Embedded System Lab
Fall 2020\2021
Project: Temperature Control System in Datacenter

Omar Mesheh 0174034 - Tuesday
Abdalrahman Al-Tayeh 0175737 - Tuesday
Ahmad Salameh 0175330 - Wednesday
Omar Al-Alsubaihi 0172028 - Wednesday

Introduction

This project is a simulation of a server room with two fans that are used to control the temperature inside a data center room by adjusting the speed of two fans according to the room temperature.

Our main objective is to use the Microcontroller to control the speed of the 2 DC fans either automatically using the current temperature or manually using a control knob, we also want to display the mode we are currently using and the temperature with its corresponding fan speed using an LCD screen.

System Description

The system is composed of three main parts that are interconnected together to achieve the various functions of the systems. The first of these parts is the A/D converter which interprets the voltages coming from the temperature sensor and the potentiometer as 10-bit value number that can be used to determine the required speed of the fans. The second part is the Pulse Width Modulation which has the purpose of interacting with the motors and controlling the average voltage on them which in turn controls the speed of the fans depending on the value of a register that stores the desired speed. The last part is the LCD display which shows the mode the system is operating in and the speed of the fans and corresponding temperature by reading the mode and speed from GPRs in the memory of the micro-controller.

Figure 1 shows a general flow chart of the system and how it behaves in the different main situations. We will discuss each part in details after the flowchart.

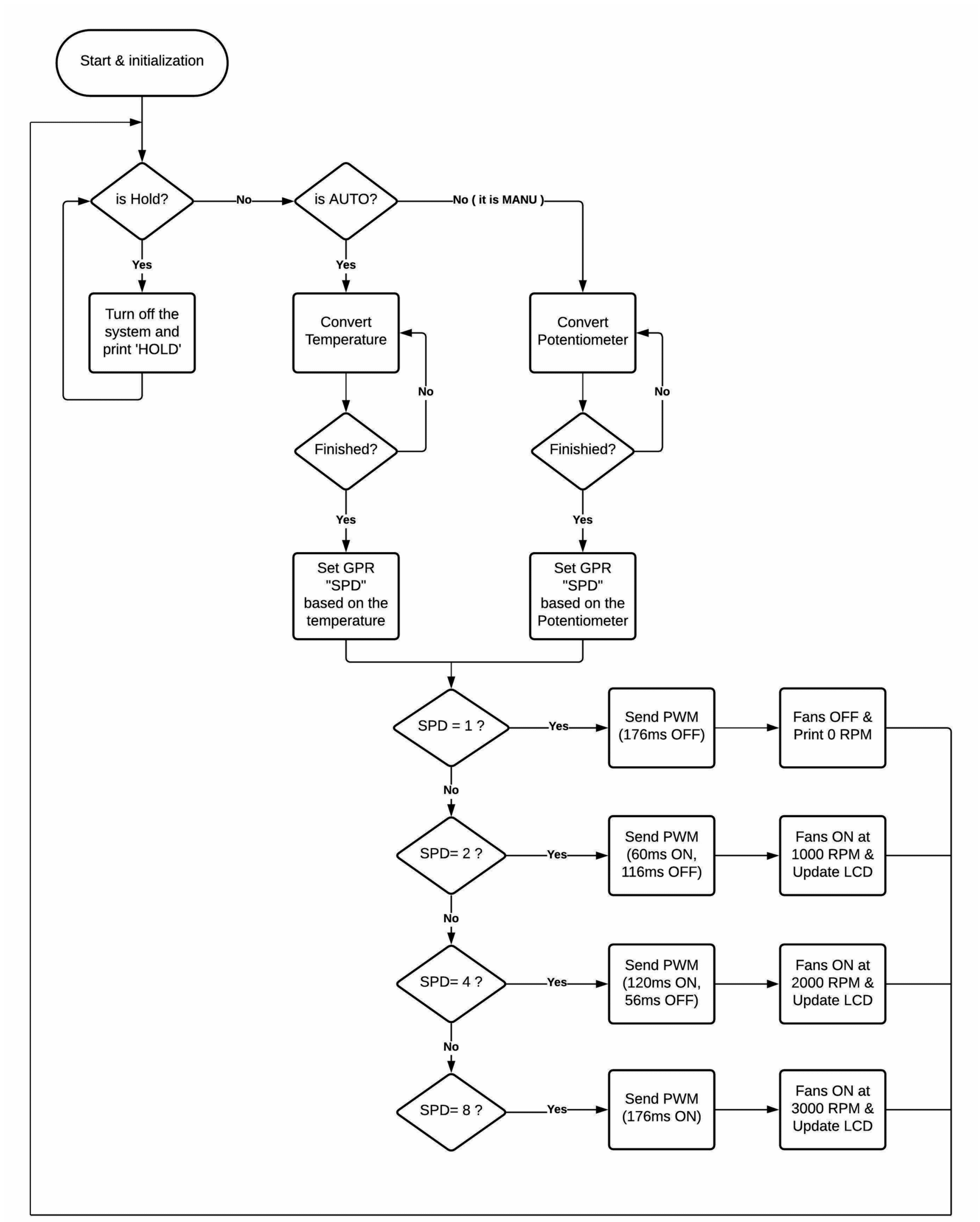
Start & initialization
is Hold?
No
is AUTO?
No (it is MANU)
Yes
Turn off the system and print 'HOLD'
Yes
Convert Temperature
Convert Potentiometer
No
No
Finished?
Finishied?
Yes
Yes
Set GPR "SPD" based on the temperature
Set GPR "SPD" based on the Potentiometer
SPD = 1 ?
Yes
Send PWM (176ms OFF)
Fans OFF & Print 0 RPM
No
SPD= 2 ?
Yes
Send PWM (60ms ON, 116ms OFF)
Fans ON at 1000 RPM & Update LCD
No
SPD= 4 ?
Yes
Send PWM (120ms ON, 56ms OFF)
Fans ON at 2000 RPM & Update LCD
No
SPD= 8 ?
Yes
Send PWM (176ms ON)
Fans ON at 3000 RPM & Update LCD

Figure 1: a Flow chart that shows the general execution stages of the system.

A. Analog to Digital Converter (ADC)

The objective of the ADC is to basically convert the voltage coming from either the temperature sensor coming from RA5 or the potentiometer coming from RE0 to a 10-bit binary value and store it in the two registers ADRESH and ADRESL.

We designed the input voltage to the ADC to be between 1.5V and 2V by choosing Vs to be 5V and R1 as 1.8kΩ and that is why we chose Vref- and Vref+ to be 1.5 and 2V. We decided to store the data left justified (8 most significant bits on ADRESH and 2 least significant bits on ADRESL) because we noticed that a 1°C change in temperature or a 1% change in the potentiometer changes the result of the conversion by a value greater than 3 (the maximum value that the least 2 significant bits can hold), so this way we can limit our work with ADRESH only and neglect the ADRESL register.

Now, To figure out what speed of the fans the value of the conversion corresponds to, we constructed the following table:

Temperature (°C)	ADRESH value
9	0x4C
10	0x50
19	0x7A
20	0x7F
29	0xA9
30	0xAE

To find the values of the ADRESH register for each temperature, we displayed the value stored in ADRESH on PORTD and recorded the results in the table above.

Figure 2 shows a flowchart that describes the work of the subroutine that given the value of ADRESH and returns the speed the fans should work on if the system is operating in AUTO mode:

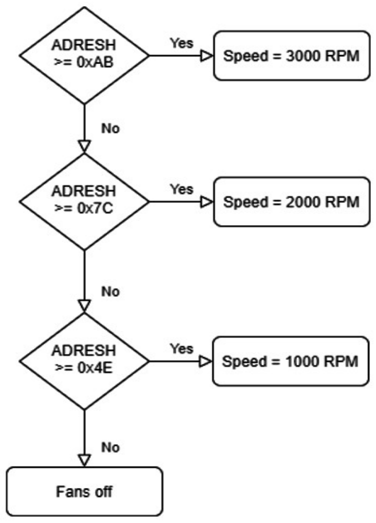

Figure 2: a flowchart of the speed conversion process in AUTO mode.

The values chosen to be compared with ADRESH are the midpoints of the values corresponding to 29°C and 30°C, 19°C and 20°C, 9°C and 10°C. And to figure if ADRESH is greater than a specific value, you just take the 2's complement of that value, add it to ADRESH, and check the carry flag.

The same approach was taken if the system is operating in MANU mode, but the values to be compared with ADRESH were different of course, so that is why the program has to check what mode the system is operating in, and then choose which subroutine to execute.

The following are the table constructed mapping different positions of the potentiometer to a value of ADRESH, and the flowchart in Figure 3 describing the work of the subroutine executed when the system is operating in MANU mode.

Position of the potentiometer (%)	ADRESH value
0	0x00
25	0x40
26	0x42
50	0x7F
51	0x82

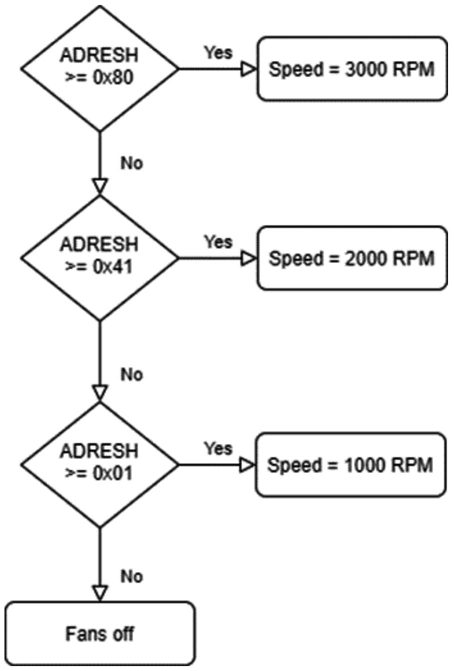

Figure 3: a flowchart of the speed conversion process in MANU mode.

B. Pulse width modulation (PWM)

Our objective is to control the rotating speed of the motors attached to the microcontroller. This can be done using Pulse Width Modulation (PWM) which means to output the value 1 for a certain period of time then output 0 for the another period, this will control the average voltage applied to the motors according to the equation:

$$V_{avg} = \frac{1}{T} * \int_0^T V_s(t)\, dt$$

By having V_s as a constant DC value=220V we can integrate the equation to obtain:

$$V_{avg} = \frac{t_{on} * V_s}{T} = \frac{220\, t_{on}}{T}$$

We will be using both timer0 and timer2 to do the PWM, we will use the timer0 interrupt to output the value 0, and timer2 interrupt to output the value 1. We will also be using the value 176ms as our period (T).

The program starts off by setting the Port C as output and sets the delay for timer0 and timer2 to 4ms according to the following equations:

TMR0:

$$\text{TMR0 Delay} = \frac{4}{\text{Fosc}} * \text{Prescale} * X$$

We set the F_{osc} value to 1MHz and the Prescale value to 8

$$\rightarrow X = \frac{4\,\text{ms} * 1\,\text{MHz}}{4 * 8} = 125\,, \quad X = 265\text{-TMR0} \quad \rightarrow \quad \text{TMR0} = 131$$

But since we have a lot of commands in our ISR, setting TMR0 value to 131 will result in a higher delay than what we want so we set it to 134 to get more accurate results, we set up the TMR0 value and Prescale using the SETTMR0 subroutine

TMR2:

$$\text{TMR2 Delay} = \frac{4}{F_{osc}} * \text{Prescale} * \text{Postscale} * (PR2 + 1)$$

By setting F_{osc} value to 1MHz, the Prescale value to 1 and the Postscale value to 8 we are able to obtain PR2

$$PR2 = \frac{4\,ms * 1\,MHz}{1 * 8} - 1 = 124$$

And since TMR2 is less affected by the multiple lines in our program, we will keep the value of PR2 as it is.

We repeat each interrupt depending on how long our PWM is using TMR0VAR and TMR2VAR which are the counters for their respective timers. We first activate TMR0 and turn off the TMR2 enable for 1 cycle(4ms) so we can check on what our current speed is using in the SPD GPR, In the ISR there are 3 options for our interrupt, either an external interrupt which takes priority over our timers interrupts or a TMR0 or a TMR2 interrupt, if a TMR0 interrupt occurs it means that we are done with our zero output and need to switch to our 1 output, to do that we need to disable the TMR0 interrupt and enable the TMR2 interrupt, and if we got a TMR2 interrupt then TMR2 interrupt is switched off and TMR0 interrupt is enabled and our output changes from 1 to 0, after that we check our speed which will be stored in GPR SPD as follows:

1. **SPD= 1**, in this case our motors have to be off, for this to happen we need to have V_{avg} equal to 0V, by substituting T=176 in the equation, $V_{avg} = \frac{220 * t_{on}}{T} = 0$ we obtain $t_{on} = 0$ which is the only case where we don't switch from a 0 output to a 1 output, in this case we will disable the TMR0 interrupt and re-enable our TMR0 interrupt, we then set the value of TMR0VAR to 44 to obtain our full period(T) (44*4=176) as in Figure 4

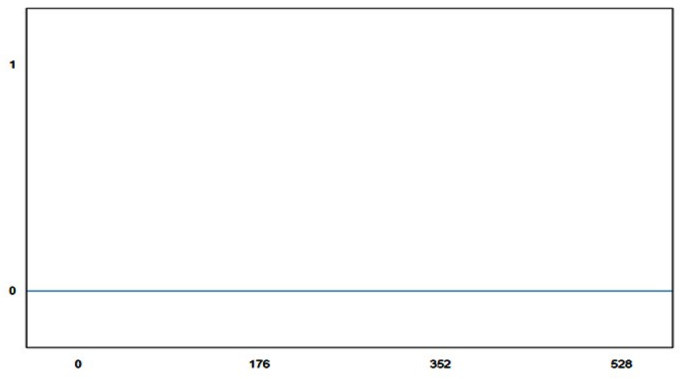

Figure 4: Output=0 always

2. **SPD= 2,** in this case our output should have a rotation speed of 1000RPM, to obtain that speed we need to have V_{avg} equal to 75V, by substituting T=176 in the equation, $V_{avg}=\frac{220*t_{on}}{T}=75$ we obtain $t_{on}=60$ and since each interrupt is 4ms long then we need to set TMR2VAR to 15 to get the 60ms (15*4=60), after the 60ms is done TMR2 will be disabled and TMR0 will be enabled for 116ms to continue the 176ms cycle, to obtain the 116ms delay we need to set TMR0VAR to 29 (29*4=116), as in Figure 5.

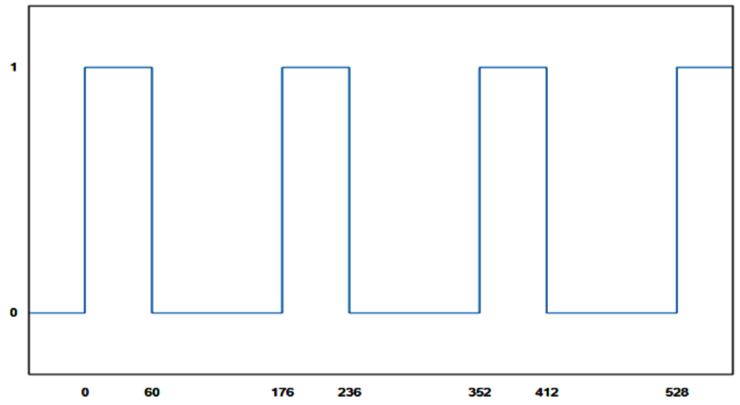

Figure 5: Output=1 for 60ms and 0 for 116ms and the total period=176ms

3. **SPD= 4**, in this case our output should have a rotation speed of 2000RPM, to obtain that speed we need to have V_{avg} equal to 150V, by substituting T=176 in the equation $V_{avg}=\frac{220*t_{on}}{T}=150$ we obtain $t_{on}=120$ and since each interrupt is 4ms long we have to set TMR2VAR to 30 (30*4=120), after the 120ms is done TMR2 will be disabled and TMR0 will be enabled for 56ms to continue the 176ms cycle, to obtain a 56ms delay we need to set TMR0VAR to 14 (14*4=56), as in Figure 6.

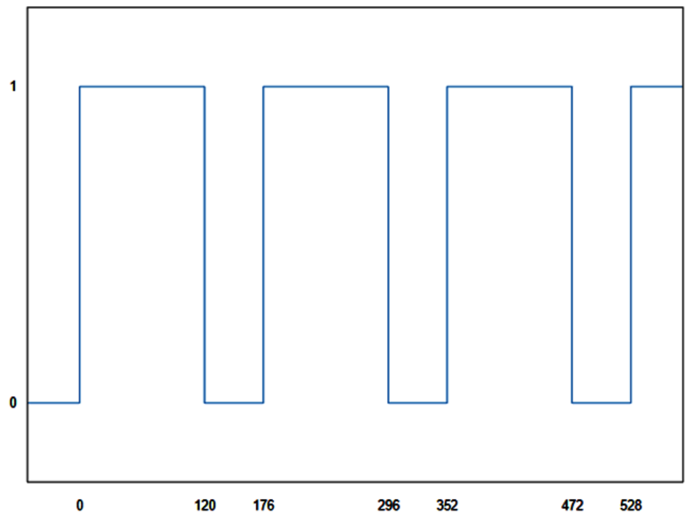

Figure 6: Output=1 for 120ms and 0 for 56ms and the total period=176ms

4. **SPD= 8**, in this case our motors should have a rotation speed of 3000RPM, to obtain that speed we need to have V_{avg} equal to 220V , by substituting T=176 in the equation, $V_{avg}=\frac{220*t_{on}}{T}=220$ we obtain $t_{on}=220$which means that we have to keep our output 1 for the whole 176ms period, to do that we will have to set TMR2VAR to 44(44*4=176) after the 176ms are done TMR2 will be disabled and TMR0 enabled but for 3000RPM we have to have our output=1 at all times so we will have to re-enable TMR2 and disable TMR0 and set TMR2VAR to 44 again, as in Figure 7

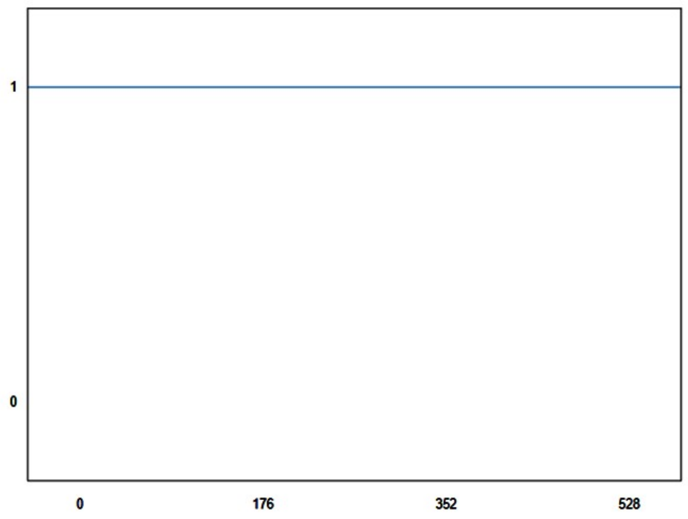

Figure 7: Output=1 always

C. LCD Part

The LCD part is one of the main indicators that tell us how the system behaves in different situations. The LCD data line is connected to the PIC through PORTD. The Rs, Rw and E pins are connected via RB1, RB2 and RB3, respectively.

In the initialization stage, two commands are sent to the LCD to initialize it as follows:

- **0x0C**: To turn ON the LCD, with cursor underline and blinking settings set to OFF.
- **0x38**: To set the LCD in 8-bit mode, 2 lines display and 5x7 dot format.

If the state of the system is set to **hold**, the entire system will turn off except the LCD such that it will display **"System in Hold"** with no speed and temperature. The least significant bit in a general-purpose register called **"**wasHOLD**"** will be set to 1 to indicate that the system was in the hold state. We will use this bit in the next system iteration to check if the previous state of the system was hold AND we are currently in AUTO or MANU modes, a clear command will be sent to the LCD. the flow chart in Figure 8 describes this process:

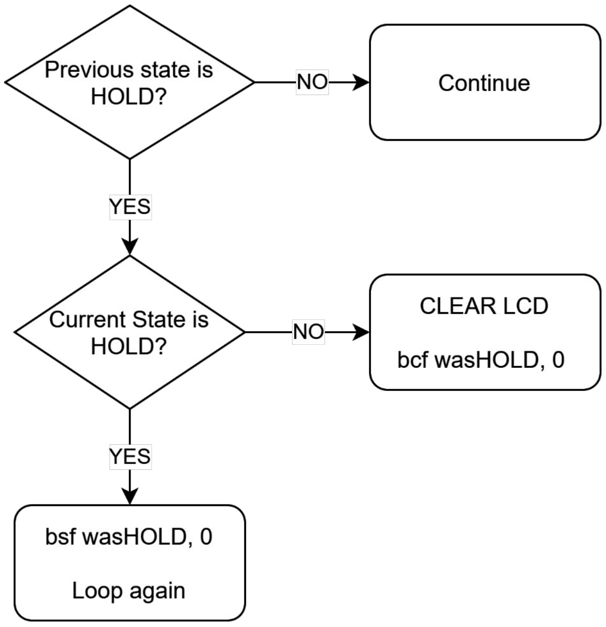

Figure 8: a Flow chart diagram of the LCD clearing process.

On the other hand, if we are switching between AUTO and MANU modes and vice-versa, there is no need to clear the LCD since we have initialized it such that it will display 'AUTO' and 'MANU' in the same LCD positions, so the current mode will override the previous one.

If the current state of the system is the AUTO mode, The LCD will display “**AUTO**“ in the first line followed by the fans speed and temperature in the second line. The same thing is done if the system was in MANU mode, except that the LCD will display “**MANU**“ in the first line.

Now, to tell the LCD what speed and temperature it should display, the general-purpose register “**SPD**“ is used to store the fans speed based on the A/D conversion result, this register could have one of the following values:

GPR SPD Value	Fans Speed	Temperature
1	0000 RPM	00 °C
2	1000 RPM	10 °C
4	2000 RPM	20 °C
8	3000 RPM	30 °C

Note that the only changing number is the most significant digit of the speed and temperature values, so we can basically check the SPD register and display the thousands digit of the speed and the tens digit of the temperature, then fill the remaining digits by zeros. the following piece of code explain this process:

```
;To display the appropriate speed:
      btfsc   SPD,0
      movlw   '0'
      btfsc   SPD,1
      movlw   '1'
      btfsc   SPD,2
      movlw   '2'
      btfsc   SPD,3
      movlw   '3'
      Call    send_char
      movlw   '0'
      Call    send_char
      movlw   '0'
      Call    send_char
      movlw   '0'
      Call    send_char
      movlw   'R'
      Call    send_char
      movlw   'P'
      Call    send_char
      movlw   'M'
      Call    send_char
```

```
;To display the appropriate temperature:
      btfsc   SPEED,0
      movlw   '0'
      btfsc   SPEED,1
      movlw   '1'
      btfsc   SPEED,2
      movlw   '2'
      btfsc   SPEED,3
      movlw   '3'
      Call    send_char

      movlw   '0'
      Call    send_char
      movlw   'C'
      Call    send_char
      movlw   0xdf    ; the (°) symbol
      Call    send_char
```

LCD Delay:

To let the LCD process the commands it receives, a software-generated delay is used in the ‘send_cmd‘ subroutine such that it will loop for 12820 cycles (or 51.8 mSec) with a PIC frequency of 1MHz.

```
LCD_delay
        movlw   0x10
        movwf   msd
        clrf    lsd
loop2
        decfsz  lsd,f
        goto    loop2
        decfsz  msd,f
endLcd
        goto    loop2
return
```

Also, to make the LCD immediately update any written sentences, we have removed this delay in the ‘send_char‘ subroutine so now any changes in the mode or speed will be immediately displayed with no delay between the characters.

Hardware System

Figure 9 shows the final circuit that is ready to use.

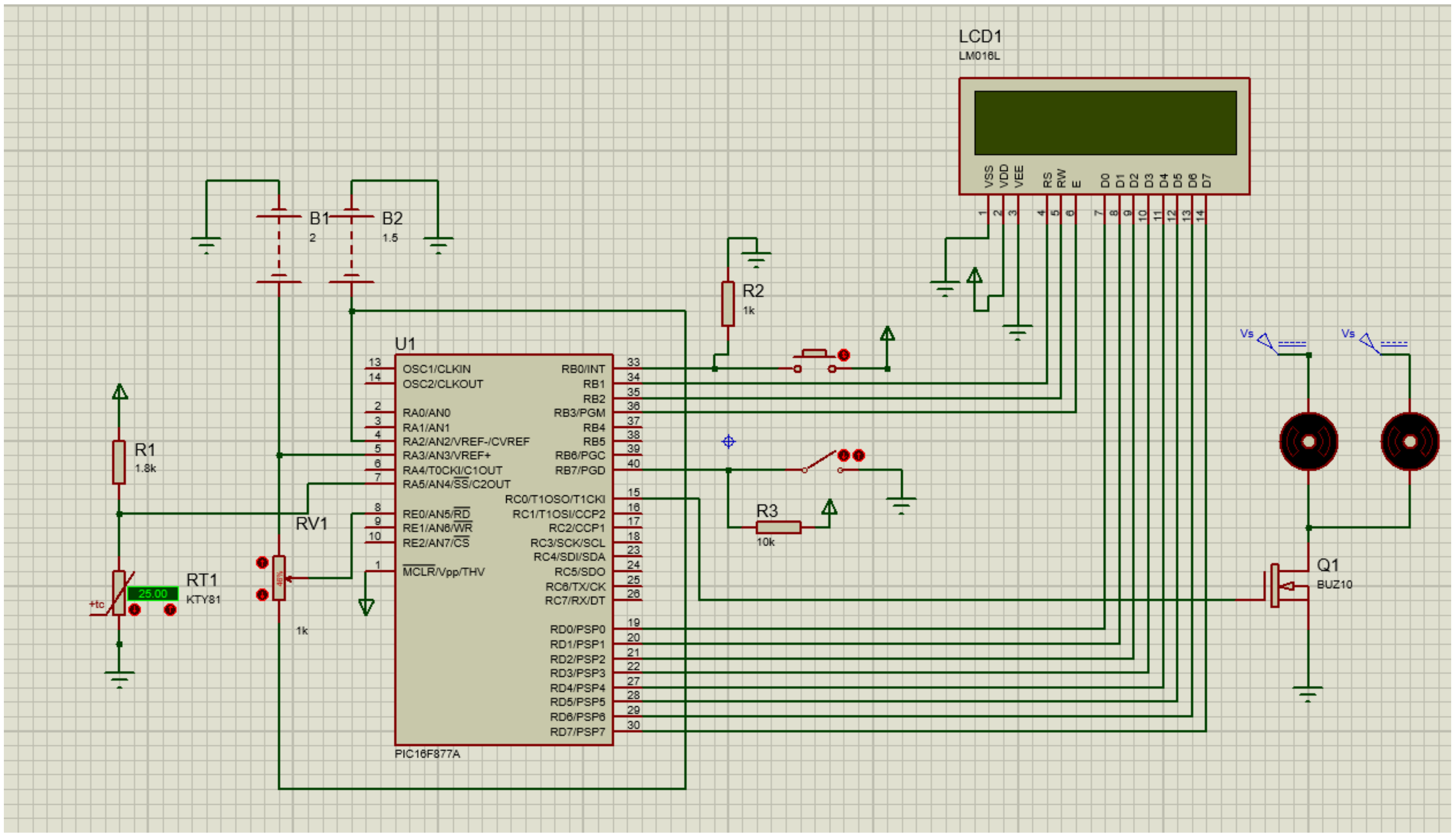

Figure 9: The full and final circuit of the system

This circuit can be divided into small parts as follows:

- **Push-button:**
  The purpose of the push-button is to change the mode of operation from AUTO to MANU or vice versa. It is connected to RB0 using a pull-down resistor such that a rising edge occurs on the pin when the button is pressed. And since the external interrupt is enabled, an interrupt occurs when the push-button is pressed and the ISR changes the mode of operation by changing the channel of the A/D converter from RA5 (Temperature sensor) to RE0 (Potentiometer) or vice versa.

- **Hold switch:**
  The hold switch should put the system in the hold state (Fans off) when closed no matter what the temperature or the position of the potentiometer is. It is connected to RB7 via a pull-up resistor, so that it gives logic 0 when closed, and that is why the program has to check on RB7 every time it returns to the program loop.

- **KTY81 sensor:**
  The idea of this sensor is that it changes resistance with change in temperature, and so we can connect it in series with another resistor and use the principle of voltage division to find the voltage on the sensor. Then the voltage on the sensor is taken as the input to the A/D converter, and since each temperature corresponds to a specific voltage the PIC can figure what the temperature is.

- **Potentiometer:**
  The potentiometer also uses the principle of voltage division to change the voltage input to the A/D converter depending on the position of the knob. This way we can program the PIC to interpret each position of the knob as a different speed.

- **LCD:**
  The LCD displays the mode of operation of the system, the speed the fans are working on, and the corresponding temperature if the system is not in the hold state. And displays ‘System in hold’ when in hold state. This was achieved by having all the pins in port D as the input data lines to the LCD, and pins RB1, RB2 and RB3 as the controls lines.

- **DC Motor and Buz10 MOSFET transistor:**
  The fans are modelled as 220V DC motors, and because the PIC doesn’t provide this kind of high voltage, a transistor is used to interface between them while a 220V DC voltage is connected to the motors. To control the speed of the motors, PWM is used to alter the average voltage on the motors.

System Testing and Results

To test the system, we will use the following test cases and see the results of the speed and temperature. We already seen the results of the fans in the PWM section, so we will focus on the LCD results. You can click on the 'Figure #' in the result column to see the result of each case directly.

Case	Mode	Temperature Sensor	Potentiometer	Speed	Result
1	AUTO	0	28% (don't care)	0000 RPM	Figure 11
2	AUTO	11 °C	28% (don't care)	1000 RPM	Figure 13
3	MANU	16 °C (don't care)	0%	0000 RPM	Figure 15
4	MANU	16 °C (don't care)	17%	1000 RPM	Figure 17
5	HOLD	don't care	don't care	0000 RPM	Figure 19
6	AUTO	28 °C	8% (don't care)	2000 RPM	Figure 21
7	AUTO	43 °C	8% (don't care)	3000 RPM	Figure 23
8	MANU	13 °C (don't care)	38%	2000 RPM	Figure 25
9	MANU	13 °C (don't care)	67%	3000 RPM	Figure 27

We will first turn the system ON, we will find out that the system starts in the AUTO mode as supposed to be. If we put a zero on the temperature sensor, the fans will stop and the LCD will display the result as in Figure 11.

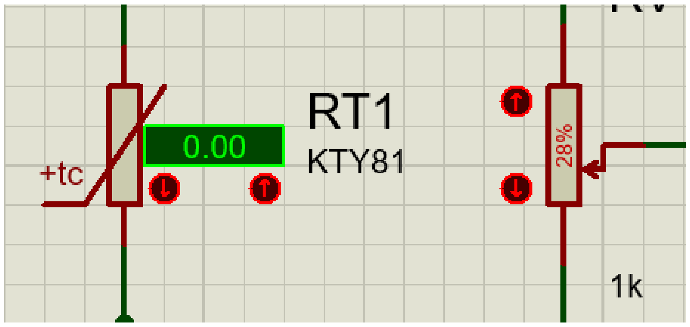

Figure 10: Case 1, AUTO mode with 0 °C.

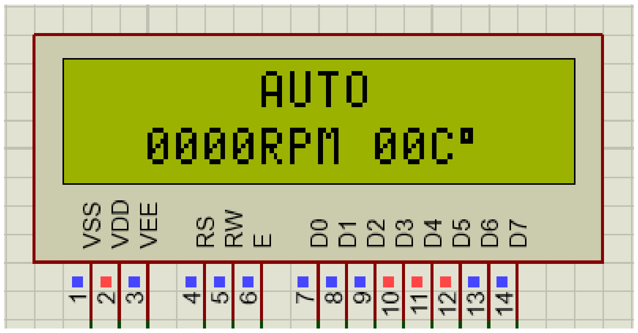

Figure 11: Case 1, AUTO mode with 0 °C result.

If we keep the mode on AUTO and raise the temperature to 11 °C, we will get the following result:

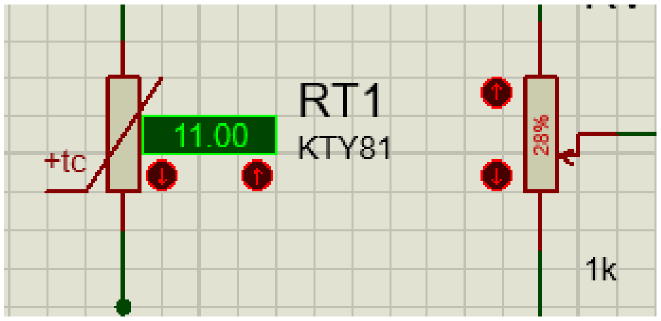

Figure 12: Case 2, AUTO mode with 11 °C.

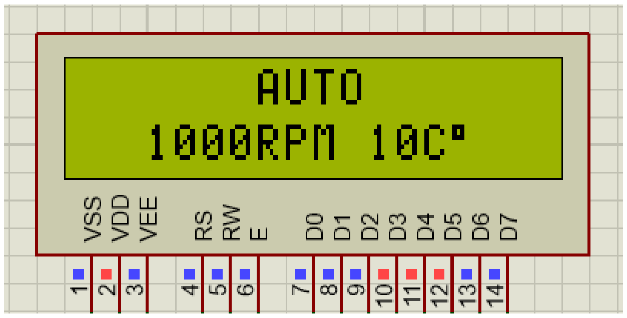
AUTO
1000RPM 10C°

Figure 13: Case 2, AUTO mode with 11 °C result.

Note that in both previous cases the potentiometer has no effect on the result, since we are reading from the channel associated with the temperature sensor.

Now, we will press the mode push-button and set the potentiometer reading to 0%, the result is shown in Figure 15.

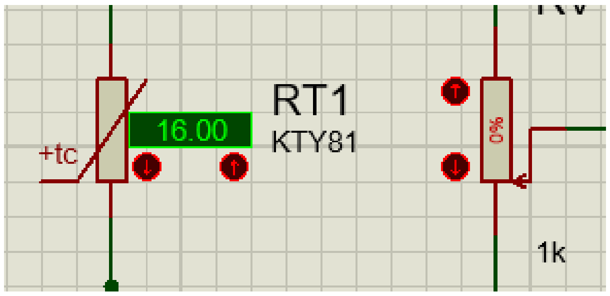

Figure 14: Case 3, MANU mode with 0%.

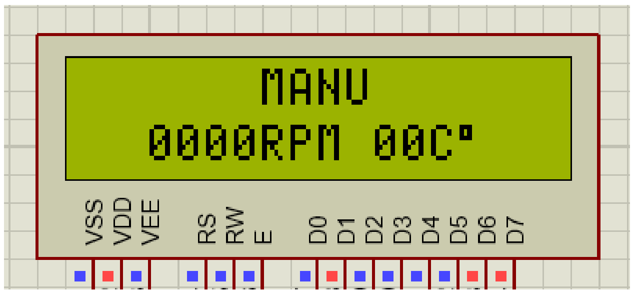
MANU
0000RPM 00C°

Figure 15: Case 3, MANU mode with 0% result.

If we keep the mode on MANU and raise the value of the potentiometer to 17%, we will see the result in Figure 17.

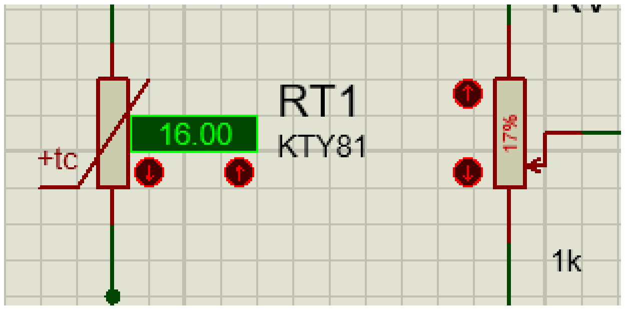

Figure 16: Case 4, MANU mode with 17%.

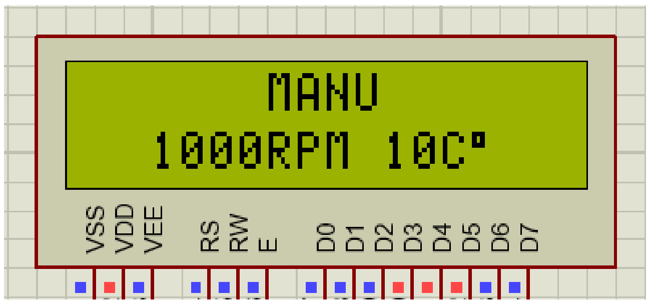
MANU
1000RPM 10C°

Figure 17: Case 4, MANU mode with 17% result.

Again, note that the temperature sensor readings don‘t affect the result while we are running in the MANU mode.

Let‘s close the switch associated with the hold state and see the results in Figure 19.

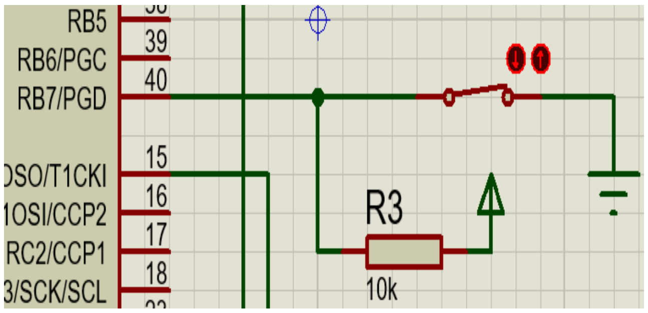

Figure 18: Case 5, Switch is closed.

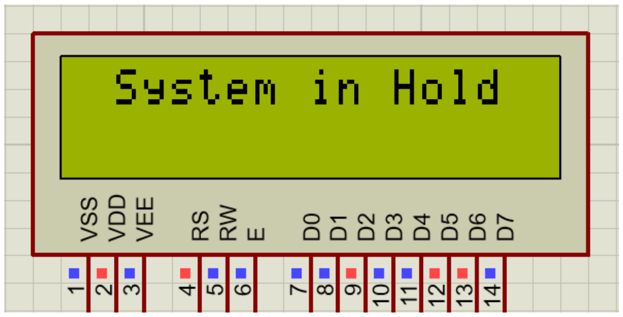

Figure 19: Case 5, Hold state result.

Now, if we press the mode button while we are in the hold state, the system will automatically switch between modes; this is because the mode button is represented as the external interrupt of the micro-controller. So, whenever we press the button, the mode will be switched.

After pressing the button and releasing the switch, we will be in the AUTO mode, if we raise the temperature to 28 °C, we will see the result in Figure 21.

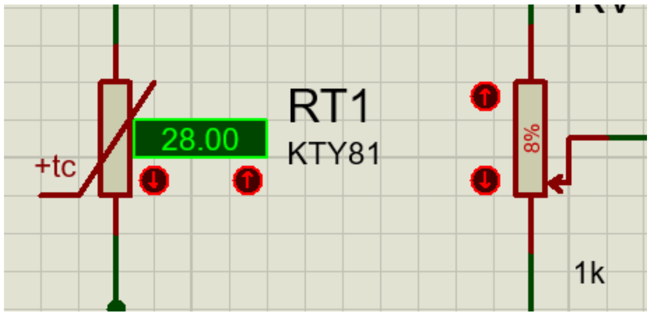

Figure 20: Case 6, AUTO mode with 28 °C.

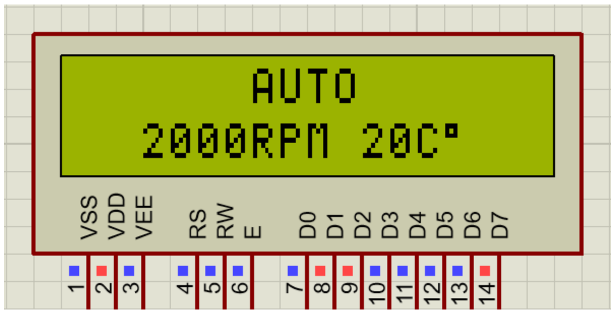

Figure 21: Case 6, AUTO mode with 28 °C result.

The last case in the AUTO mode is a temperature reading higher than 30 °C, we will try it with 43 °C.

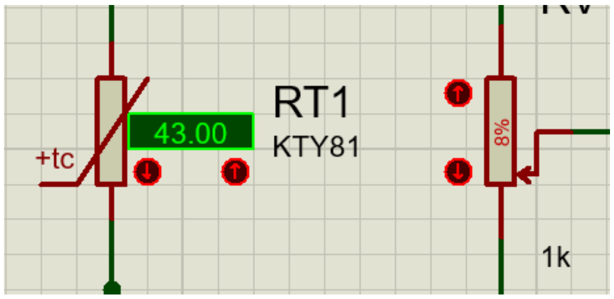

Figure 22: Case 7, AUTO mode with 43 °C.

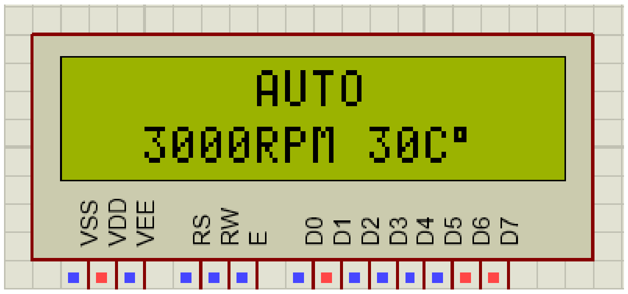

Figure 23: Case 7, AUTO mode with 43 °C result.

In the MANU mode, we still have two more cases to test. If the potentiometer is set to 38%, we expect a speed of 2000RPM, as shown in Figure 25.

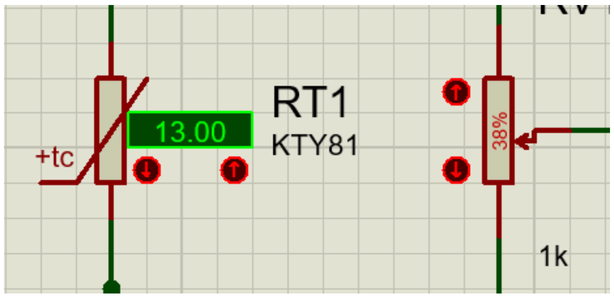

Figure 24: *Case 8, MANU mode with 38%*.

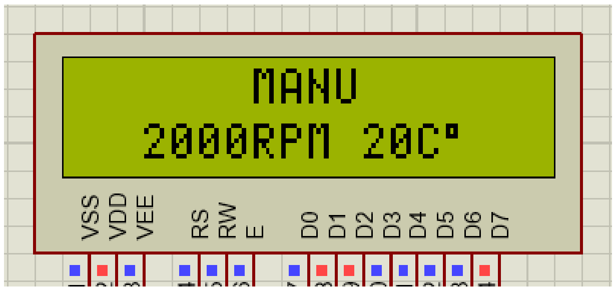

Figure 25: Case 4, MANU mode with 17% result

and the last test case occurs at a potentiometer reading higher than 50%, we will test the system at 67%.

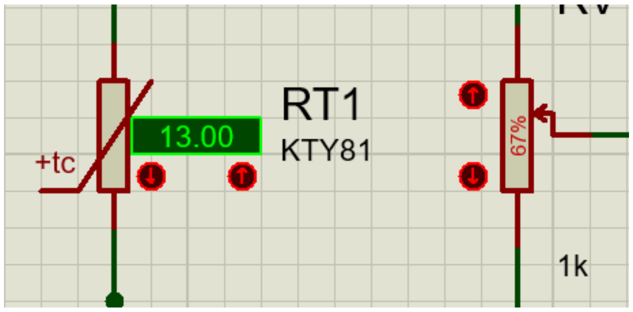

Figure 26: Case 9, MANU mode with 67%

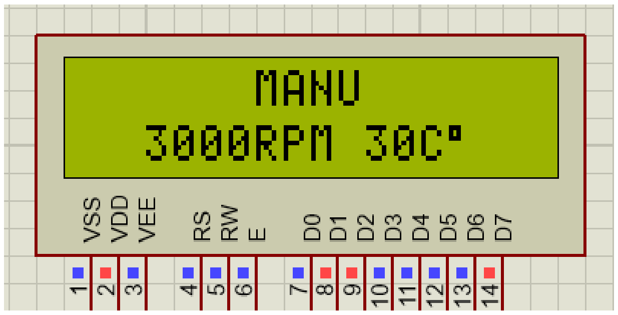

Figure 27: Case 9, MANU mode with 67% result.

Conclusion

In this project, we were asked to build a simple control system that controls the temperature of a room using two fans either automatically using a temperature sensor or manually using a control knob.

We first receive a signal from either the temperature sensor or a potentiometer which we used as our control knob, these signals are analog in nature so we need to convert them into digital signals using the PIC16F877A analog to digital converter module, the digital signal is then converted into one of four speeds depending on the value of that signal, our program takes that value and the mode we are operating on and displays them on the LCD screen, that value also controls the speed of the fan motors using the pulse width modulation(PWM) technique, for a speed of 0 RPM our program output will be 0 for the whole period, For a speed of 1000RPM our program output will be 1 for 60ms and 0 for 116ms, For a speed of 2000RPM our output will be 1 for 120ms and 0 for 56ms, to obtain the speed of 3000RPM our output will be 1 for the whole period, these values will keep on repeating until a change in the speed value occurs.

When we were merging our program together, we faced one major problem, our timer0 and timer2 interrupt delays were interfering with our LCD delays which affected what‘s displayed on our LCD screen, we were able to solve that issue by changing our frequency from 4MHz to 1MHz, that made the timer interrupts take more time between each interrupt (4ms instead of 1ms) which minimized the interference of the interrupts with the LCD screen‘s output.

	Omar Mesheh	Abdalrahman al-Tayeh	Ahmad Salameh	Omar Alsubaihi
Project	• LCD Part • System integration (Main + LCD + ADC)	• PWM Part • ISR • System integration (Main + ISR)	• ADC Part • Speed equations derivation and coding	• External interrupts • System integration (Proteus)
Report	• System Description - LCD. • System testing and results. • Organizing the final report.	• Introduction • System Description - PWM • Conclusion	• System Description + ADC part • Hardware System	