THE UNIVERSITY OF JORDAN

DEPARTMENT OF COMPUTER ENGINEERING

Embedded Systems Lab Report
Oxygen Therapy Controller

SUBMITTED BY

Ehab Younes (0174788)
Zeyad Al Najjar (0175642)

TO INSTRUCTORS

PROF. Iyad Jafar

ENG. Rawan Al-Jamal

(Spring 2020-2021)

Contents

1 Introduction

As per specification, the system presented here is used in hospitals to track and regulate the amount of oxygen used for oxygen therapy. This specific system was designed to accommodate only two beds.

There are three main functionalities that the system performs. First, it interactively controls the oxygen flow for each bed, depending on the need, using control knobs (potentiometers) with the option to turn off the machine for either or both beds. The second functionality is interfacing the oxygen pump and producing the correct voltages for the pump to be able to deliver the total required oxygen flow. The final and third functionality is the tracking of the amount of oxygen in the tank and notifying the user when the amount falls beyond a specified volume.

The user can interact with the system either by using the potentiometers as described above to change the flow rate or by flipping the switches to toggle the oxygen for a certain bed. Alternatively, the user can also see the amount of oxygen left in the tank by switching the LCD View switch they can see whether the system is on or off and the individual flow rate for each bed.

The oxygen tracker can also turn on an alarm (modeled by a red LED) to notify the user that the oxygen volume is low and that the tank must be refilled.

2 System Description

2.1 Subsystems

The system can be split into four different subsystems, each performing a specific functionality:

- **Rate Control Knobs**: Two different potentiometers are used to control the flow rate of the oxygen for each bed. The potentiometer for bed1 is connected to RA0/AN0, while the potentiometer for bed2 is connected to RA1/AN1. For the values of V_{s1} and V_{BAT}, we have chosen $V_{s1} = 5V$ and $V_{BAT} = 0V$. We also connected them to RA3/Vref+ and RA2/Vref-, respectively, for more accurate conversion.
- **LCD**: This subsystem displays the volume of oxygen left in the tank. It can also display the flow rate and whether the system is turned on for each bed. The LCD (model LM016L) uses the Hitachi HD44780 controller. So, RC5 - RC7 were connected to RS, RW, and E respectively which can be used to configure the LCD mode of operation. While RD0 - RD7 were connected to the data pins of the LCD to send ASCII characters and the actual configuration commands.
- **Pump**: The pump is responsible for delivering the oxygen rate required from both beds. The pump is represented as a 20V DC motor that can only run at a constant speed. To control the speed of the pump, the pulse-width modulation technique is used where the pump is switched on and off periodically at a high frequency to achieve an average voltage that will run the pump at the required speed.

- **Oxygen Volume Tracker**: This subsystem keeps track of the oxygen volume available in the tank. That's achieved by tracking the consumption rate of both beds and updating the oxygen volume periodically. This information is displayed on the LCD in Mode 1. It's also used to decide when to trigger the alarm mode that turns on the alarm LED and displays a warning message on the LCD.

When integrating the system, we used the Timer 1 module to calculate a 1-second delay. By setting the prescale to the maximum possible (8) and

$$TMR1 = 62{,}500$$

We only need 2 iterations of the timer.

$$Prescale \times TMR1 \times Counter = Total\,Count$$
$$8 \times 62{,}500 \times 2 = 1{,}000{,}000$$

3 Hardware System

3.1 Rate Control Knobs

When configuring the ADC we also connected the Vref+ and Vref- for more accurate values. This has the added advantage of making the values of V_{s1} and V_{BAT} arbitrary (within reason), since as long as we connect V_{s1} to Vref+ and V_{BAT} to Vref-, we'll always read the same value from the ADC.

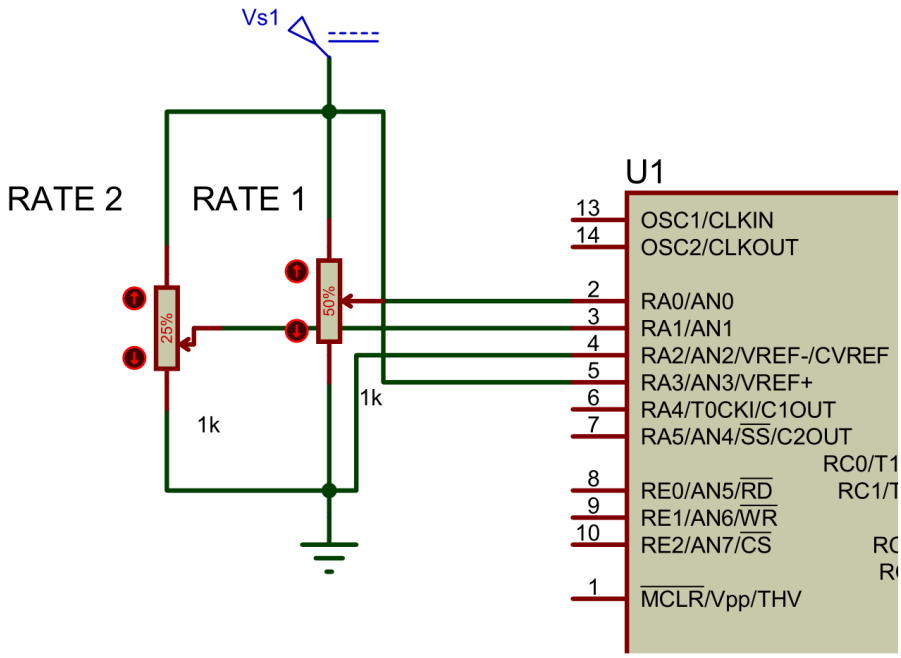


Figure 1: Control Knobs Circuit

Given these values, the resolution:

$$Resolution = \frac{5}{2^8} = 19.53125mV$$

We can then find the ADC result that corresponds to the voltage by the following:

$$ADC\ Result = \frac{V_{in}}{Resolution}$$

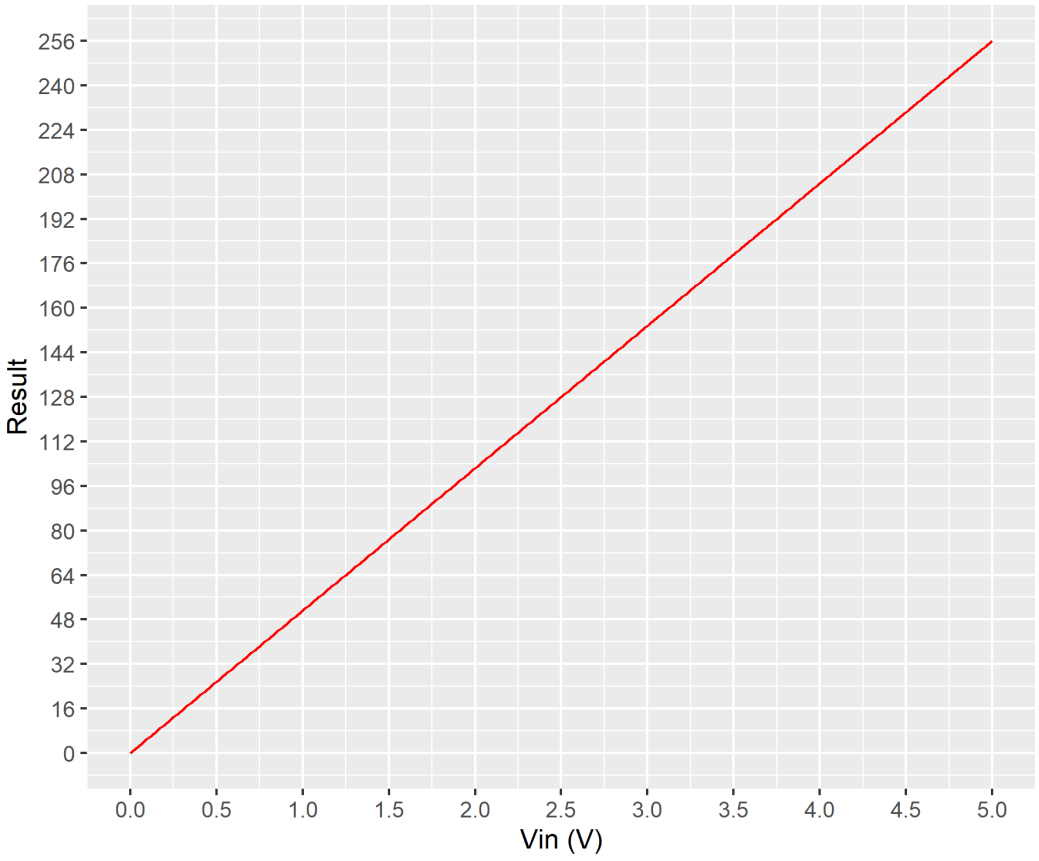


Figure 2: ADC Result vs. V_{in}

Voltage (V)	ADC Result (8-bit)	Oxygen Rate (Liter/sec)
$V_{pot} = 0$	ADRESH $= 0$	0
$0 < V_{pot} \leq 1.25$	$0 < ADRESH \leq 64$	1
$1.25 < V_{pot} \leq 2.5$	$64 < ADRESH \leq 128$	5
$2.5 < V_{pot} \leq 5$	$128 < ADRESH \leq 255$	10

Table 1: ADC Result to Oxygen Flow Rate

Now that we have the digital value (ADRESH). We can compare the key values by subtracting then returning the required oxygen rate in the working register.

```
GET_RATE
    movwf       ADC_VALUE
    sublw       .128
    btfss       STATUS,     C
    retlw       .10                     ; ADC_VALUE > 128
    movf        ADC_VALUE, W
    sublw       .64
    btfss       STATUS,     C
    retlw       5                       ; 128 >= ADC_VALUE > 64
```

```
    movf        ADC_VALUE, W
    btfss       STATUS,    Z
    retlw       1                       ; 64 >= ADC_VALUE > 0
    retlw       0                       ; ADC_VALUE = 0
```

3.2 LCD

The main basic functions of the LCD were the SEND_CHAR which sends an ASCII character to be printed at the current cursor location, and SEND_CMD which changes the configuration of the LCD (To clear the display or change the cursor location).

However, the interesting part of the subsystem is perhaps the subroutine DISPLAY_STR, which prints a C-style string, meaning it keeps printing to the display until it encounters NIL or 0x00.

This, along with the STR_LOOKUP that returns a character based on the display mode (Mode 1, Mode 2, and Alarm Mode). It is done by performing PCL + CURRENT_MODE which moves the PC to the actual lookup table.

```
STR_LOOKUP
    movf        CURRENT_MODE, W
    addwf       PCL, F
    goto        MODE_1_LOOKUP
    goto        MODE_2_ON_LOOKUP
    ...

MODE_2_ON_LOOKUP
    movf        STR_INDEX, W
    addwf       PCL, F
    dt          "Bed", 0x00
    dt          " ON  ", 0x00
    dt          " L/s", 0x00
```

Now the higher level subroutine simply prints the strings and fills the required custom data in between the strings, like, the bed number, the flow rate, or the oxygen volume.

Another interesting subroutine is the GET_BCD_BIG (and GET_BCD_SMALL) which converts a number to packed BCD. Using the inexpensive "Double Dabble" algorithm that only requires shifting and adding 3.

Since the oxygen volume requires more than 8-bits it was split into OXYGEN_VOLUME_LO and OXYGEN_VOLUME_HI, then arithmetic and logical operations are done first on the low-byte then the high-byte while taking care to propagate the carry (See GET_BCD_BIG and SUB16).

3.3 Pump

As mentioned earlier, the PWM technique works by alternating the signal between high and low at a high frequency. The total time for each cycle is called the period (T) and the percentage of time the signal stays high (ton) is called the duty cycle. This can be shown in the following graph:

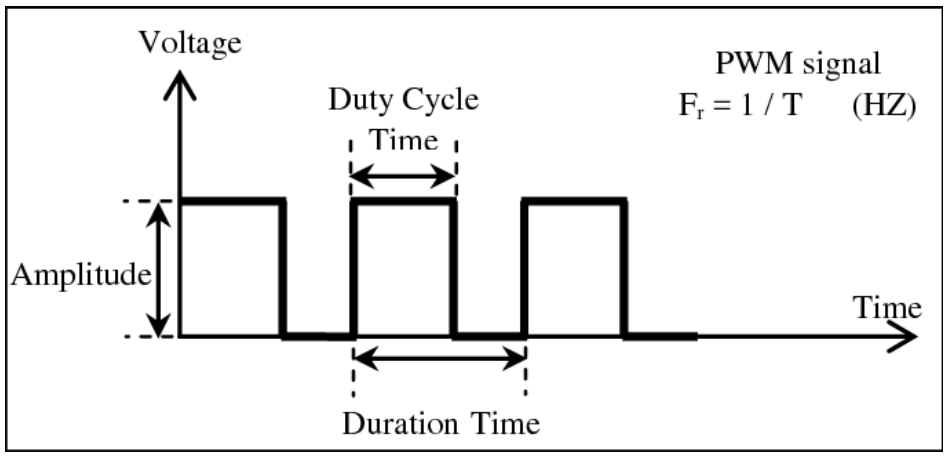


Figure 3: PWM

By adjusting the duty cycle, we can change the average voltage applied to the motor to achieve the required speed for the pump.

Another important point to mention is that the pump operates at 20 volts whereas the PIC microcontroller can only provide 5 volts so, we used an N-Channel MOSFET (2N6660) to be able to control the pump through the following circuit:

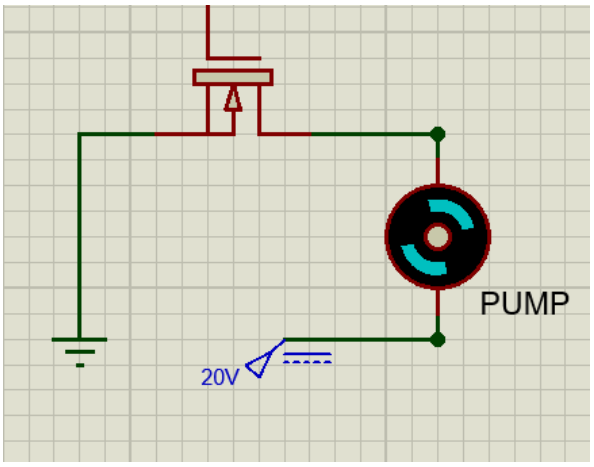


Figure 4: Pump Circuit

When the microcontroller outputs a high signal, the transistor switches on connecting the motor to the ground and completing the circuit which lets the current pass through the pump switching it on. When the output is set to low, the transistor goes into the cutoff mode which creates an open circuit and disconnects the circuit switching the pump off.

Now, each oxygen rate range is achieved by setting the pump to work at a specific speed. These values are shown in Table 2.

Total Oxygen Rate (Liter/sec)	Speed (RPM)
$Rate = 0$	0
$0 < Rate \leq 5$	10
$5 < Rate \leq 10$	40
$10 < Rate \leq 15$	70
$15 < Rate \leq 20$	100

Table 2: Rate to RPM

The required voltage to be applied to the pump to achieve these speeds can be calculated with:

$$V_{avg} = \sqrt{\frac{speed}{4}}$$

To get these voltages, we need to use appropriate values for the period and duty cycle. We chose a period of 1/2500 seconds giving us a frequency of 2.5KHz which after testing appeared to be the best option for both stability and accuracy. The period is fixed for all speed levels whereas the duty cycle varies. The duty cycle is calculated with:

$$Duty\,cycle = \frac{V_{avg}}{20} \times 100\%$$

Now, we can use these percentages to calculate the ton which is the amount of time the signal stays on:

$$t_{on} = Duty\,cycle \times T$$

These calculations are shown in Table 3.

Speed (RPM)	V_{avg} (V)	Duty cycle (%)	$t_{on}(\mu S)$
0	0	0	0
10	1.58	7.9	32
40	3.16	15.8	63
70	4.2	20.9	83
100	5	25	100

Table 3: PWM Calculations

3.4 Oxygen Volume Tracking

This subsystem is responsible for tracking the consumption of oxygen and the available oxygen volume. It works by updating the oxygen volume each cycle, subtracting the total oxygen rate for that cycle from the oxygen volume.

It then checks if the remaining volume is less than the alarm volume (500) and if so, it sets the alarm flag to high which alerts the LCD subsystem to display the warning message.

Both of these operations introduced a bit of a challenge because the oxygen volume is a 16-bit value and the PIC microcontroller doesn't support 16-bit operations (see the obstacles section).

Both the Max Oxygen Volume, which the oxygen volume is initialized to and is set to when the fill button is clicked, and the Alarm Volume are stored in constants that can be easily changed in case the system has to be deployed in different environments.

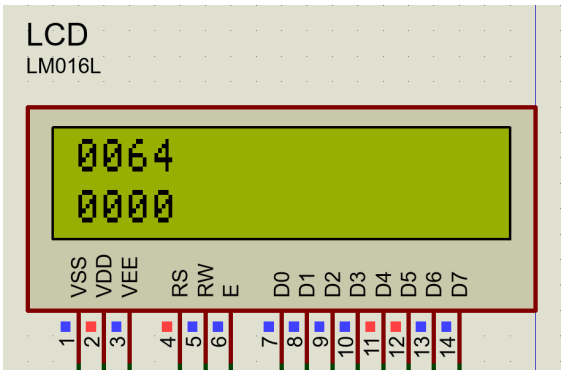
4 System Testing and Results

At first, each individual subsystem was implemented and tested on its own.

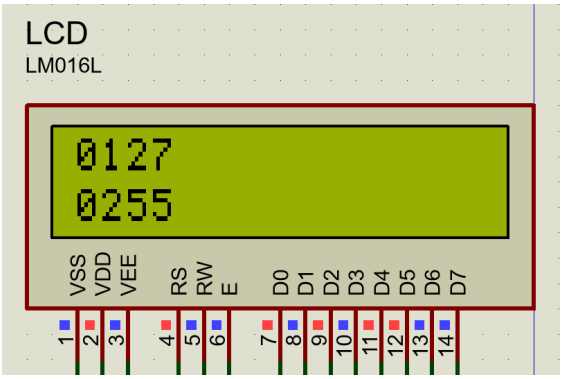
4.1 Rate Control Knobs

Since the first thing we implemented was the LCD, we were able to actually print the register values to the LCD to test it.

Here are some various percentages of both knobs (0%, 25%, 50%, and 100%). As we can see the values are correct.



Rate1 at 25% and 0%



Rate1 at 50% and 100%

```
call    READ_ADC_VALUES

        movf            ADC_VALUE1, W
        movwf           NUMBER_LO
        clrf            NUMBER_HI
        call            GET_BCD_BIG
        SEND_BCD        BCD_HI
        SEND_BCD        BCD_LO

        call            GO_TO_LINE2

        movf            ADC_VALUE2, W
        ...
```

4.2 LCD

To test the DISPLAY_STR subroutine in the LCD, we put different texts and separated them with NIL (0x00) then tried to call DISPLAY_STR various times to see if it prints the required string.

For example, if we execute this code then we expect it to NOT print the "4th" string, since it is not reached:

```
        clrf                STR_INDEX
        call                DISPLAY_STR_TEST      ; 1st string

        call                GO_TO_LINE2

        call                DISPLAY_STR_TEST      ; 2nd string
        call                DISPLAY_STR_TEST      ; 3rd string
        movlw               0xD2
        movwf               NUMBER_LO
        movlw               0x04
        movwf               NUMBER_HI             ; 0x04D2 = (1234) base 10
        call                GET_BCD_BIG
        SEND_BCD    BCD_HI
        SEND_BCD    BCD_LO

TEST_LOOKUP
        movf                STR_INDEX, W
        addwf               PCL, F
        dt                  "Text on line1", 0x00
        dt                  "2nd", 0x00
        dt                  "3rd", 0x00
        dt                  "4th", 0x00
```

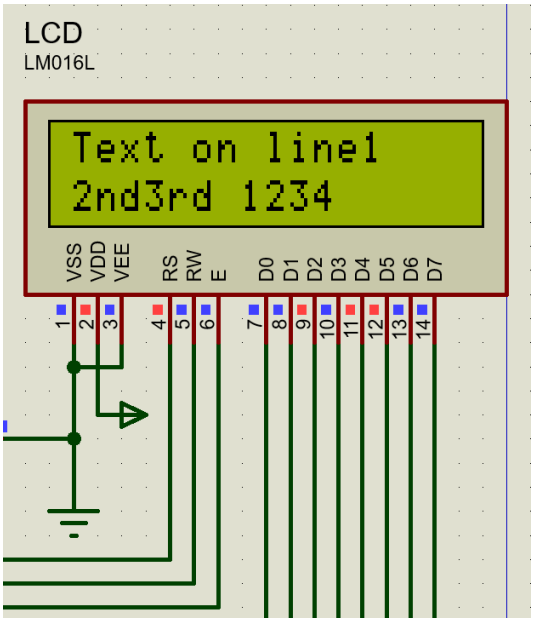


Figure 5: LCD Test

Notice also that 0x04D2 which is 1234 in base 10 has also been printed correctly.

4.3 Pump

To make sure our calculations are correct and the pump subsystem is working as expected, we used an AC voltmeter which measures the RMS voltage of the pump.

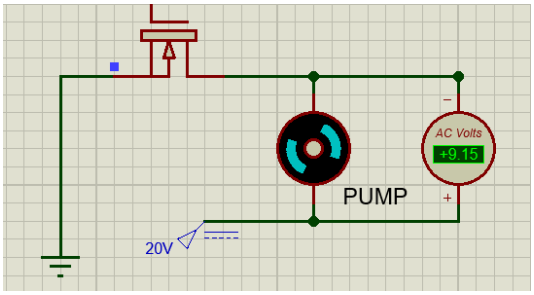


Figure 6: Pump Test

Then we used this value to calculate the duty cycle with the following formula:

$$V_{RMS} = \sqrt{\frac{1}{T}\int_0^T f(t)^2 dt}$$

Since this is a square wave that is either 20V or 0 (Assuming ON is from 0 to t_{on}):

$$V_{RMS} = \sqrt{\frac{1}{T}\int_0^{t_{on}} 20^2\, dt}$$

Finally:

$$Duty\ cycle = \frac{t_{on}}{T} \times 100\% = \frac{V_{RMS}^2}{400} \times 100\%$$

After that, we compared the measured values of the duty cycle with the theoretical values and made sure they're all within the margin of error.

4.4 System Integration

Finally, after testing each subsystem fully, we created the system by combining all the subsystems. Which was then tested as per the project specification.

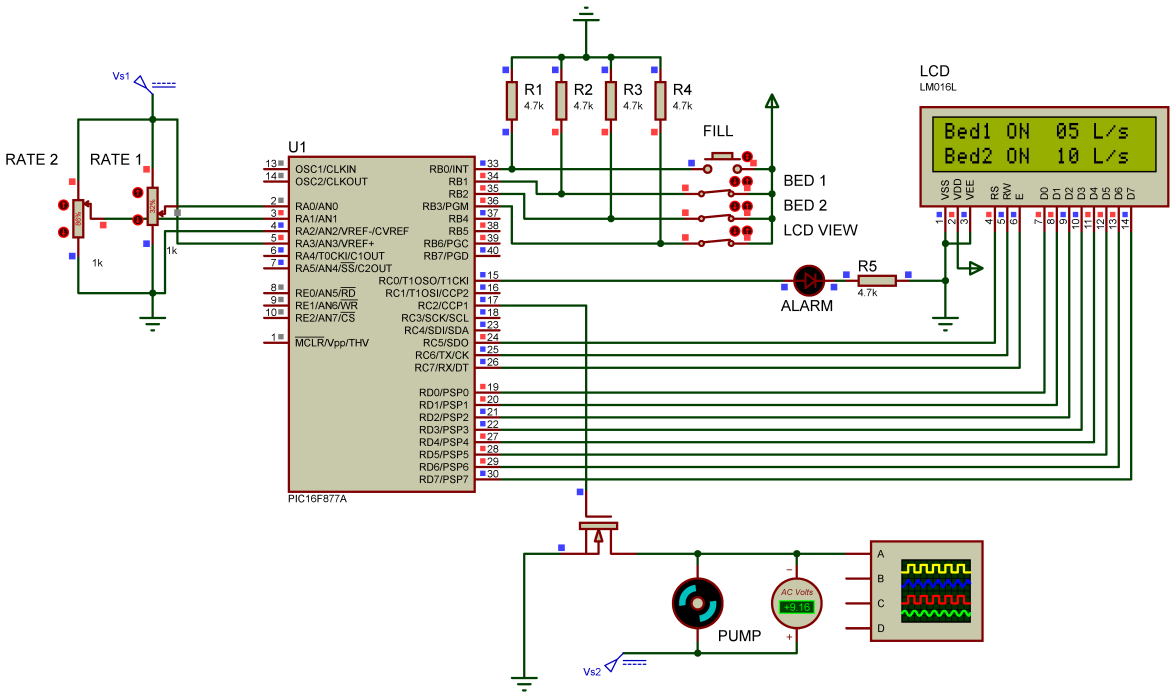


Figure 7: Full System Design

Here, we tested different input combinations, toggled all the switches and pushed the button. Everything was working correctly as specified.

5 Conclusion

5.1 Summary

At the end of this project, we had a fully functional oxygen therapy system with the option to track the oxygen volume, the oxygen flow rate, control the pump, and notify the user when the tank is running low. The software was modularly designed such that each module or subsystem is independent of the other. This made testing, integration, and modification much easier and more straightforward.

Regarding the hardware, the system design is simple and efficient. All of the components are cheap and readily available, while also the power consumption is low.

Notice how the PROCESS subroutine actually calls each individual subsystem, then in each subsystem there functionality is also split into multiple subroutines.

5.2 Contributions

Since we had 4 different subsystems, we each took 2 subsystems to work on:

- **Rate Control Knobs and LCD**: Ehab Younes
- **Pump and Oxygen Volume Tracker**: Zeyad Al Najjar

System integration and testing was done by both of us together.

5.3 Obstacles

We encountered many obstacles when working on this project, here are the most prominent ones:

5.3.1 Strings and Display Modes

We wanted to create a single lookup table to display the required information, however, there were essentially 4 different display modes. So, instead of creating multiple subroutines and copying DISPLAY_STR 4 different times, we used the look up trick to choose which display mode to print. Now we only have to set CURRENT_MODE to pick the display mode, along with using STR_INDEX to keep track of what we are printing in that display mode.

```
STR_LOOKUP
    movf        CURRENT_MODE, W
    addwf       PCL, F
    goto        MODE_1_LOOKUP
    goto        MODE_2_ON_LOOKUP
    goto        MODE_2_OFF_LOOKUP
    goto        MODE_ALARM_LOOKUP
```

5.3.2 16-bit Subtraction and Comparison

There are two instances where we need to subtract from a 16-bit number. First, when subtracting the total oxygen rate from the oxygen volume. Second, when comparing to see if the oxygen volume has fallen below 500.

To do this operation we created a 16-bit subtraction that also produces the correct carry as described by [1]:

```
SUB16           macro   DST, SRC
    movf        (SRC), W
    subwf       (DST), F        ; Subtract DST(low) - SRC(low)
    movf        (SRC)+1, W       ; Now get high byte of subtrahend
    btfss       STATUS, C        ; If there was a borrow increment SRC(high)
    incfsz      (SRC)+1, W
    subwf       (DST)+1, F       ; Subtract DST(high) - SRC(high)
    endm
```

5.3.3 Converting a Binary Number to Binary Coded Decimal (BCD)

This was also used in two instances. First, when displaying the 16-bit oxygen volume calculated above. Second, when displaying the 8-bit oxygen flow rate.

To find the value of the units, tens, hundreds, and even thousands using loops can be extremely inefficient in code and performance. So I deployed the popular algorithm "Double Dabble" [2] which is mainly used for simple controllers that lack a division unit.

Essentially, this algorithm adds 3 to a BCD digit if it's bigger than or equal 5 before shifting left. Because in decimal 2×5 produces a 2-digit number. This operation (addition of 3 then shifting left) is done until all bits have been processed. To reduce computing even more, I left justified the bits then performed the algorithm in less iterations.

Here's an example of how it works, assuming we want to convert the number $(27)_{10}$ using only the first 5 bits to BCD:

```
0000 0000              00011011      Initialization
0000 0000              11011000      Left justify the lower 5 bits
0000 0001              10110000      Shift
0000 0011              01100000      Shift
0000 0110              11000000      Shift
0000 1001              11000000      Add 3 to ones, since it was 6
0001 0011              10000000      Shift
0010 0111              00000000      Shift (Fifth)
   2    7
```

5.3.4 Inaccurate Timer 0

After we integrated the system and ran a different number of scenarios. We noticed that the oxygen tank reaches 500 Liters after 105 seconds instead of 100 seconds when the rate is 20 L/s.

This was caused by TMR0 drifting off with time, and becoming more noticeable as time went by. After investigation, we found that this is mostly caused by the software counter that has a high value of $(125)_{10}$.

To fix this issue, we used the more accurate 16-bit Timer 1 as described above.

This change resulted in very high accuracy and the oxygen tank reaches the 500 Liter mark at exactly 100 seconds in the described settings.

5.3.5 Stack Overflow

Again, this issue happened when we integrated the system and had to call subroutines within subroutines. It appeared that the stack was overflowing because of 8 nested subroutine calls. To fix this issue, we implemented macros instead of subroutines when appropriate.

In general, we used macros a fair amount. This is because we needed to execute certain functionalities quickly while also passing inputs and outputs without having to move values around. (See the macros at the beginning of the source code)

References

[1] Rudy Wieser, *16-bit Subtraction with Borrow*, PICList, viewed 21 May 2021, <http://www.piclist.com/techref/microchip/math/sub/16bb.htm>

[2] *Double Dabble*, Wikipedia, viewed 16 May 2021, <https://en.wikipedia.org/wiki/Double_dabble>