ELEC2117 Design Report

Design Specification

On startup the LCD is required to display both team members full names for approximately one second each. The LCD must additionally continually update temperature and humidity, temperature and humidity can either be toggled between each other or both displayed simultaneously.

The motor is required to run at 3 different speeds (must be done varying the duty cycle) based on 3 different temperature thresholds, LEDs must be used to indicate which threshold the temperature is currently between (one LED lit per #thresholds below current temperature).

For an advanced implementation the thresholds should be able to be dynamically changed by the user with a “service mode” (in which LEDs and the motor are turned off). The service mode must be able to switch back to “normal mode” and needs to be password protected by one of the team members zID.

Hardware Design

The design of the hardware to meet the requirements centered around the use of the PIC16F886 chipset. As seen in figure 1, there were numerous components connected to the chipset: keypad, LCD display, DHT11 sensor, Motor, and LEDs.

The keypad was connected so that the four pins connected to the rows were connected to the first four pins of port A, and the four pins connected to the columns were connected to port B. This connection was utilised so that the column pins, which are set as input, could be connected to the internal pull-up resistors that are located within port B. The rows and columns of the keypad were separated so that the pins could not be unintentionally modified during their usage.

The DHT11 was connected to port B, pin 4, which was set periodically between an input and output pin, depending on what stage of communication the system is at. 5 Volts DC was applied to the Vcc terminal, and a 10k pull-up resistor is connected to the IO pin of the sensor.

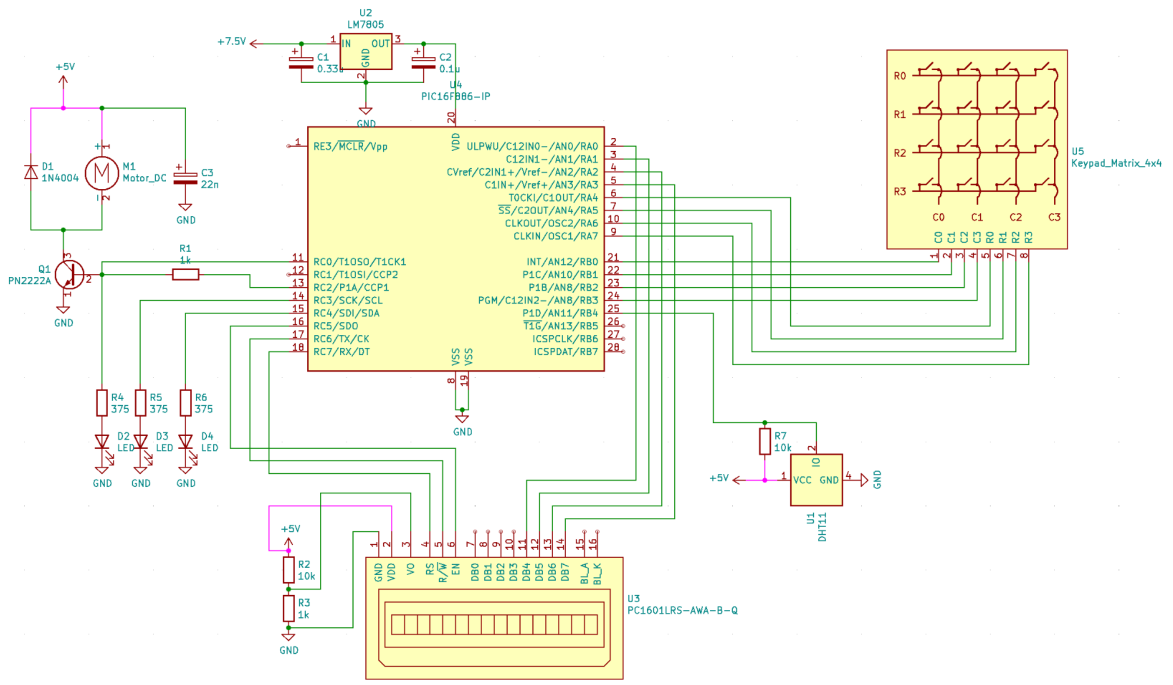
For the LCD display, the four bit mode of the system was used, which meant that a lower amount of pins were required for complete usage of the LCD. the pins DB4 through DB7 were connected to the first four pins of port A, and the RS, R/W, and EN pins were connected to port C, pins 5 through 7. Vcc, Gnd, and VO are connected using a voltage divider with a 10k and 1k resistor.

For the motor, a 1k resistor was connected to a PN2222 BJT, which acted as a switch for the circuit. These were connected to the CCP1 pin of the microcontroller, which is used for a

PWM output. The collector of the transistor was connected to a motor and a 1N4004 rectifier diode in parallel, to prevent back EMF to pass back through the input. The ends of both the motor and diode are connected to a 5 Volt terminal. A capacitor was placed in parallel with the motor and ground to reduce noise.

Three LEDs were connected to port C, pins 1, 3, and 4, each in series with a 375 Ohm resistor, which was chosen to ensure that the maximum current input of the LED was not surpassed, so that the LED is bright enough.

Lastly, a voltage regulator was connected in the recommended configuration with two polarised capacitors, to supply 5 Volts to both the PIC16F886, as well as the 5 Volt terminals that are throughout the system. As the required current of each element is quite low, only one voltage regulator was required for the overall system.



Software Design

LCD

As stated in the design specification the LCD has an important role of displaying required information. This includes displaying both teammates names on startup and continually displaying and updating temperature and humidity, our team decided to toggle between the two modes for a more elegant implementation.

Before the LCD can be used it must be set up correctly, our code utilises a loop called LCD_init and a lookup table called LCD_init_lookup. First the LCD is put into instruction mode then LCD_init_lookup sets the LCD to 8-bit mode, sending the instruction 3 times to make sure the LCD ends up at the start of an instruction cycle. It is then set to 4-bit mode, display mode is set (interface data is set to 8 bits, number of lines is set to 2 and font size is

5x8), display enabled, cursor and blink disabled, set entry mode (DDRAM address decreases by 1) then the LCD is cleared.

Writing characters to the LCD is done using the lcd_lookup table and the lcd_instrcution function. The lcd_lookup table simply contains the different 8-bit instructions for the different characters while the LCD is set to write to its data register. lcd_instruction function performs the operation cycle needed to send the 8-bit instruction to the LCD in 4-bit mode, the cycle is given by:

set PORTA pins RA0-RA3 to the first 4 bits of the instruction -> set enable pin to HIGH to load data -> wait 100us -> set enable pin to LOW to submit data -> wait 2ms for instruction to execute.

This function takes advantage of another function called lcd_split which stores the first 4 and last 4 bits of the byte stored in the working register into the bottom 4 bits of instruction1 and instruction2 respectively. This allows an 8-bit instruction to be loaded into the working register and executed in 4-bit mode by the lcd_instruction function. Therefore the process of sending an instruction to the LCD is very simple. Firstly, move the 8-bit instruction you want to execute into the working register, lcd_lookup can be used if the instruction is a character then call_lcd instruction.

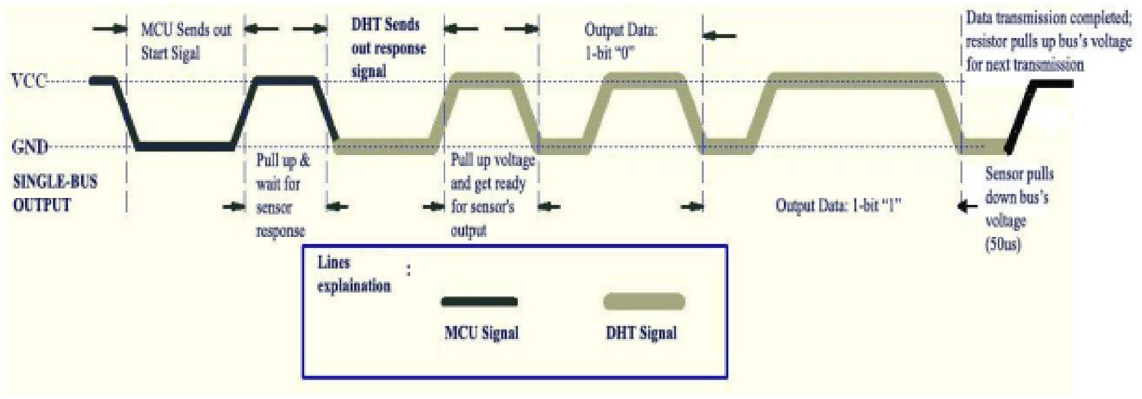
The method that was just discussed is used to print out ‘FLYNN CAWOOD’ and ‘RILEY DEAN’ although there are a few additional steps. Firstly, the first 8 digits of the LCD are at a different memory location than the second 8 digits, this requires the DDRAM address be changed to the address of the second 8 characters. This is done by the new_line function after 8 characters are written to the LCD, then the remaining characters can be printed. Once the full name has been displayed by the LCD for one second the display must be cleared, this can be done with the reset_display function.

For displaying and continually updating either the temperature or humidity the display_info function is used. The display_info function goes to either the display_temp or display_humid functions depending on if the toggle is set or not. The display_temp and display_humid functions work very similarly to each other, the only differences being the memory location they use (temperature1 or humidity1) and the non-integer characters printed (TEMP ##· or HMD ##%). Since the calculate_info function places the integral value of the temperature and humidity into memory locations temperature1 and humidity1 as binary values, all that needs to be done is to separate the binary value into two decimal integers.

The lcd_lookup table and lcd_instruction function can then be used to display the decimal values as the lookup table decimals are in order (e.g. a value of 3 in the working register finds the instruction to print 3). The process to convert the binary values into 2 decimal integers follows this procedure. For the first digit subtract 10 from the value until the value is negative, increment digitcount every time 10 is subtracted, digitcount will contain the value of the first digit. For the second digit subtract 10 digitcount times from the value, this should leave you with the lower digit, the two digits can then be printed independently.

DHT11

The DHT11 uses a form of serial communication which sends a series of 40 bits, which are encoded by a logic high of uniform length, and a logic low of varying length, depending on whether a 1 or a 0 is to be passed through, as seen in the diagram below.


MCU Sends out Start Sigal
DHT Sends out response signal
Output Data: 1-bit "0"
Data transmission completed; resistor pulls up bus's voltage for next transmission
VCC
GND
SINGLE-BUS OUTPUT
Pull up & wait for sensor response
Pull up voltage and get ready for sensor's output
Output Data: 1-bit "1"
Sensor pulls down bus's voltage (50us)
Lines explaination :
MCU Signal
DHT Signal

In order to start communication with the DHT11, a logic low has to be sent through the circuit for at least 100 us, in order for the sensor to know to begin communication, once this time has passed, a logic high will be passed through, and then the DHT11 will start to transmit the data signals. Initially, TRISB bit 5 is set to input, so the DHT11 can be read, however to send the initialisation signal above, TRISB port 5 is set to output, with PORTB being set by clearing bit 5, and then calling a delay for 100us. Afterwards, PORTB bit 5 is set, and TRISB bit 5 is turned to input. At this stage, the PIC will wait for a logic high, and then a logic low, by consistently checking the value of PORTB bit 5. Once these two loops are past, the main algorithm begins.

Each set of 8 bits is used for a different function: The first two are the integer and decimal part of the humidity, followed by two bits which are the integer and decimal part of the temperature, and then finally a parity bit, which is the logical OR of each of the other sets of 8 bits, which can be used to ensure that the algorithm works correctly. To read each bit, a register for each was created, and a multiplexing bit mask was created, which is initially set to the binary value 10000000. Then, a one loop and a zero loop were completed to make sure that the PIC was reading the correct bit. As the length of a 0 bit was 26-28us, a delay loop of 40us was called, and then the value of PORTB bit 5 was measured. It was noted that this value was always equal to the value the DHT11 was attempting to output, and hence it was saved directly to the required register using the multiplex bit mask above. Each time a bit was registered, the multiplexer was shifted to the right, and once it got through the last bit, the next register was used to read data, until all data was read.

Sometimes the DHT11 initialisation can fail, causing the sensor to stay at a logic high, which would break the above algorithm. To ensure this is not an issue, a counter loop was utilised, to make sure that, if no zero is found for a certain period of time, then the loop will be broken, and the sensor reading will start again. This allowed for a robust solution to reading the data from the DHT11 sensor.

Motor and LED Output

To utilise the motor, a PWM output is produced, with a varying duty cycle to reflect the varying threshold temperature values coded into the program. This is done by using the CCP1 pin of the PIC16F886. To initialise the PWM output, the CCP1CON and T2CON registers are set to initialise both the PWM pin and the timer for the PWM output respectively. By altering the values of the CCPR1L and PR2 registers, the duty cycle of the output would change, resulting in different motor speeds.

Each temperature level was stored in their own individual register, and to test for each level, the individual register is subtracted from the temperature calculated previously. If the number is negative, then it will underflow, causing the MSB to become a 1, which is the condition that is tested for. If bit 7 is 1, then that level is active. As each level increases, the value of CCPR1L increases by 10, from 0 to 30, which will cause an increase in the motor speed at each level. Also, at each level from 1 to 3, an LED will turn on, which is done by setting the value of PORTC pins 0, 3, and 4 to 1. It was also noted that to ensure consistency in output, the LEDs that weren't on at each level had to be set to 0 as well.

Keypad

This implementation didn't include the advanced feature of 'service mode', therefore the keypad is only used to toggle between displaying humidity or temperature on the LCD. Our keypad is set up using a walking zero implementation, this means a 0 is right rotated through RA4-RA7 (with carry set 1 so the other RA4-RA7 pins are set to 0). This correlates to different rows of the keypad being set to logic 0 one by one, due to the pull up resistors in RB0-RB3 if a button is pressed while the row is logic 0 the PORTB pin corresponding to the same column will be pulled down to logic 0. This can be used to determine the exact keypad button that has been pressed, but in our implementation this is not required as all keypad buttons toggle the LCD. Our code is set up to wait until any bit in the PORTB register has been set to 0, the value of PORTB and PORTA are saved (in store_port_a and store_port_b) and later bit tested to determine which row and column was pressed. A check_press_state function is used to ensure that if the value of PORTA and PORTB corresponding to the rows and columns of the keypad have not changed since the initial button press (stored in variable press_state) the toggle variable is not changed.

Testing

LCD

To test the LCD we simply observed the output after certain conditions. The LCD was required to print both students names for approximately one second after startup, each student's name was observed and timed to ensure this functionality. The LCD was also required to toggle between temperature and humidity and continually update their values when displayed. The toggle was tested by instructing a lab demo to press any button on the keypad and observing if the LCD toggled modes. To determine if the temperature or humidity was being updated and displayed properly the displayed value was checked against our temperature1 and humidity1 variables in MPLAB.

DHT11

To test the DHT11 sensor, first it was turned on at room temperature, to ensure that the correct temperature was working. To ensure that it was updated correctly, one of the demonstrators would cover the sensor with their finger to heat it up, and this would be used to see how accurately the temperature would increase. It was found that the sensor updated quickly to changes in temperature, and, through the use of the error-detection built into the function, responded well to circumstances that would cause problems, resulting in a consistently working temperature sensor.

Keypad

In order to test the keypad, as with most other sections of the program, lab demonstrator input was required. For testing, the expected output was noted, which was generally that the printed elements on the display would change from temperature to humidity, or vice versa, and then the change in display after a button press was recorded. If the expected output was different to what was recorded, then breakpoints would be added in sections of the code, specifically the function call that is used when a button is pressed. This allowed for comprehensive debugging of the code, as the functionality of the programs could be paused and deconstructed, which allowed for the realisation of multiple errors in the code, for example that the pins corresponding to the rows and columns of the system were flipped, causing the checks for the wrong keys to be passed. Through this testing, it was made clear that the keypad was not responsive enough, and as outlined in the discussion below, numerous methods were used to rectify this.

Motor and LED Output

To test that each of the levels worked correctly, the threshold values were placed close together, at 25, 26, and 27 degrees Celsius respectively. Through this, it allowed a demonstrator to heat up the sensor enough to go through all four levels. It was seen that each LED lit up as required, and the motor, with some fine tuning to the values of CCPR1L and PR2, would increase in speed as the level increased. Then, to ensure that it worked correctly when cooled, the response after the sensor was cooling was noted. When stretching the difference between threshold values, it was found that the sensor would consistently output the required level, with no inconsistencies in LED output or motor speed.

Discussion

Through using the PIC16F886 microcontroller, a system could be designed that reliably measured the temperature and humidity, and outputting these changes onto an LCD, while also modifying the output LED and motor states. The main challenges while creating this system was interfacing with the LCD display, reading the temperature sensor, and working with the keypad.

Due to work in previous labs, initialising the keypad largely required refactoring previous code to make it more intuitive, using look up tables, rather than hard coded values. This led to a large increase in the readability of the code, and made it very easy to debug this code. Because the LCD display was used in 4 bit mode, to simplify the instructions, they were saved as 8 bit instructions in the lookup table, with an algorithm then used to convert them into 4 bit instructions that could be used in the program to communicate with the LCD, which allowed for the code to be much simpler and easier to read. The main challenge throughout development was making sure that the screen was visible throughout the code, as initially the proportion of time the screen was cleared was much greater than when it was printed, which required some restructuring of the main program loop. Overall, the design of the LCD display code was fairly straightforward, however some small challenges were encountered in the process of completing the code for it, which had to be rectified.

The DHT11 was one of the more challenging sections of the assignment, as handling this type of communication was something not encountered before in the course. This meant that a lot of research and preparation was required before writing the code in order to understand how the algorithm used in the program should run. It was noticed that there was a consistent pattern before each bit used in the communication, which is that there was a zero, followed by the required one. This led to the checking for zeros and ones mentioned above, and allowed for the rest of the code to be fairly easily manipulated to ensure that communication worked. Another challenge faced was the length of the delay after a one was first encountered. It was tested having the delay longer, which caused the program to reach the next zero bit, causing errors, as well as with a shorter delay, which sometimes resulted in the one bit still being a high when it was an expected zero bit at the time. Hence, 40us was chosen, and it was found to work consistently to allow for the calculation of the temperature and humidity. The main error that was found, was that sometimes the code would get stuck trying to find a zero before communication even started, suggesting that the initialisation of the sensor was not initialised correctly. This was rectified by utilising a delay loop and a counter inside the checking for zeros loop, and allowed the code to break out of the sensor if it was required to by the program, which allowed for a much more robust program, and also made other sections easier to test, as the system would no longer have to be restarted once it got stuck in the infinite loop as before.

Designing the keypad was the greatest challenge of the project, and also was the largest shortcoming in the program. The feature of the keypad that was to be designed, was to make a button press cause the display to toggle between temperature and humidity, however there were numerous issues in the development of this feature. Firstly, the system was not as responsive as expected, which was caused by multiple reasons. The first was that the keypad was not being polled for a large portion of the time the main function was running. This was because reading the temperature sensor takes up a large portion of time, and so to rectify this, an additional loop was added around the keypad polling, as well as a small delay, which created a much larger portion of the loop that was dedicated to keypad polling, and made the system more responsive. Next, there were bugs related to holding a key down, which was rectified using a debouncing system, where the state of the switch after the previous state is saved, and compared to the new state. If they were different, then the display would change, and this addition made the keypad function a lot more naturally. The last change that had to be made was updating the display after each keypad press.

Originally, the display was only updated after keypad polling was finished. However, this meant that there was occasionally a delay after pressing the keypad until the display would update, which made the system appear less responsive. To fix this, the update display function was called upon each successful key press, which greatly improved responsiveness. While the keypad is still slightly unresponsive, it is much more consistent than in previous implementations. To improve this, potentially another, greater loop could be created, making the proportion of time dedicated to keypad polling could be even larger, perhaps improving the responsiveness of the system.

While there were numerous features implemented in this project, there is still room for improvement. Firstly, as stated above, the keypad needs to be designed to be more responsive in nature, which could require a restructuring of the main loop to include a larger proportion of time dedicated to keypad polling, ensuring that the keypad is checked at what is perceived to be all the time the program is running. The next possible change that could be implemented is the service mode features specified in the design specification, which would use the method of indirect addressing, reading 8 button presses, and then comparing each of those button presses to the values saved into 8 registers, which correspond to the bits of the passcode. If all are equal, then service mode would be entered. The main change possible in service mode would be to change the speed of each motor, by reading in up to three numbers off the keypad, and changing the value of the CCPR1L register to correspond to this number. This functionality was not included due to time constraints, as the functionality of the toggle caused problems in development, which took up a large portion of time. The groundwork for indirect addressing was included in the code, with registers being defined for each bit of the passcode, as well as command entering, however these weren't able to fully fleshed out due to the reason outlined above.

Overall, through this implementation, a robust temperature sensor was created, which could respond to errors and prevent hangs, while still being able to respond to user changes through the toggle command on the keypad. Given the time constraints, it was a feat to produce the implementation given, however this did result in the lack of full functionality through having no service mode.