****

**College of Science and Engineering**

**Department of Electrical and Computer Engineering**

**EE 2361 Final Project**

**Temperature-Controlled Fan**

**Yuan Jiang**

[jian0655@umn.edu](mailto:jian0655@umn.edu)

**Soorya Sundravel**

[sundr030@umn.edu](mailto:sundr030@umn.edu)

**Hongyu Wang**

[wang9529@umn.edu](mailto:wang9529@umn.edu)

**Thomas Skrzek**

[skrze005@umn.edu](mailto:skrze005@umn.edu)

**Introduction**

Modern mechanical and electrical systems require temperature control to ensure that they are functioning properly. Proper airflow and operating temperature help the components maintain their peak working efficiency. This project aims to offer a solution by implementing a fan that turns on automatically at a certain temperature. When the fan is turned on, its speed could be changed manually by the press of a button. This involves reading the data through a temperature sensor and controlling the speed of the fan using a motor driver. The fan turns off and the button is disabled when the temperature falls below a certain threshold. An LCD display is used to report the temperature values.

The libraries used in this project involve setting up the respective devices and reading data from, or writing data to them. Their intended function is to perform the tasks as stated above. There are three libraries created for this project: Temperature Sensor, Motor Driver, and LCD Display.

The temperature sensor library initializes the sensor by establishing I2C protocol, setting baud rate, and writing commands to receive high precision temperature data. The library also implements a function to read data from the device by sending an I2C packet with the sensor slave address and read command.

The motor driver library initializes Output Compare, as PWM signals are used to control the speed of the fan. Input Capture is also set up in this library to detect button presses and toggle between speed settings of the fan. An Input Capture Interrupt is called every time the button is pressed, which in turn calls the setSpeed function to change the speed of the fan using OC1RS.

The LCD library configures the display by sending instruction bytes and provides functionality for printing characters and strings using the I2C protocol. A delay function is also implemented to be used in the initialization.

**Hardware Description**

The PIC24FJ64GA002 microcontroller was used in this project. The microcontroller was interfaced with the SHT40 Humidity and Temperature Sensor, TB6612FNG Motor Driver, and the AQM0802A-RN-GBW LCD Display. A 5V DC source is connected to power the Motor Driver, along with a push button wired to the microcontroller. A DC motor with attached fan blades is connected to the output pins of the Motor Driver. The hardware block diagram and connection of the system are shown below in Fig. 1.

|  |
| --- |
| Figure 1: Hardware Block Diagram and Connection of Temperature-Controlled Fan |

Devices Used:

* PIC24FJ64GA002
  + Overview: <https://www.microchip.com/en-us/product/PIC24FJ64GA002>
  + Datasheet: <https://ww1.microchip.com/downloads/en/devicedoc/39881e.pdf>
* SHT40 Humidity and Temperature Sensor
  + Overview: <https://www.adafruit.com/product/4885>
  + Datasheet: <https://cdn-learn.adafruit.com/assets/assets/000/099/223/original/Sensirion_Humidity_Sensors_SHT4x_Datasheet.pdf?1612388531>
* TB6612FNG Motor Driver
  + Overview: <https://www.sparkfun.com/products/14451>
  + Datasheet: <https://www.sparkfun.com/datasheets/Robotics/TB6612FNG.pdf>
* AQM0802A-RN-GBW LCD Display
  + Overview: <https://international.switch-science.com/catalog/1407/>
  + Datasheet: <https://akizukidenshi.com/download/ds/xiamen/AQM0802.pdf>
* DC Motor
  + Overview: <https://www.adafruit.com/product/711>

Pins Used:

* Microcontroller (PIC24FJ64GA002)
  + SCLx, SDAx pins for I2C (Temp Sensor and LCD)
  + Output Compare (Motor Driver)
  + Input Capture (Push Button)
* Motor Driver (TB6612FNG)
  + PWMA -> RB6
  + AIN2 -> RB7
  + AIN1 -> RB8
  + STBY -> RB9
* Humidity & Temp Sensor (SHT40)
  + SCL -> SCL2
  + SDA -> SDA2
* LCD Display (AQM0802A-RN-GBW)
  + SCL -> SCL2
  + SDA -> SDA2
* Push Button
  + Input Capture -> RP5 (pull-up resistor)
* 5V DC Source (INSTEK GW3030D)
  + 5V DC -> VM
* Motor + Fan
  + A01
  + A02

**Full Documentation**

| finalproject\_main.c |
| --- |
| **void pic24\_init()**   * Purpose: Sets the clock speed of PIC24 to 16 MHz, and all pins to digital. * Arguments: None. * Outputs: None.   **int main(void)**   * Purpose: Runs all initialization functions except button initialization, displays real-time temperature, and compares the set temperature to decide whether to initialize the button to control the motor speed. * Arguments: None. * Outputs: Main function of the program and no integer outputs. |

| TempLib.c |
| --- |
| **void tempInit(void)**   * Purpose: Initializes SHT40, sets I2C rate to 100KHz, and writes temp data into sensor by high precision. * Arguments: None. * Outputs: None.   **void tempRead(unsigned char data[], int dataSize)**   * Purpose: Master receives data[] from sensor by reading temp data. * Arguments: An array data[] to save temperature data bytes, and dataSize determines the array’s length. * Outputs: None. |

| MotorLib.c |
| --- |
| **void initMotor(void)**   * Purpose: Initialization of the TB6612FNG Motor Driver and sets OC1RS to 0. * Arguments: None. * Outputs: None.   **void initPushButton(void)**   * Purpose: Initializes button to detect falling edge by pressing using Input Capture. * Arguments: None. * Outputs: None.   **void setSpeed(int speed)**   * Purpose: Changes value of speedMode and motor speed modes by changing OC1RS of PWM in Output Compare. * Arguments: The speed Mode to be changed. * Outputs: None.   **void \_IC1Interrupt(void)**   * Purpose: Changes the speed mode as the button was pressed and resets the flag to 0. * Arguments: None. * Outputs: None. |

| final\_LCDLib.c |
| --- |
| **void delay\_ms(unsigned int ms)**   * Purpose: Used for delay in millisecond. * Arguments: unsigned integer in millisecond. * Outputs: None.   **void lcd\_cmd(char Package)**   * Purpose: Sends an instruction byte to the LCD to configure it according to the peripheral data sheet. * Arguments: A package in char type. * Outputs: None.   **void lcd\_init(void)**   * Purpose: Initializes the LCD. * Arguments: None. * Outputs: None.   **void lcd\_setCursor(char x, char y)**   * Purpose: Sets the cursor from which characters can be set on the LCD. * Arguments: Column x and Row y. * Outputs: None.   **void lcd\_printStr(const char \*s)**   * Purpose: Prints a series of characters on the LCD based on the cursor. * Arguments: The string in char type to be printed. * Outputs: None. |

The software block diagram is shown below in Fig.2, which shows how the different functions interact with each other.

|  |
| --- |
| Figure 2: Software Block Diagram of the Program |

**Basic Usage Example:**

For the basic usage test, the room temperature is displayed on the LCD to test its functionality. The temperature is changed by pressing a fingertip to the sensor, and the LCD display is checked to view the change in temperature data. Using OC1RS in Output Compare, the motor driver is instructed to run the DC motor at a constant speed. This procedure ensures that the individual devices are working as intended.

**Advanced Usage Example:**

In the advanced usage test, a button is added to the system and Input Capture is set to trigger on falling edges. The IC Interrupt is called every time the button is pressed, where the speed mode of the motor is changed. The button is then tested to see whether the speed of the fan changes based on presses. The next step is to implement logic in the main function so that the fan only turns on when it goes above a certain temperature. The button press is disabled and the fan is turned off when it falls below the threshold. The value is set to 30 degrees Celsius. A hairdryer is used to raise the temperature, and the DC motor is checked to see whether it starts spinning automatically. The LCD display is checked to see if the temperature values still update when the fan is running. It is made sure that the fan turns off and the button press does not work when the temperature drops below 30. This procedure verifies the operation of the different components when working together.

**Header Files**

| final\_TempHeader.h |
| --- |
| #ifndef FINAL\_TEMPHEADER\_H #define FINAL\_TEMPHEADER\_H  #ifdef \_\_cplusplus extern "C" { #endif  void tempInit(void);  void tempRead(unsigned char data[], int dataSize);   #ifdef \_\_cplusplus } #endif  #endif /\* FINAL\_TEMPHEADER\_H \*/ |

| motorHeader.h |
| --- |
| #ifndef MOTORHEADER\_H #define MOTORHEADER\_H  #ifdef \_\_cplusplus extern "C" { #endif void setSpeed(int speed); void initMotor(void); void initPushButton(void); void \_\_attribute\_\_((interrupt, auto\_psv)) \_IC1Interrupt(void);  #ifdef \_\_cplusplus } #endif  #endif /\* MOTORHEADER\_H \*/ |

| final\_LCDHeader.h |
| --- |
| #ifndef FINAL\_LCDHEADER\_H #define FINAL\_LCDHEADER\_H  #ifdef \_\_cplusplus extern "C" { #endif void delay\_ms(unsigned int ms); void lcd\_cmd(char Package); void lcd\_init(); void lcd\_setCursor(char x, char y); void lcd\_printChar(char Package);  void lcd\_printStr(const char \*s);  #ifdef \_\_cplusplus } #endif  #endif /\* FINAL\_LCDHEADER\_H \*/ |

**C Libraries**

| TempLib.c |
| --- |
| #include <xc.h> #include <stdio.h> #include "final\_LCDHeader.h"  void tempInit(void){ // temp write    I2C2CONbits.I2CEN = 0; // disable i2c  I2C2BRG = 0x9D; // IC2 speed/rate: 100KHz from the Data sheet  I2C2CONbits.I2CEN = 1; // enable i2c  IFS3bits.MI2C2IF = 0; // clear interrupt flag    I2C2CONbits.SEN = 1; // start sending    while(I2C2CONbits.SEN == 1);   IFS3bits.MI2C2IF = 0;  I2C2TRN = 0b10001000 ; // slave address 0x44 and w    while(I2C2CONbits.SEN == 1);  IFS3bits.MI2C2IF = 0;  I2C2TRN = 0b11111101 ; // write command 0xfd (measure T & RH with high precision)    while(IFS3bits.MI2C2IF == 0);  IFS3bits.MI2C2IF = 0;  I2C2CONbits.PEN = 1; // send STOP   while(I2C2CONbits.PEN == 1); // wait for STOP to finish }   void tempRead(unsigned char data[], int dataSize){ // // temp read    IFS3bits.MI2C2IF = 0; // Clear interrupt flag    // send START  I2C2CONbits.SEN = 1; // send START  while(I2C2CONbits.SEN == 1); // wait for START to finish  IFS3bits.MI2C2IF = 0; // clear interrupt flag    // send slave address with W bit  I2C2TRN = 0b10001000 ; // slave address 0x44. write command  while(!IFS3bits.MI2C2IF); // wait for ACK  IFS3bits.MI2C2IF = 0; // clear interrupt   // send read register  I2C2TRN = 0xfd; // write command 0xfd (measure T & RH with high precision)  while(!IFS3bits.MI2C2IF); // wait for ACK  IFS3bits.MI2C2IF = 0; // clear interrupt    delay\_ms(10);   // send resend bit  I2C2CONbits.RSEN = 1; // repeated start bit  while(I2C2CONbits.RSEN == 1); // wait for the resend bit to be 0  IFS3bits.MI2C2IF = 0; // clear interrupt    // send slave address with R bit  I2C2TRN = 0b10001001; // slave address 0x44. read command  while(!IFS3bits.MI2C2IF); // wait for ACK  IFS3bits.MI2C2IF = 0; // clear interrupt    // get data  I2C2CONbits.RCEN = 1; // start receiving  int i;  delay\_ms(10);   for (i = 0; i < 5; i++) {  data[i] = I2C2RCV; // save temp bytes in data[]  while(!IFS3bits.MI2C2IF);  }   // send NACK  I2C2CONbits.ACKDT = 1; // use NACK  I2C2CONbits.ACKEN = 1; // start NACK sequence  while(I2C2CONbits.ACKEN == 1); // wait for NACK to finish sending    // send STOP  I2C2CONbits.PEN = 1; // send STOP  while(I2C2CONbits.PEN == 1); // wait for STOP to finish } |

| MotorLib.c |
| --- |
| #include <xc.h> #include "final\_LCDHeader.h" int speedMode = 0;   void setSpeed(int speed){ // change the speed of fan when fan turns on     if (speed==0){ // mode: 0, stop  speedMode = 0;  OC1RS = 0;  }  else if (speed==1){ // mode: 1, low speed (also is the initialization speed)  speedMode = 1;  OC1RS = 12000;  }  else if (speed==2){ // mode: 2, medium speed  speedMode = 2;  OC1RS = 24000;  }  else if (speed==3){ // mode: 3, high speed  speedMode = 3;  OC1RS = 36000;  } };   void initMotor(void){ // in1(RB8):H in2(RB7):L PWM(RB6):H STBY(RB9):H -----> clockwise  // \*\*\*\*\*speed control   TRISB = 0b1111110000111111; // RB6 through RB9 are set as outputs to motor driver  LATB = 0b0000001101000000; // set RB6,8,9 high    //PPS  \_\_builtin\_write\_OSCCONL(OSCCON & 0xbf); // unlock PPS  RPOR3bits.RP6R = 18; // use Pin RP6 for Output Compare 1 = "18" (Table 10-3)  \_\_builtin\_write\_OSCCONL(OSCCON | 0x40); // lock PPS    //TMR3  T3CON = 0; // stop Timer  T3CONbits.TCKPS = 1; // PRE 1:8  TMR3 = 0; // Initialize to zero  PR3 = 39999; // period for 20 ms with 1:8 prescaler  \_T3IF = 0; // clear Timer 3 interrupt flag    //OC1   OC1CON = 0; // turn off OC1 for now    OC1R = 200;   OC1RS = 0; // stop at initial (speed mode: 0)   OC1CONbits.OCTSEL = 1; // use Timer 3 for compare source  OC1CONbits.OCM = 0b110; // output compare PWM w/o faults     T3CONbits.TON = 1; // turn on the timer3 now! }  void \_\_attribute\_\_((interrupt, auto\_psv)) \_IC1Interrupt(void){  // every time the push button is pressed,   // the speed mode will increase by 1  delay\_ms(10); // delay for button debounce  \_IC1IF = 0; // clear input capture Interrupt    if (speedMode==0)   setSpeed (1);    else if (speedMode==1)   setSpeed (2);    else if (speedMode==2)   setSpeed (3);    else if (speedMode==3) // Reset the mode to 0 to stop the fan  setSpeed (0); }    void initPushButton(void){ // connect push button to RB5    //AD1PCFG & TRIS bits  TRISBbits.TRISB5 = 1; // RB5 to input  CNPU2bits.CN27PUE = 1; // pull-up resistor    //TMR2  T2CON = 0; // stop Timer  T2CONbits.TCKPS = 0b11; // PRE 1:256  TMR2 = 0; // initialize to zero  PR2 = 62499; // period for 1 sec with 1:256 prescaler  \_T2IF = 0; // clear timer 2 interrupt flag  \_T2IE = 0; // disable Timer 2 interrupt    //PPS  \_\_builtin\_write\_OSCCONL(OSCCON & 0xbf); // unlock PPS  RPINR7bits.IC1R = 5; // Use Pin RP8 = "5", for Input Capture 1 (Table 10-2)  \_\_builtin\_write\_OSCCONL(OSCCON | 0x40); // lock PPS    // IC1  IC1CON = 0; // reset IC1   IC1CONbits.ICTMR = 1; // timer 2  IC1CONbits.ICM = 0b010; // capture on every falling edge    \_IC1IF = 0;  \_IC1IE = 1; // enable IC1 interrupt      T2CONbits.TON = 1; // turn on the timer2 now! } |

| final\_LCDLib.c |
| --- |
| #include "xc.h" #include "final\_LCDHeader.h" //#define CONTRAST 0b0000 // Make sure the contrast is adjustable   void delay\_ms(unsigned int ms){ // set X ms delay  while(ms-- > 0){  asm("repeat #15998");  asm("nop");  } }  void lcd\_cmd(char Package){ // Sends an instruction byte to the LCD to configure   //it according to the peripheral?s data sheet.    //Send START  IFS3bits.MI2C2IF = 0; //Clear interrupt flag  I2C2CONbits.SEN = 1; //Begin (S)tart sequency    while(I2C2CONbits.SEN == 1);// SEN will clear when Start Bit is complete  IFS3bits.MI2C2IF = 0;  I2C2TRN = 0b01111100; // 8-bits consisting of the slave address and the R/nW bit    while(IFS3bits.MI2C2IF == 0);  IFS3bits.MI2C2IF = 0;  I2C2TRN = 0b00000000; // 8-bits consisting of control byte    while(IFS3bits.MI2C2IF == 0);  IFS3bits.MI2C2IF = 0;  I2C2TRN = Package; // 8-bits consisting of the data byte    while(IFS3bits.MI2C2IF == 0);  IFS3bits.MI2C2IF = 0;  I2C2CONbits.PEN = 1; //Stop bit    while(I2C2CONbits.PEN == 1);// PEN will clear when Stop bit is complete }   void lcd\_init(void){ //Initializes the LCD using the lcd\_cmd function by data sheet.     I2C2CONbits.I2CEN = 0; //Disable I2C for safe  I2C2BRG = 0x9D; // IC2 speed/rate: 100KHz from the Data sheet  IFS3bits.MI2C2IF = 0; //Clear interrupt flag  I2C2CONbits.I2CEN = 1; //Enable I2C    delay\_ms(40);    lcd\_cmd(0b00111000); // function set, normal instruction mode  lcd\_cmd(0b00111001); // function set, advanced instruction mode  lcd\_cmd(0b00010100); // interval osc  lcd\_cmd(0b01010110); // contrast low  lcd\_cmd(0b01011110);   lcd\_cmd(0b01101100); // follower control    delay\_ms(200);    lcd\_cmd(0b00111000); // function set, normal instruction mode  lcd\_cmd(0b00001100); // Display On  lcd\_cmd(0b00000001); // Clear Display     delay\_ms(1); }   void lcd\_setCursor(char x, char y){ // Uses the lcd\_cmd function to set the cursor from which   // characters can be set on the LCD. Uses values x and y, column and  // row    char cursor = ( (0x40 \* y) + x) | 0b10000000; // Sets cursor using x, y  lcd\_cmd(cursor);  };    void lcd\_printChar(char Package){ // A test function that sets a single character  // on the LCD based on the position of the cursor    I2C2CONbits.SEN = 1;// Initiate Start condition    while(I2C2CONbits.SEN == 1);// SEN will clear when Start Bit is complete  IFS3bits.MI2C2IF = 0;  I2C2TRN = 0b01111100; // 8-bits consisting of the slave address and the R/nW bit    while(IFS3bits.MI2C2IF == 0);  IFS3bits.MI2C2IF = 0;  I2C2TRN = 0b01000000; // 8-bits consisting of control byte w/ RS=1    while(IFS3bits.MI2C2IF == 0);  IFS3bits.MI2C2IF = 0;  I2C2TRN = Package; // 8-bits consisting of the data byte    while(IFS3bits.MI2C2IF == 0);  IFS3bits.MI2C2IF = 0;  I2C2CONbits.PEN = 1;  while(I2C2CONbits.PEN == 1);// PEN will clear when Stop bit is complete   };   void lcd\_printStr(const char \*s){    I2C2CONbits.SEN = 1;//START BIT    while(I2C2CONbits.SEN == 1);// SEN will clear when Start Bit is complete  IFS3bits.MI2C2IF = 0;  I2C2TRN = 0b01111100; // 8-bits consisting of the slave address and the R/nW bit    while(\*s){    if(\*(s+1)!='\0'){ // Not the last char  while(IFS3bits.MI2C2IF == 0);  IFS3bits.MI2C2IF = 0;  I2C2TRN = 0b11000000; // 8-bits consisting of control byte w/ RS=1, C0=1    while(IFS3bits.MI2C2IF == 0);  IFS3bits.MI2C2IF = 0;  I2C2TRN = \*(s); // 8-bits consisting of the data byte    }else{ //the last char  while(IFS3bits.MI2C2IF == 0);  IFS3bits.MI2C2IF = 0;  I2C2TRN = 0b01000000; // 8-bits consisting of control byte w/ RS=1, C0=0    while(IFS3bits.MI2C2IF == 0);  IFS3bits.MI2C2IF = 0;  I2C2TRN = \*(s); // 8-bits consisting of the data byte  }  s++;    }    while(IFS3bits.MI2C2IF == 0);  IFS3bits.MI2C2IF = 0;  I2C2CONbits.PEN = 1; // STOP BIT  while(I2C2CONbits.PEN == 1);// PEN will clear when Stop bit is complete   }; |

**Main Function**

| finalproject\_main.c |
| --- |
| #include "xc.h" #include <stdio.h> // for sprintf #include "final\_LCDHeader.h" #include "final\_TempHeader.h" #include "motorHeader.h"  // CW1: FLASH CONFIGURATION WORD 1 (see PIC24 Family Reference Manual 24.1) #pragma config ICS = PGx1 // Comm Channel Select (Emulator EMUC1/EMUD1 pins are shared   // with PGC1/PGD1) #pragma config FWDTEN = OFF // Watchdog Timer Enable (Watchdog Timer is disabled) #pragma config GWRP = OFF // General Code Segment Write Protect (Writes to program   // memory is allowed) #pragma config GCP = OFF // General Code Segment Code Protect (Code protection is   // disabled) #pragma config JTAGEN = OFF // JTAG Port Enable (JTAG port is disabled)   // CW2: FLASH CONFIGURATION WORD 2 (see PIC24 Family Reference Manual 24.1) #pragma config I2C1SEL = PRI // I2C1 Pin Location Select (Use default SCL1/SDA1 pins) #pragma config IOL1WAY = OFF // IOLOCK Protection (IOLOCK may be changed via unlocking   // seq) #pragma config OSCIOFNC = ON // Primary Oscillator I/O Function (CLKO/RC15 functions as   // I/O pin) #pragma config FCKSM = CSECME // Clock Switching and Monitor (Clock switching is enabled,   // Fail-Safe Clock Monitor is enabled) #pragma config FNOSC = FRCPLL // Oscillator Select (Fast RC Oscillator with PLL module (FRCPLL))  void pic24\_init(){  \_RCDIV = 0; //Sets 16MHz frequency for PIC24  AD1PCFG = 0xffff; //Sets pins to digital }   int main(void) {     pic24\_init(); // PIC24 initialization first  unsigned char tempbits[6]; // save the temp data to [0] and [1] got from temp   // sensor  unsigned long int temp; // get the long int "fake" temp by tempbits[0] and   // tempbits[1]      initMotor(); // initialize the Motor, PWM makes it stop   // (speedMode: 0)  lcd\_init(); // initialize the LCD    while(1){    tempInit(); // write temp by sensor  delay\_ms(10); // delay 10 ms until finish read  tempRead(tempbits,6); // read temp data in tempbits array with six bytes   // length     temp = (unsigned long int)tempbits[0]\*256+(unsigned long int)tempbits[1];   // get the long int "fake" temp   double decTemp; // "real" temp with decimals   decTemp = -45.0+175.0\*(temp)/65535; // get the "real" temp with decimals from "fake"   // temp by particular math calculation  if(decTemp>100){   decTemp = 100.0; // set the upper limit (the highest temp is 100   // degree in Celsius)  }  if(decTemp<0){ // set the lower limit (the lowest temp is 0 degree   // in Celsius)  decTemp = 0.0;  }      lcd\_setCursor(0,0);  lcd\_printStr("Temp:"); // LCD display (first line)  lcd\_setCursor(0,1); // display decimal temp (second line)  char realTemp[8];    sprintf(realTemp, "%5.2f D", decTemp); // ?x.xx D?   // 5.2 in the format string ?%5.2f? means 5 place holders   // for the whole floating-point number, 2 of which are   // for the fractional part.  lcd\_printStr(realTemp); // prints to the LCD with the decimal temp    if(decTemp >= 30.0){ // when temp >= 30 degree, makes the fan turn  // automatically in low speed  // and we can control speed by push button    setSpeed(1); // initializes the speed to mode 1((low))  initPushButton(); // initializes the push button  }  while(decTemp >= 30.0){ // always running till the temp < 30 degree in order to   // automatic fan with speed controls   delay\_ms(10); // Again! delay 10 ms until finish read   tempRead(tempbits,6); // Again! update temp data in tempbits array with six   // bytes length    temp = (unsigned long int)tempbits[0]\*256+(unsigned long int)tempbits[1];   decTemp = -45.0+175.0\*(temp)/65535; // get the "real" temp with decimals  if(decTemp>100){   decTemp = 100.0; // set the upper limit (the highest temp is 100   // degree in Celsius)  }  if(decTemp<0){ // set the lower limit (the lowest temp is 0 degree   // in Celsius)  decTemp = 0.0;  }    lcd\_setCursor(0,0);  lcd\_printStr("Temp:"); // LCD display (first line)  lcd\_setCursor(0,1);  char realTemp[8]; // display decimal temp (second line)   sprintf(realTemp, "%5.2f D", decTemp); // ?x.xx D?   // 5.2 in the format string ?%5.2f? means 5 place   // holders for the whole  // floating-point number, 2 of which are for the   // fractional part.  lcd\_printStr(realTemp); // Again! prints to the LCD with the decimal temp  if(decTemp < 30.0){  break; // Once temp < 30 degree, dumps out of while loop  }  }    // when temp < 30 degree, makes motor speed   // stop(mode: 0)  setSpeed(0);  CNPU2bits.CN27PUE = 0; // when temp < 30 degree, disables the push button    };    return 0; } |

**Conclusion**

This project makes use of three peripherals: Motor Driver, Humidity & Temperature Sensor, and LCD Display. The device libraries allow for the automatic toggle of a fan with a temperature limit, and the manual control of fan speed with the push of a button. Many important concepts were put into practice through working on this project. Some highlights are sending and receiving packets of data using I2C, using Input Capture to detect the falling edge of a button press, and utilizing Output Compare for PWM control. The project helped build experience working with new devices such as the Motor Driver and Humidity & Temperature Sensor. There were difficulties encountered throughout the course of this work. Some errors with the project are that the buttons are not stable enough, resulting in the speed modes changing unexpectedly at times. Currently, a 10 ms delay is implemented for the purpose of button debouncing, but a more robust system could be put in place in the future. The second issue has to do with reading I2C data from the temperature sensor. According to the datasheet, the sensor sends two bytes of temperature data. However, the second byte is not read correctly. Another issue in the project design is that the sensor can quickly rise in temperature, but is slow to dissipate the heat. So the temperature values take a longer time to cool down than to heat up. Overall, the project was a successful implementation of a temperature-controlled fan. It has good practical significance by presenting itself as a viable and cheap solution to systems that require cooling down or improving airflow.