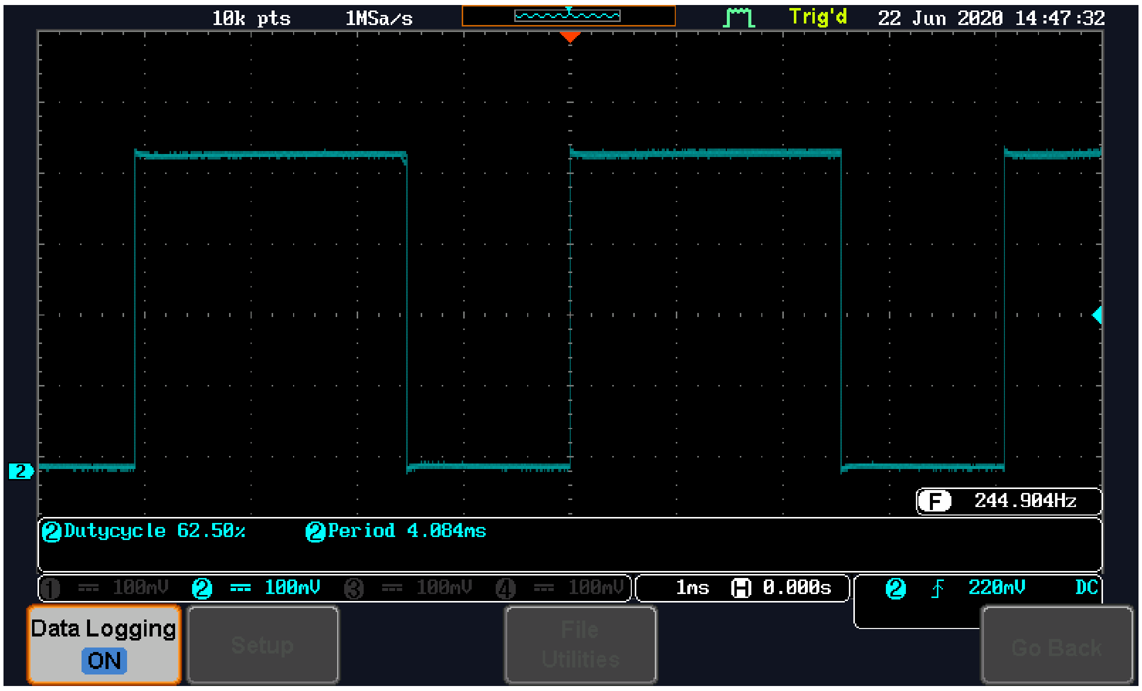
$$PWMPeriod = 4.096 \cdot 10^{-3}s \tag{7}$$

8.2 EVIDENCE OF CORRECT FUNCTIONING



Dutycycle 1.57%   Period 4.084ms

Figure 6: Output from PWM pin D1 showing the duty cycle



Dutycycle 62.50%   Period 4.084ms

Figure 7: Output from PWM pin D1 showing the duty cycle

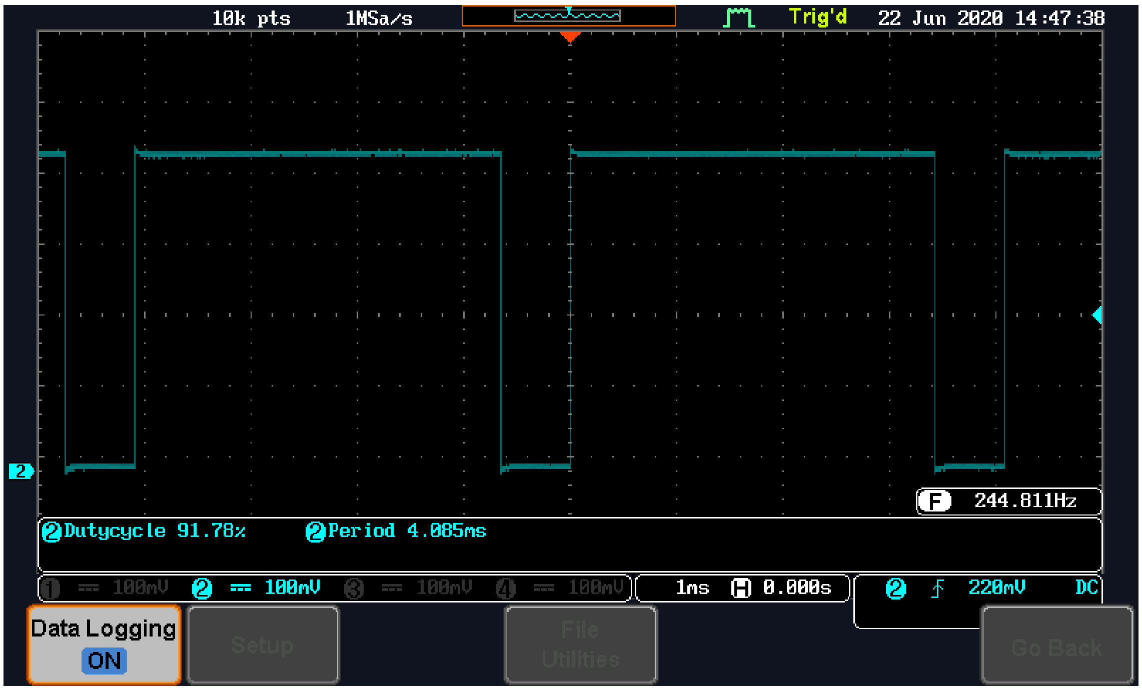


Figure 8: Output from PWM pin D1 showing the duty cycle

9 INTERRUPTS

Interrupts are used in micro-controllers to interrupt the program execute a subset of instructions, the interrupt instructions can either allow the program to return from the instruction and continue or change what the micro-controller is doing completely.

9.1 DESCRIPTION OF IMPLEMENTATION

In the breathing program there are two different interrupts implemented, and based on the code below execute different subroutines based on which interrupt flag is triggered.

```
    org     08h
    BTFSS   INTCON,RBIF
    goto    TIME_TEST
    goto    ISR_PORT
TIME_TEST
    BTFSS   INTCON,TMR0IF
    RETFIE
    goto    ISR_TIME
```

In the above code when an interrupt is triggered the micro-controller moves to 08h in the program memory (since high and low priority interrupt were not implemented) and thus the program tests which interrupt flag was triggered and based on this the following three options occour,

- Change on port B interrupt This interrupt is implemented as part of the capacitive touch system. The interrupt is triggered when a change is detected on port B5 the system then checks if this was a false trigger or if it was a valid trigger for more details see section 8 and based on that pin A7 is either set 'low' or 'high' and then returns to place in the program where is was before the interrupt was triggered.

- Timer interrupt was implemented using timer0 on the pic this timer is chosen because it features a 16 bit counter and the interrupt is triggered when the timer counter overflows. This timer was configured according to code below,

```
Timer Setup              (Page 155)
;Timer0
MOVLW    b'00000001'
MOVWF    T0CON
```

  With a clock of 4Mhz this timer triggers an interrupt at every 0.25s, when this happens the pic executes the following code,

```
ISR_TIME
    BCF     INTCON,GIE       ;Disables global interrupts
    INCF    Timecnt          ;Time inbetween preiods
    INCF    Timecnt1
    BCF     INTCON,TMR0IE    ;Clear the timer flag
    BSF     INTCON,GIE       ;Enables global interrupts
    MOVFF   Timecnt1,WREG
    CPFSEQ  MINUTE           ;If Timecnt = 240 skip
    RETFIE
    CLRF    Timecnt1
    RETFIE
```

  In the above timer interrupt there are two timer counts Timecnt is used to determine the BPM described in section 5 and Timecnt1 is used to keep track of the 60 second timer that is displayed on the LCD screen during normal operation. Global interrupts are disabled at the start on the instructions to prevent another interrupt being triggered turn the execution of the code. After the above instructions are completed the pic returns to instruction it was busy with before the interrupt was triggered.

- Non-valid interrupt thus the program just returns to where is was before the interrupt was triggered.

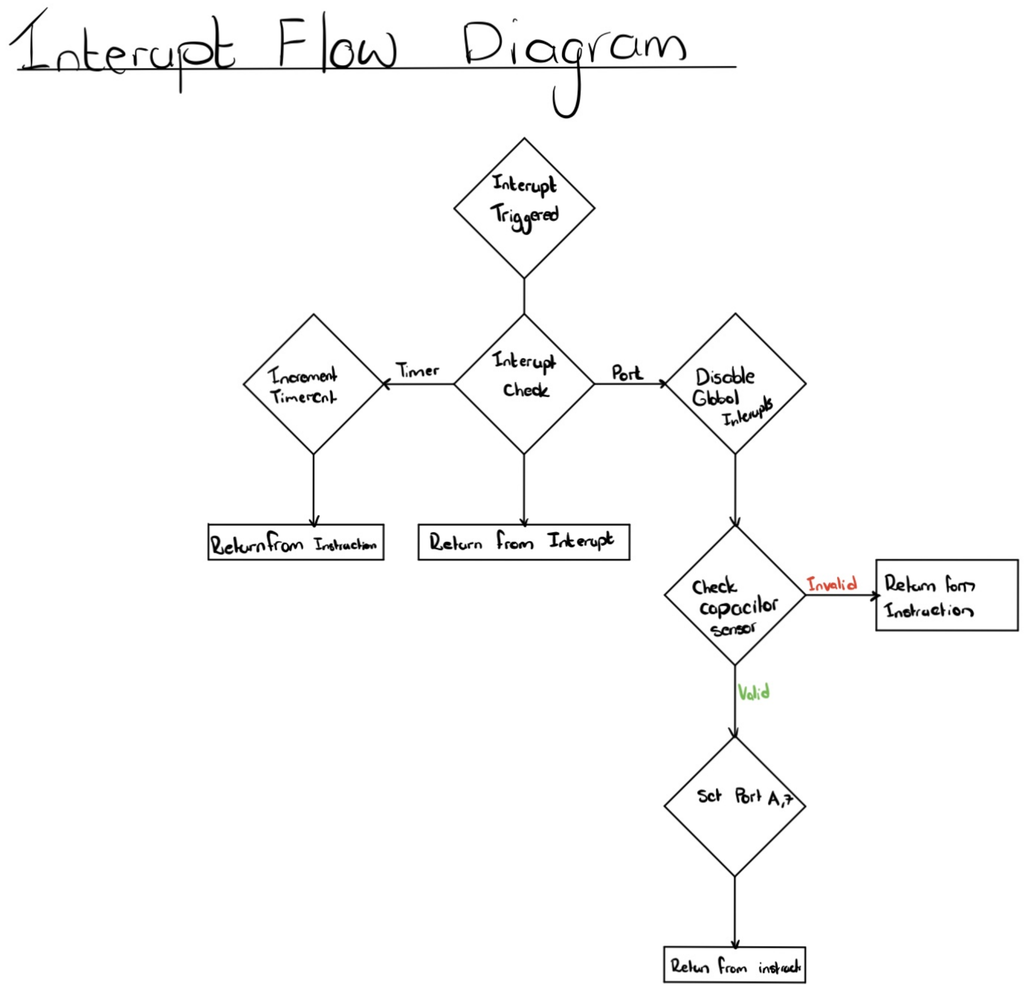


Figure 9: Interupt program flow digram

9.2 EVIDENCE OF CORRECT FUNCTIONING

The program would be able to continue if the interrupts did not work, see demo video for proof.

10 TOUCH START/STOP

10.1 DESCRIPTION OF IMPLEMENTATION

The implementation of the capacitive touch sensor works as follows the charge capacitor is charge by setting the pin to a digital pin that is set to VCC, the ADC is then connected to the capacitive sensor, if the voltage of the capactive sensor is above a threshold voltage this counts as a valid capacitor press if not then it is not a capacitive press. This can be seen in the following code excert,

```
MOVLW    b'10101110'; right justify
MOVWF    ADCON2      ; Frc , Acquisition Time
BSF      PORTA,4
```

```
MOVLW    b'00010001' ; Point ADC to pin B5
MOVWF    ADCON0

BSF      TRISB,5             ;Set RA2 as input
MOVLW    b'00110101' ; Point ADC to pin B5
MOVWF    ADCON0
CALL     ADC_FUNC_1
MOVFF    STORED,WREG
CPFSLT   Cap
```

10.2 EVIDENCE OF CORRECT FUNCTIONING

For evidence of correct functioning please see the attached demo video, as one can see in the video as soon as a finger touches the tin foil plate the program begins to calculate the BPM, and is reset by the same touch button once no breathing has been detected for 60s.

11 INTEGRATION

All of the subroutines described above are integrated into one program. According to the block diagram depicted below, in figure 10

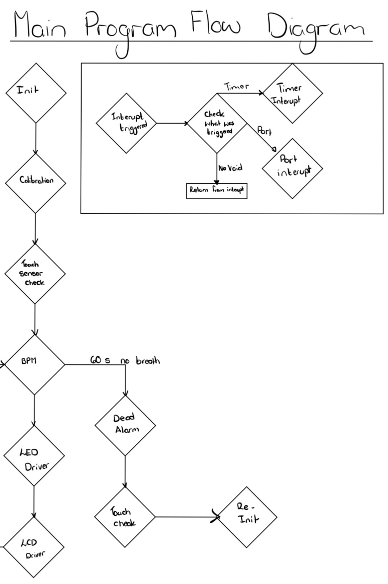


Figure 10: Program flow diagram of the integration

12 CONCLUSION

The following implementation meets all the design requirements that were stated in the project outline were achieved except for the implementation of a function respiratory sensor and had we substituted with an external potentiometer.

The above program is able to demonstrate the calibration , and BPM detection with warnings if the BPM is deemed dangerious.

13 I^2C

13.1 EEPROM

Description of implementation

To demonstrate that the I^2C communication protocol can be implemented and used correctly it was used to print the classic ”HELLO WORLD!” to the LCD screen and to the serial terminal. This was implemented by first writing the charters 'H','E','L','O','W','R','D' and '!' to the EEPROM and then read the values back store them as variables that will then be printed.

The first step in implementing the above was setting the baud-rate of 100 kHz for the I^2C protocol which is required by the LCD driver module, the I^2C module then needs to be set to master mode and the ports C3 and C4 configured correctly.

Once this is done the communication protocol procedure needs to be coded. The details of which can be found on the EEPROMS data sheet but simplified are are as follows for reading and writing 8bits of data respectfully,
Writing:

1. PIC(master) generates the start condition, (High to Low transition of SDA while SCL is high)
2. Control byte is sent which is the address of the device and the write operation (A0 in hex)
3. The slave(EEPROM) device then sends an acknowledge to the master device
4. The master(PIC) then send the word address to the slave
5. The slave(EEPROM) device then sends an acknowledge to the master device
6. The master(PIC) then sends the 8bits of data to the slave
7. The slave(EEPROM) device then sends an acknowledge to the master device
8. PIC(master) generates the stop condition, (High to Low transition of SDA while SCL is high)

Random Read:

1. PIC(master) generates the start condition, (High to Low transition of SDA while SCL is high)
2. Control byte is sent which is the address of the device and the write operation (A0 in hex)
3. The slave(EEPROM) device then sends an acknowledge to the master device
4. The master(PIC) then send the word address to the slave
5. The slave(EEPROM) device then sends an acknowledge to the master device
6. PIC(master) generates another start condition, (High to Low transition of SDA while SCL is high)

7. Control byte is sent which is the address of the device and the raed operation (A1 in hex)
8. The slave(EEPROM) device then sends an acknowledge to the master device
9. The slave(EEPROM) then sends the 8bits of data to the PIC(master)
10. PIC(master) generates the stop condition, (High to Low transition of SDA while SCL is high)

Evidence of correct functioning

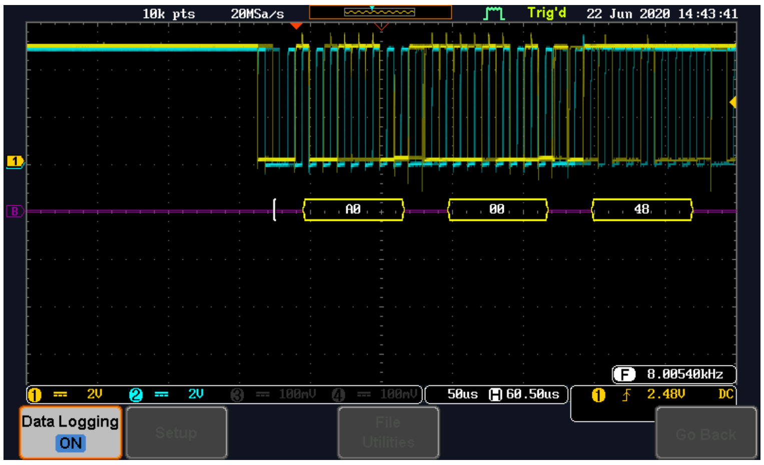


Figure 11: I2C bus decoded addressing the EEPROM writing 0x48 to address 0x00

0x48 is hex converted to ascii is 'H' which is the first letter being written to the EEPROM

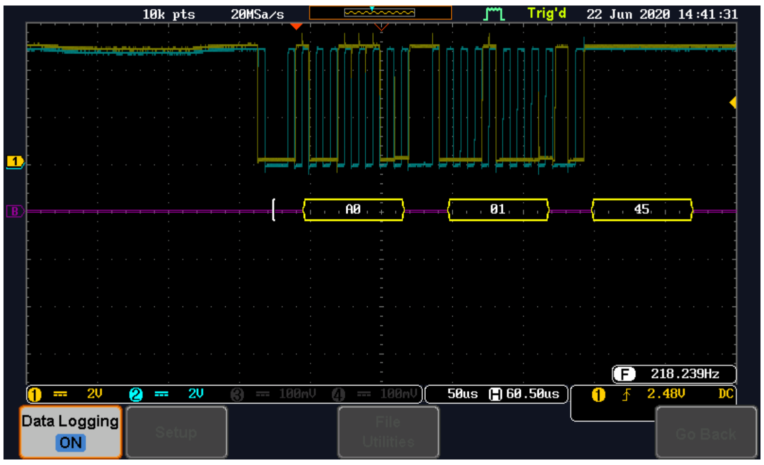


Figure 12: I2C bus decoded addressing the EEPROM writing 0x45 to address 0x01

0x45 is hex converted to ascii is 'E' which is the second letter being written to the EEPROM

13.2 LCD DISPLAY

Description of implementation

The LCD is implemented by sending the command over I2C protocol which involves addressing the module and sending the first 4 bits of data and then addressing the module again and then sending the lower 4 bits what make up a charter or a command.

Evidence of correct functioning

Please see the demonstration video.

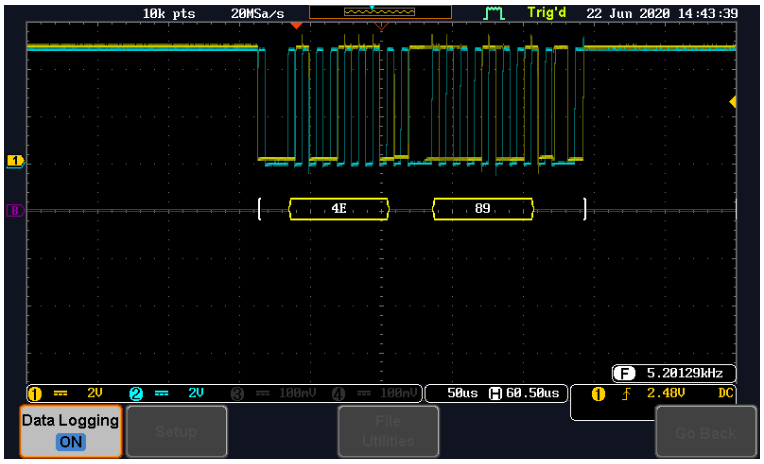


Figure 13: I2C bus decoded addressing the LCD display module