Trinity University
Digital Commons @ Trinity

Mechatronics Final Projects | Engineering Science Department

5-2018

Variable Input Pressure Regulator

Sean Farrell
Trinity University, sfarrel1@trinity.edu

Dupri Grimes
Trinity University, dgrimes@trinity.edu

William McElvogue
Trinity University, wmcelvog@trinity.edu

Nathan Richter
Trinity University, nrichte1@trinity.edu

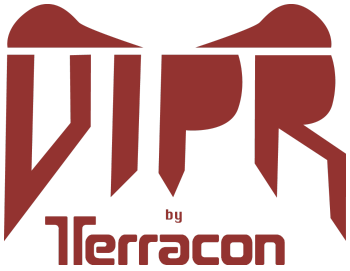
Follow this and additional works at: https://digitalcommons.trinity.edu/engine_mechatronics

 Part of the Engineering Commons

Repository Citation
Farrell, Sean; Grimes, Dupri; McElvogue, William; and Richter, Nathan, "Variable Input Pressure Regulator" (2018). *Mechatronics Final Projects*. 13.
https://digitalcommons.trinity.edu/engine_mechatronics/13

This Report is brought to you for free and open access by the Engineering Science Department at Digital Commons @ Trinity. It has been accepted for inclusion in Mechatronics Final Projects by an authorized administrator of Digital Commons @ Trinity. For more information, please contact jcostanz@trinity.edu.

Variable Input Pressure Regulator



Group C:

Sean Farrell

Dupri Grimes

William McElvogue

Nathan Richter

ENGR 4367

Spring 2018

Course Instructor: Dr. Kevin Nickels

TABLES OF CONTENTS

1. Design Summary

1.1 Motivation

The variable input pressure regulation system was designed to carry out a test in accordance with the ASTM E1105 standardized commercial window testing procedure. This testing standard dictates that a negative pressure must be maintained within 2% of a constant value while a curtain wall spray system saturates the window from the outside. The test lasts around 15 - 20 minutes depending on the window type, though it may last longer if the user chooses. Currently, the test is performed by a user regulating differential pressure with a variable pump power dial, a gate valve, and a high precision liquid manometer. This design aims to automate the system, allowing variable input and high precision pressure regulation.

1.2 Operation

This design operates by first executing an initialization loop. When the system is started, the ball valve will fully open for safety purposes. Next the system prompts the user for input to either start a test or set test parameters. Once the test has been set up and executed, the LCD will display the set pressure and measured pressure in both inches of H_2O and pound force per square foot. The ball valve will then be actuated using a variation of ON/OFF control to reach within 2% of the set pressure. Audio alerts will be activated at 20, 15, 10, 5, and 1 minute remaining, as well as 45, 30, 15, 5, 4, 3, 2, and 1 second remaining. Once the test is concluded, the pressure in the chamber is continually controlled until the user cancels the operation. Additional goals include the implementation of a calibration feature, an auto shut off safety feature and the logging of pressure data for later use.

2. System Details

2.1 User Interface

The user interface for the VIPR system was run on the PIC18F2550 microcontroller and included a 4x4 hex keypad and a 4x20 character LCD screen. This microcontroller was chosen because of the large number of pins available for I/O, 28. The user interface required 6 pins for the LCD screen and 8 pins for the keypad, which would have been more than a standard

microcontroller, like the PIC16F88, could have provided. The wiring diagram for the user interface circuit is shown in Figure 1.

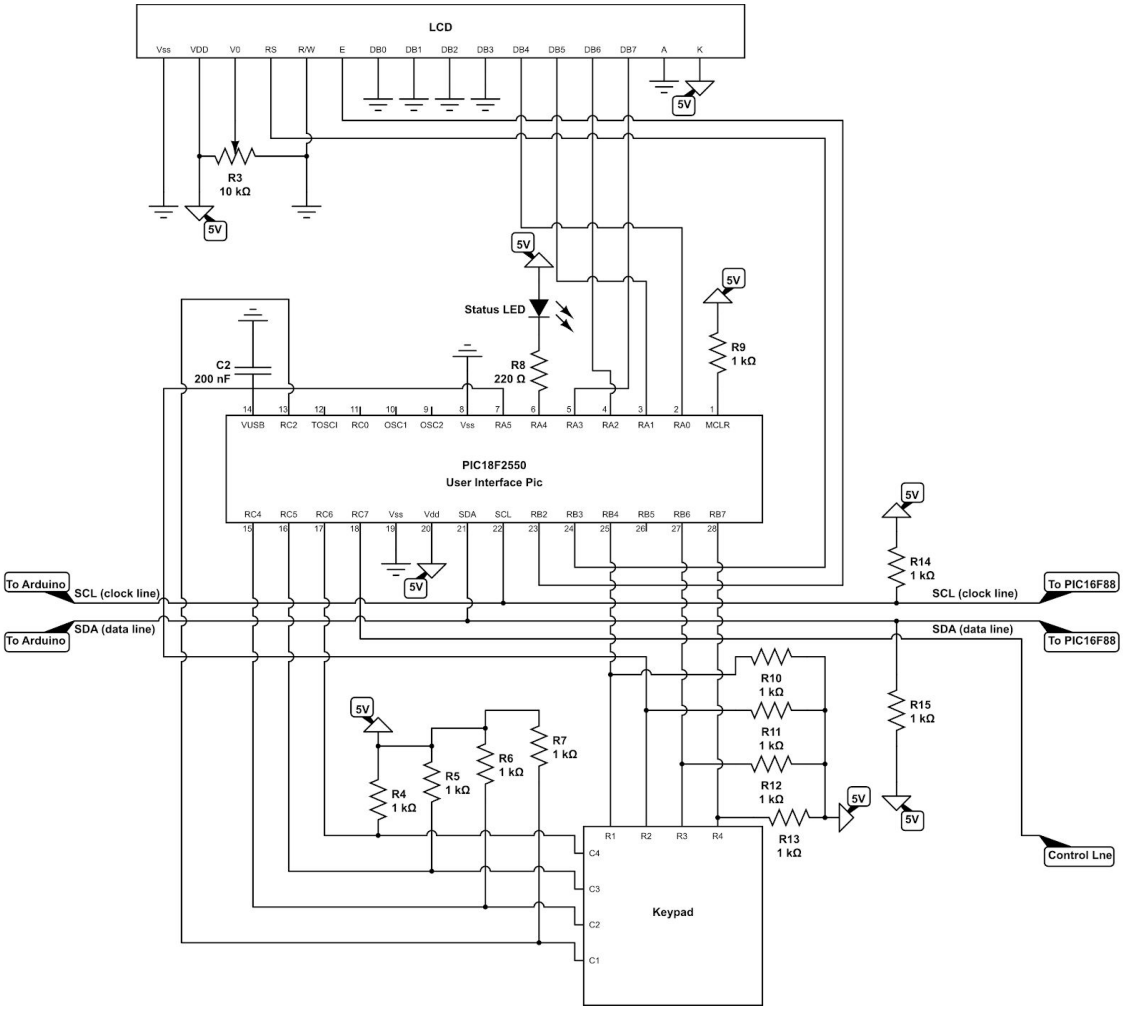


Figure 1. Wiring diagram of PIC18F2550 with user interface peripherals.

The LCD was used to display instructions and menus for the user. This portion of the project used the lcdout command of PicBasic Pro extensively, which simplified greatly the work needed to display strings and variables onto the screen. On startup, the display showed a main menu with two options: one to setup a test and one to run the test. The process of this main menu is shown in the flowchart of Figure D4, in the Appendix. Selecting the test setup option guided the user using a set of instructions displayed on the LCD. These instructions told the user how to confirm values and cancel the setup. Selecting the run test option caused the LCD to display the pressure and time remaining in the test. The display included both the setpoint and current pressure, in units of inH_20 and psf. The program for the run test option is shown in Figure D6.

The keypad was used to navigate through the various states and instructions programmed onto the microcontroller. From the main menu the user could setup the test by pressing the "A" key, while they could run a test using the "B" key. In general the "*" key would cancel whatever process was occuring, returning the user to the main menu. Meanwhile, the "#" would confirm a user input. The keypad was used extensively in the test setup option to enter the setpoint pressure and test duration for the pressure regulation. These were entered by inserting the digit of each pressed key at the end of the previous value, shifting over previous digits. This process is detailed in the test setup flowchart, given in Figure D5. Meanwhile, in the run test option the keypad was only used to cancel the test and return the user back to the main menu. An additional calibration routine was planned as a main menu option and its user interface was written but was not implemented fully. Additionally, the test setup option also included a subroutine and instructions which would have allowed the user to enter a filename using an alphanumeric keypad. However, because data logging was never implemented fully, this subroutine was unnecessary for the current version of the prototype.

2.2 Control

Pressure control in the testing rig was actuated using a DC motorized ball valve, and controlled with a PIC16F88 microcontroller that operated a L293B H-bridge. At the beginning of the control cycle, the valve moved to the fully open position before attempting to read the controlling state or any set point information from the arduino. Additionally, the PIC was also responsible for performing an analog to digital conversion of the 0V - 5V manometer pressure output. Because of the low pin requirements of the various pieces of hardware necessary for pressure control the PIC16F88 was deemed acceptable.

The chosen control scheme for this project was ON/OFF control, as there was little difference in speed of the motor valve when higher voltages were used. Additionally, the control scheme needed to be very sensitive as it is difficult to regulate the pressure of a compressible gas such as air within 2% of the set point. When ON/OFF control was first implemented, we noticed large fluctuations in pressure around the set point value. By modifying the control scheme to implement pauses in between pressure readings, we were able to finalize the control scheme as seen in Figure 2.

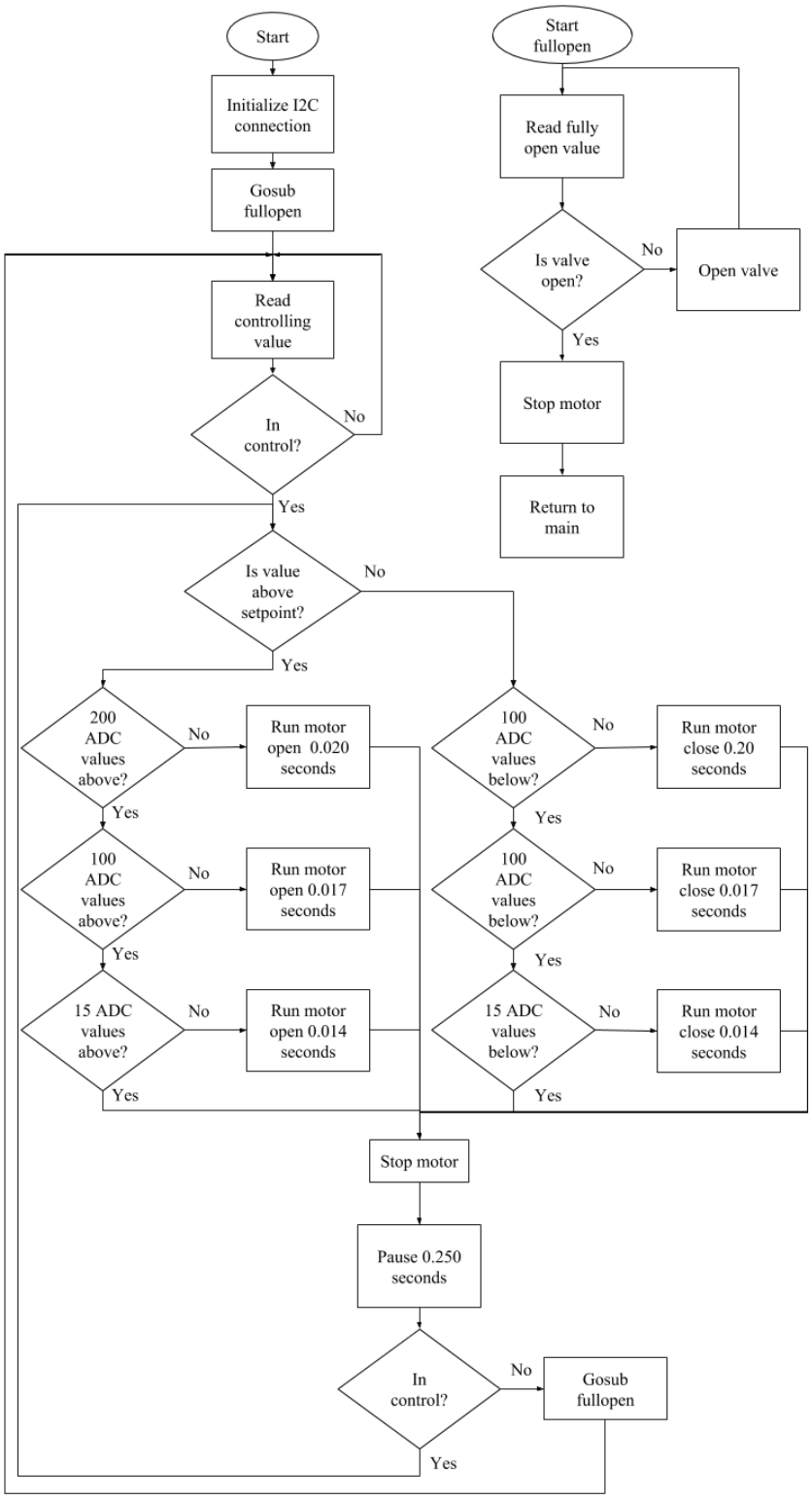


Figure 2. ON/OFF control scheme for small pressure regulation

A calibration of the digital manometer was performed using a liquid manometer attached to the same testing rig seen in Fig. B3. The internal control was set to regulate to a certain analog digital conversion value, then the pressure value on the liquid manometer was recorded in inches of water. This was performed for ADC values ranging from 0 to 800. Using this data a calibration curve was created and implemented on the PIC18F2550. One complication occurred where the ADC value recorded by the PIC microcontroller was different from that of the arduino. When checked against a separate digital multimeter, the value of the arduino ADC was around 100-200 ADC values different from the PIC ADC value. There is not yet a solution to this issue, we will likely perform a calibration using PIC ADC, arduino ADC, and a liquid manometer to provide further insight to this issue.

2.3 Communication

In this design we used two type of communication: SPI and I2C. The SPI communication was used to read .WAV files from an SD card using the Arduino Nano. The audio files needed to be formatted at a 16000 sample per second (Hz), mono channel, and 8 bits per sample. The TMPcpm.h library was used in the Arduino code, because it had internal functions for reading the .WAV audio format and playing it on a speaker. We encountered one problem with the SPI library when we tried to switch between reading and writing to and from the SD card. We found that we could either write data to the SD card or play audio files from it. To play the audio file a LM386 low power audio amplifier was to drive the 8 ohm speaker. We found that adding a low pass filter with a cutoff at approximately 15,000 Hz helped reduce the high frequency noise. However, we found that when the pump was running the speaker was audible but not as loud as we wanted it be during tests. Overall, just being able to play audio files did not inhibit the success of our project because we were tasked with playing audio during the pressure regulation test. Saving data to the SD card is a feature we hope to add in future improvements.

The I2C communication was used to transfer pressure, audio index, calibration, time, and motor control data between the PIC18F2250, PIC16F88, and Arduino Nano. The PIC's both acted as masters while the Arduino was the slave device. In order to get the I2C communication part of the project to function we found several key software details that were necessary: 1) we found that you had to leave the PIC in the default serial communication mode which actually set

it as a master in SPI, why this worked for I2C we are still unsure. 2) In order to enable the PIC for I2C communication we needed to set SSPCON.5 =1, which enables the serial ports and configures the SDA and SCL pins as serial ports and if set zero they are just I/O ports. We used this to toggle between which master was enabled for our project which allowed us to have two masters with only one slave Arduino. 3) On the PIC's the register CMCON was set to $7 to turn off all the comparators and ANSEL was set to 0 to turn off the analog select registers. 4) We found that while the Arduino and PIC had to have the same address the PIC register had to be shifted by one since the PIC used and 8-bit addressing protocol and the Arduino used a 7-bit. 5) Lastly, the PIC's needed to have I2C_HOLD activated which enables the PIC's to hold the clock line low when transfering on the I2C bus. On the arduino side there was more support for using I2C communication and we just used the Wire.h library and the internal I2C functions within that library.

3. Design Evaluation

Overall the design worked well; however, it did fail to fulfill several of the goals our team established at the beginning of the project. The VIPR system was not able to regulate pressure to within 2% of the setpoint, nor was it able to reach the setpoint pressure within 30 seconds. Additionally, automatic data logging was not implemented fully due to time constraints and an auto-shutoff feature was never created. However, it did allow for the user to easily change the setpoint pressure and duration of a test easily and provided an audio countdown at the required intervals. It was also able to maintain pressure in the testing rig for the test duration and afterwards, as it was designed to. The fact most of these goals were not fully implemented is not an issue; they were ambitious and meant to drive our team to produce a fully integrated and patentable product. However, in terms of fulfilling the course requirements for the project, this project was extremely successful, implementing a highly functional system with pieces which took significant time to research and develop.

The output display, which consisted of the LCD screen, was highly functional and repeatedly worked as intended. This part of the project required significant effort on our part to develop. Although the Pic Basic Pro language contains an lcdout command and such a command was presented in class lecture, the actual implementation of this piece was quite difficult. The

LCD content presented in the book and lecture did not match at all the level of effort required to be fully operational. This piece of hardware was implemented on the PIC18F2550, which was not covered in the class or book. In addition the display worked with the keypad, updating the screen with the pressed keys to allow the user to correctly enter a pressure and time. Due to the large amount of time implemented with this piece of software and the extra piece of functionality built into its operation, we expect between 15 and 20 points for this portion of the system.

The audio output, consisting of the speaker, was another highly functional piece of hardware which was essential to the project. The audio output was capable of playing audio files reliably and due to filtering, clearly. Although this piece of the project was implemented on the Arduino, it still took significant effort to work properly and would not have been possible on a PIC due to the software challenges in using SPI communication. The pre-recorded voice files which it played were a step-up from the simple tones which could have also fulfilled this category. However, the customer specifically requested this functionality, so we were obligated to fulfill it. The only thing which could have made the audio system better was a louder speaker. The one used was not loud enough to be clearly heard during the test. Due to the high functionality and significant effort put forth with this portion of the project we expect between 15 and 20 points.

The simple 4x4 keypad implemented for manual user input worked consistently well and had a standard level of functionality. The keypad was presented in detail in class and lab so we did not expect to receive many points for it. However, to try and further its functionality we decided to implement alphanumeric capabilities, allowing the user to type in letters, as well as numbers using the same technique employed in old phones. This required a good portion of time to develop and as such we expected to receive 15 points for it, even though it was presented in lab in detail. However, while this piece of functionality was fully implemented and worked consistently well in the end it was unnecessary as data logging was never implemented. As such we only expect to receive 10 points for this category.

The digital manometer used in the VIPR system was a part we think was really unique. Examining the parts recommendations from past students, there were no differential pressure transducers selected, therefore it was determined no one had attempted this on a previous project.

As such, there were no real resources to assist us in developing this portion of the project. Despite this the digital manometer was still a highly functional piece of hardware which did take some effort to integrate. Although the output was an analog signal in 0 to 5V range it did require development of a voltage buffer to work properly with the PIC16F88. Due to this, the fact that it was not covered in lab or class and its high functionality we expect 15 point or more in this category.

The actuator used for pressure control is a U.S. Solid ¾" threaded motorized ball valve. This valve is designed to be used with sprinkler systems and other liquid control systems. The use of this valve to control air at such a low pressure was ambitious, but if implemented correctly would provide a huge cost benefit over pressure regulators designed for the same pressure range that cost over $100. The motor was operated using a L293B H-bridge to control the variation of an ON/OFF control scheme. Using a ¾" PVC pipe, the motor provided a relief path to regulate the control pressure, this increased testing rig safety and limited wear on the vacuum pump. Additionally the motor had two extra outputs, a fully open indicator and a fully closed indicator. The fully open indicator was used to open the system when the test was completed or the system was initialized. Other than this simple wrinkle and the modifications on traditional ON/OFF control, the DC motor and H-bridge were covered in class, and therefore we expect 10 to 15 points in this category.

This project had a number of components which can fit under the logic, processing, control and miscellaneous category. The ball valve used an ON/OFF control with great success which was presented in detail in class. However, the project also required decimal mathematical operations, which required significant effort to select the proper data size and choose the right order of operations to prevent overflow and aliasing of variables. However, the biggest contributor to this category which required the most significant amount of effort was the I2C communication. I2C communication was essential to the project as multiple bytes of information had to be transferred between microcontrollers. This required weeks of research and development by half of our design team and exploration of online resources not provided in the book or lecture. In the end the code developed for I2C communication was original and worked reliably, allowing both PICs to act as master to the slaved Arduino. We feel the complex

software required for I2C communication and decimal math more than outweighs the simple on/off control used for the motor. Due to the significant, even strenuous amount of effort and research required just for I2C communication we believe we deserve 20 points.

Due to the visible and meaningful amount of effort we all exerted on this project, we expect 10 extra points to the final grade. Additionally, we felt our product was average in terms of consumer appeal. Additionally, we expect a few points, maybe 5, for frugality, since we chose to implement cheaper controllers and used components which were justifiably expensive. In terms of deductions we expect none, as our project was well integrated, performed well, was safe and assembled well, used the minimum number of controllers and did not use inappropriate or expensive power sources.

4. Partial Parts List

We used many extra components when making our pressure regulation device. The customer requested the device meet certain requirements and had certain features, so we used components that were not taught in the course or covered in the textbook. The parts we used can be seen in Table 1. The customer needed 2% accuracy for the pressure readings, so a high-end digital manometer was necessary. The motorized ball valve was used for regulating the pressure, and the H-bridge was used to reverse the motor direction. The low-power amplifier was used for the audio speaker. The micro-SD card reader was used to access the .WAV files that were played on the speaker. Finally, the Arduino Nano read the audio file from the SD card, playing them on the speaker, and read the digital manometer readings, sending them to the PIC18F2550.

Table 1. Partial Parts List

Part Name	Model Number	Vendor	Price
Dwyer Digital Manometer	616KD-B	Dwyer	$68.00
Motor H-Bridge	L293B	ST	$3.73
Motorized Ball Valve	3/4"	US Solid	$52.99
Low-Power Audio Amplifier	LM386	Texas Instruments	$0.48
Micro-SD Card Reader	254	Adafruit	$7.50
Arduino Nano	V3.0	Arduino	$7.86

5. Lessons Learned

Through this semester long project our group learned early on that in order to produce a functioning and reliable pressure regulation system we needed to manage our time and start on the project early. We had a general functional diagram and parts list way in advance compared to the deliverable due date. This allowed us to start constructing our extensive code both in BASIC and C for the PIC's and Arduino nano we wanted to use in our project. Without managing our time efficiently we would not have been able to overcome the I2C communication, power supply, and motor control issues we faced.

One challenge we faced early in our project was figuring out how to use a PIC to communicate to a SD card and read .WAV audio files. After extensive research we found little to no support for BASIC SPI communication for the PIC microcontrollers. The support we did find was for FAT16 SD cards written in C which only support 2GB SD cards. Since we wanted to use a 16GB card, which used FAT32, we would have needed to write our code in BASIC. We

decided to use the Arduino nano for the SPI communication to the SD card because there was more support for this type of endeavor online. However, we were only able to either save data to an SD card as a .CSV file or play .WAV audio files from the SD card. We could not figure out how to switch between reading and writing to and from the SD card during the program. This is a problem we hope to address for future improvements to our project. Currently, we have code written that saves pressure data with a user input file name to an SD card, it is commented out within the Arduino main program code located in Appendix E.

For our project to function properly we needed to send data between the PIC and arduino microcontrollers. This led us to our greatest challenge but also our greatest achievements in this project. We learned how to use I2C communication to send data with dual PIC masters and the Arduino nano as the slave. After weeks of extensive research learning how I2C functions and also how to set it up for the PIC's and Arduino we found some key things that were not clearly discussed online. These points are addressed in the system detail communication section of the report. Learning how to use I2C communication was crucial for making our project function, and it also broadened knowledge on a subject which has not been taught extensively in previous classes.

When we were interfacing all of the microcontrollers together into one breadboard we learned how unreliable breadboard connections can be and how important supplying the proper power is to have a functioning project. We noticed a lot of fluctuations in how our project functioned each time we powered it up and after spending a good portion of our time debugging our circuit we discovered the breadboard was making bad connections where the Arduino was located. After we moved the Arduino to a separate board and reconnected it to the rest of the circuit, the pressure regulator started to work again. On the power side we believe the ball valve motor was consuming too much power, and thus affecting the performance of the whole project. We decided to power the ball valve with a separate 12 V battery pack, solving our power shortage problems.

Overall, we learned the importance teamwork, if we all had not contributed large amounts of effort or have interest in this project it would not have gotten completed. We learned the benefits of delegating work and communication within the group. Nathan worked on

developing the user interface and I2C communication using the PIC18F2550, Dupri and William worked on the motor control, calibration of the system, and construction of the testing rig, while Sean worked on playing audio files from the SD card using the arduino and I2C communication between the PIC16F88 and Arduino. Throughout the semester we set goals for each section of the project and divided tasks accordingly. It was this divide and conquer approach, along with our willingness to help each other when we encountered roadblocks, that made the VIPR system a qualified success.

6. APPENDICES

A. Functional Diagram

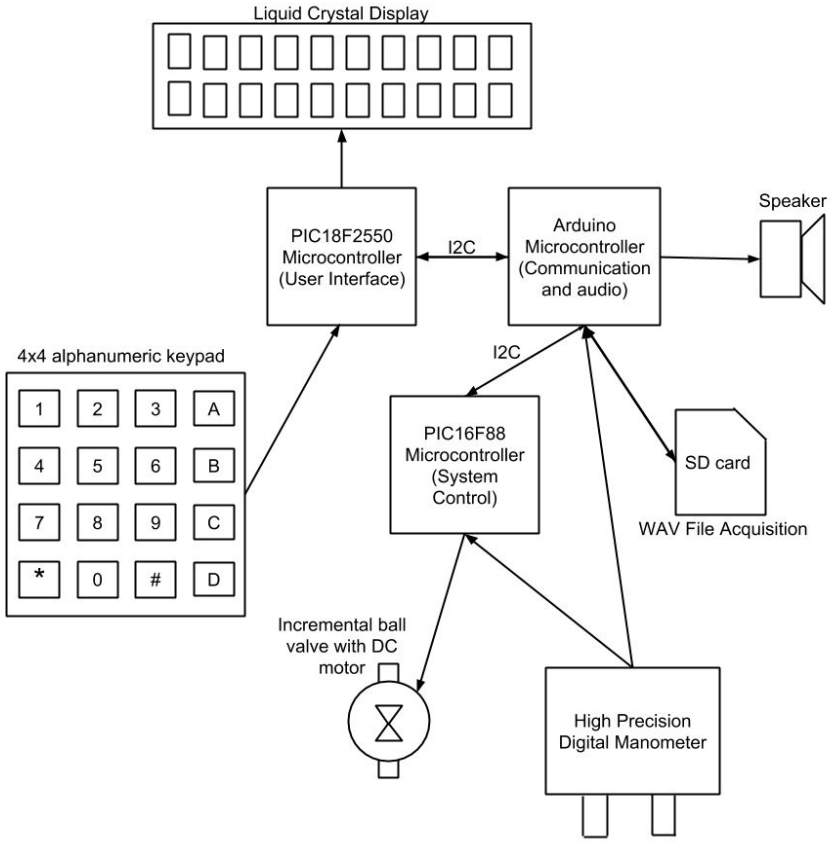


Figure A1. Final version of functional diagram

B. Design Photos

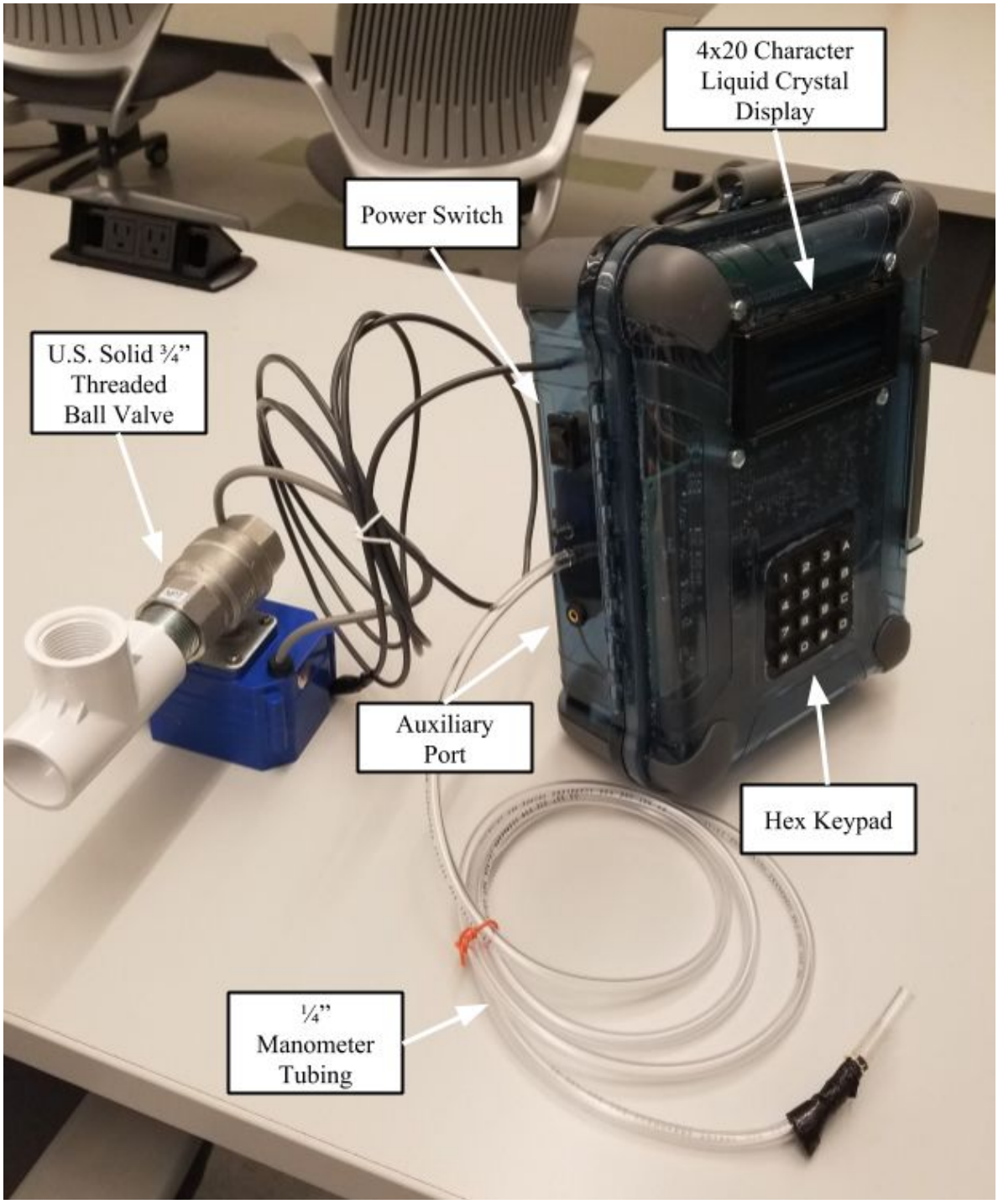


Figure B1. External diagram of boxed solution

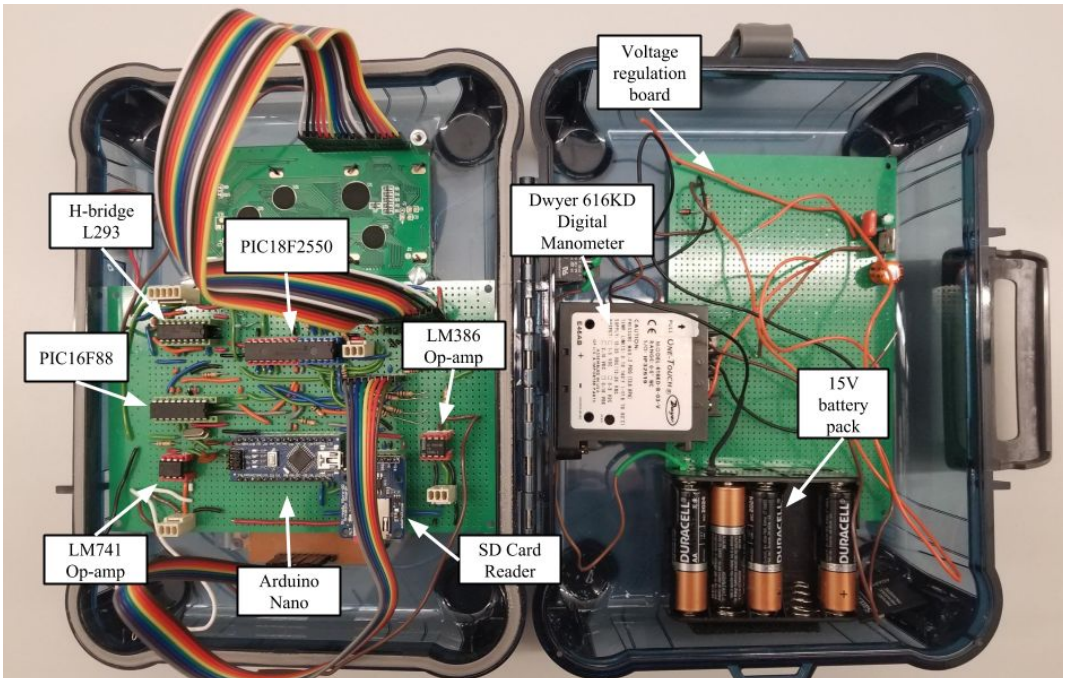


Figure B2. Internal diagram of boxed solution

Figure B3. Testing rig

C. Detailed Wiring Diagram

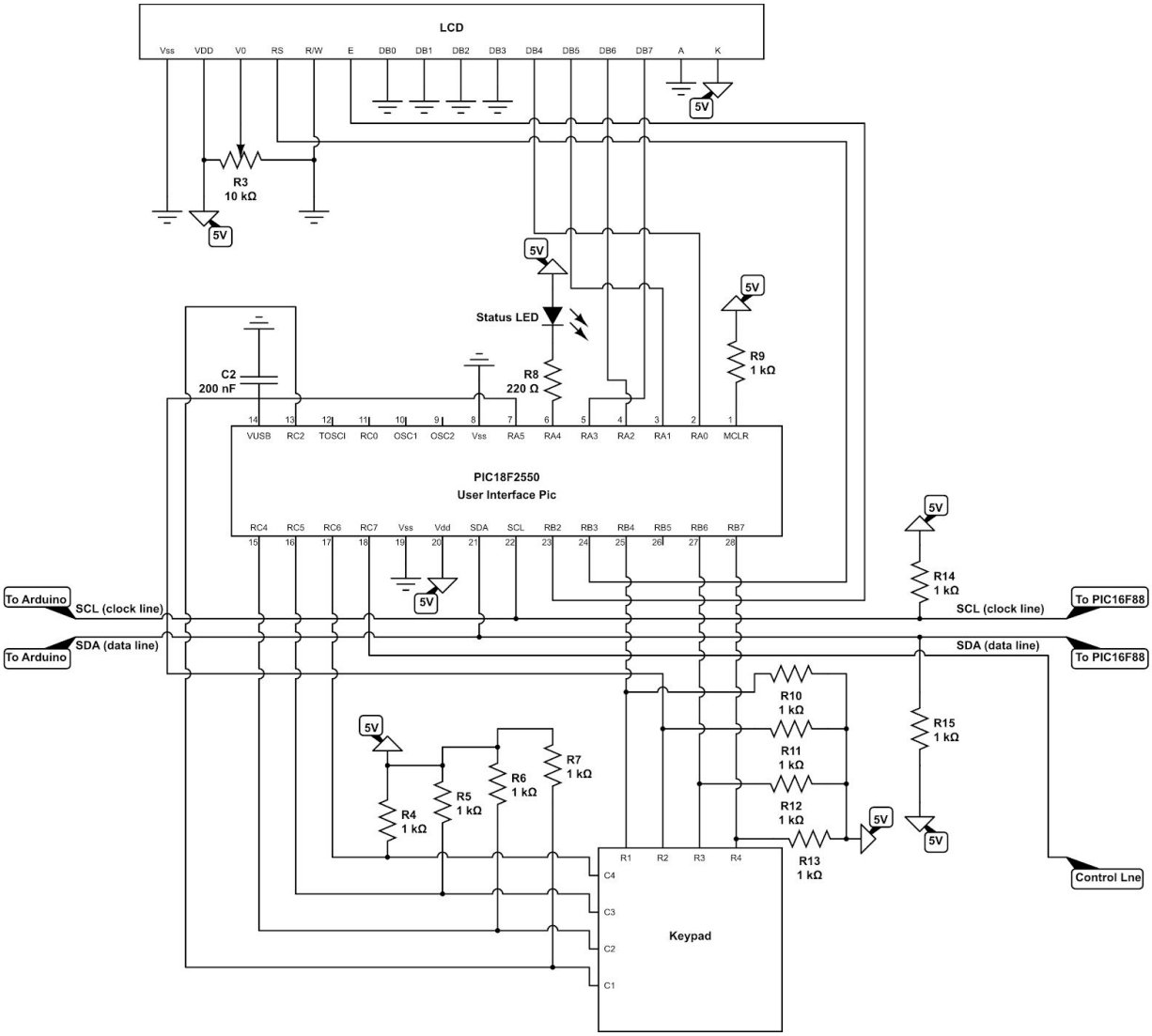


Figure C1. PIC18F2550 wiring diagram

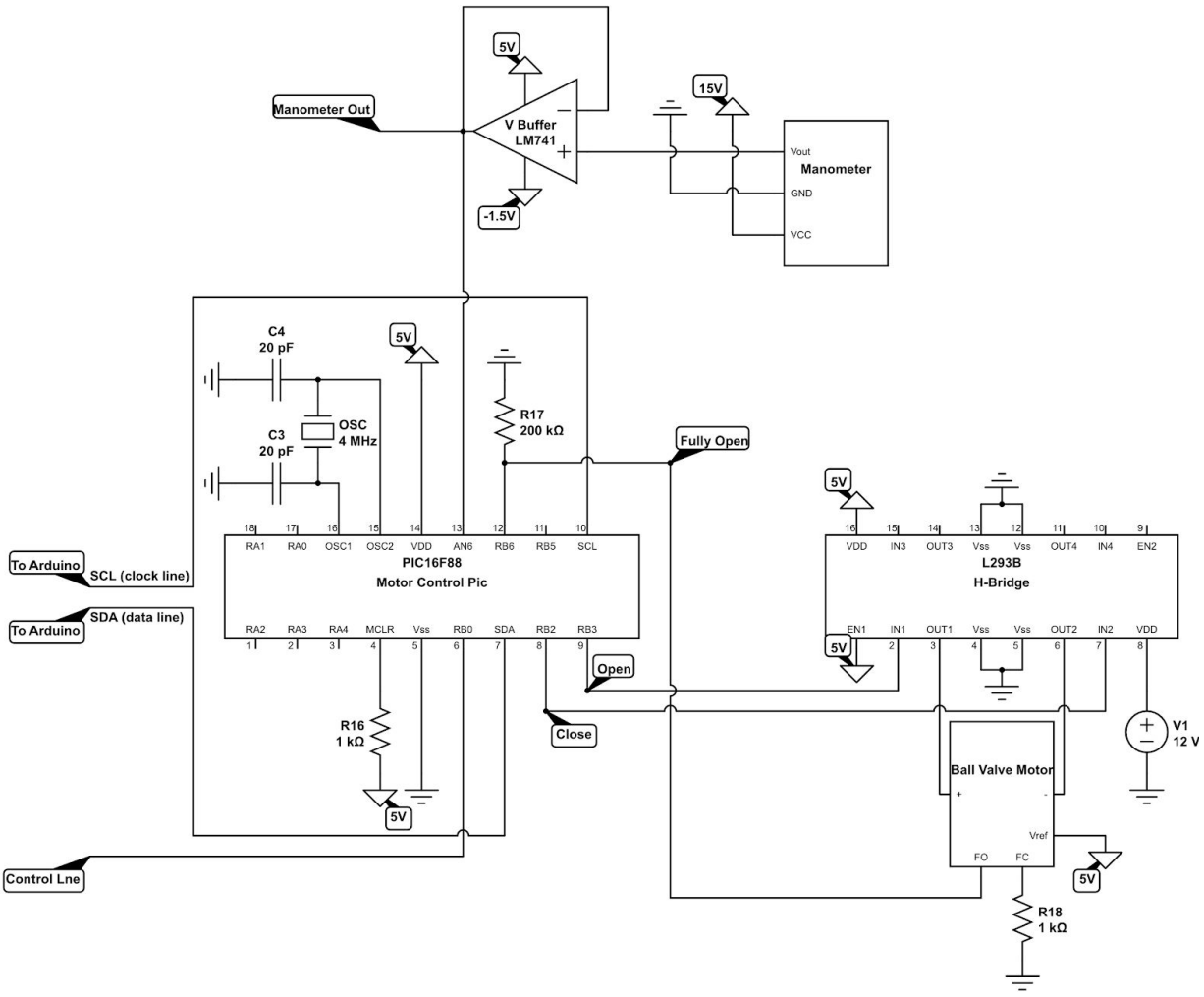


Figure C2. PIC16F88 wiring diagram

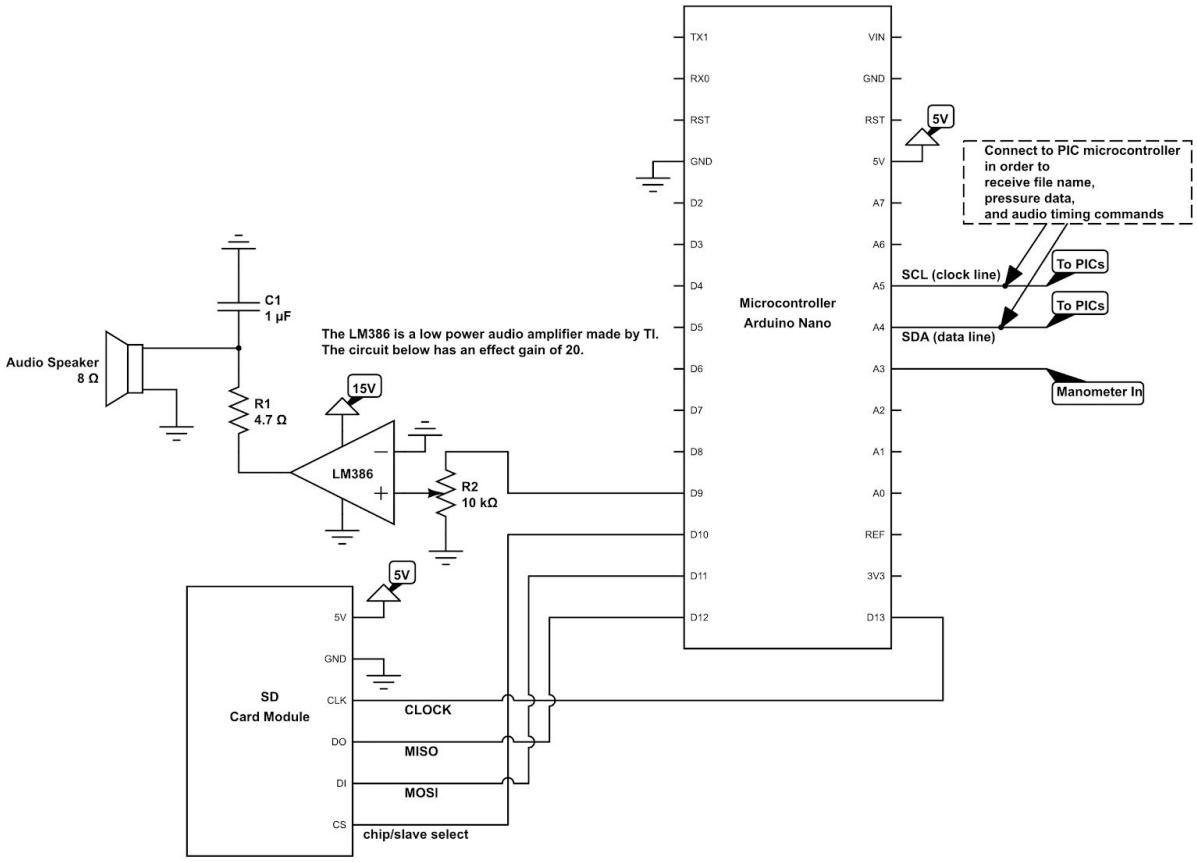


Figure C3. Arduino wiring diagram

D. Flowchart

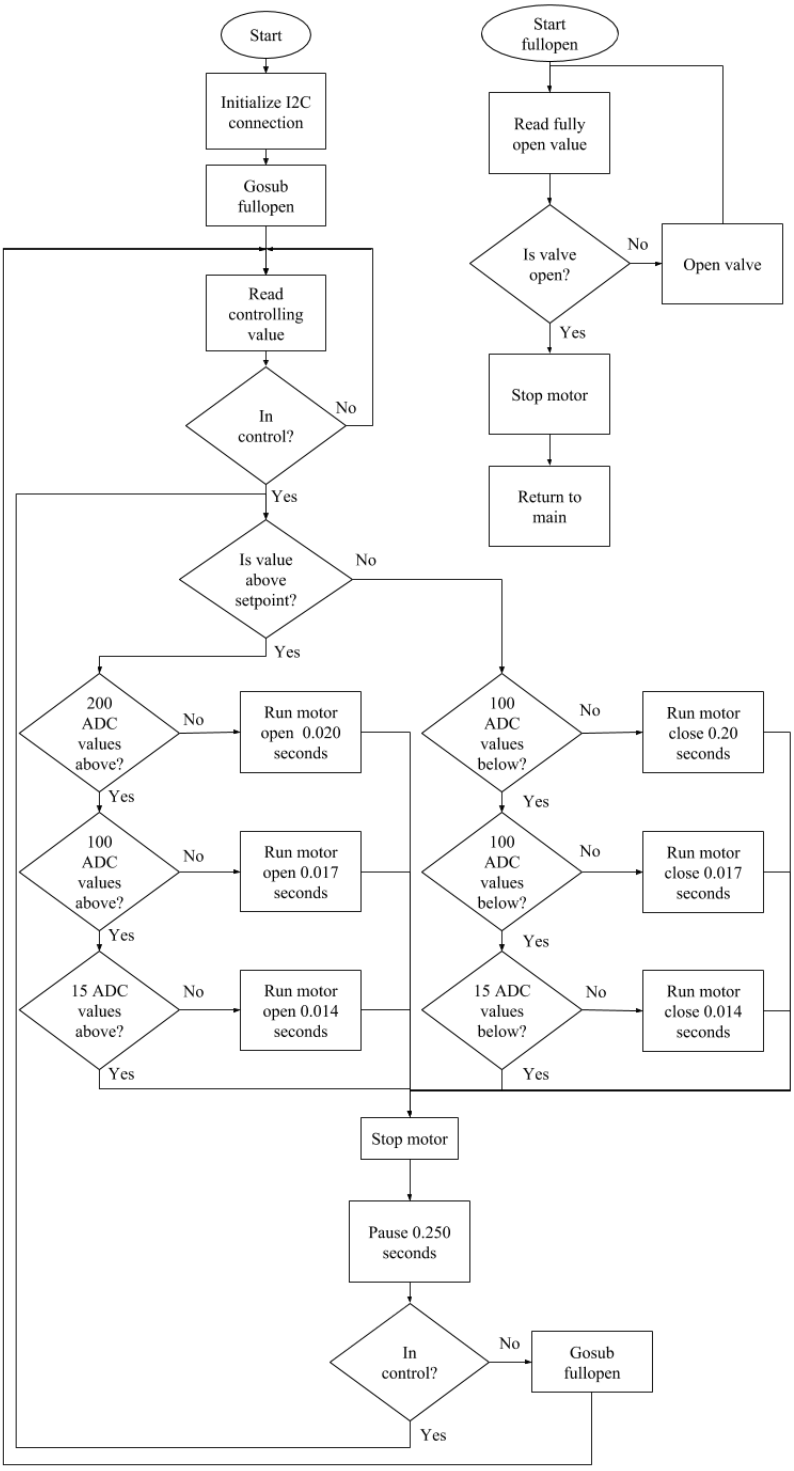


Figure D1. PIC16F88 flowchart

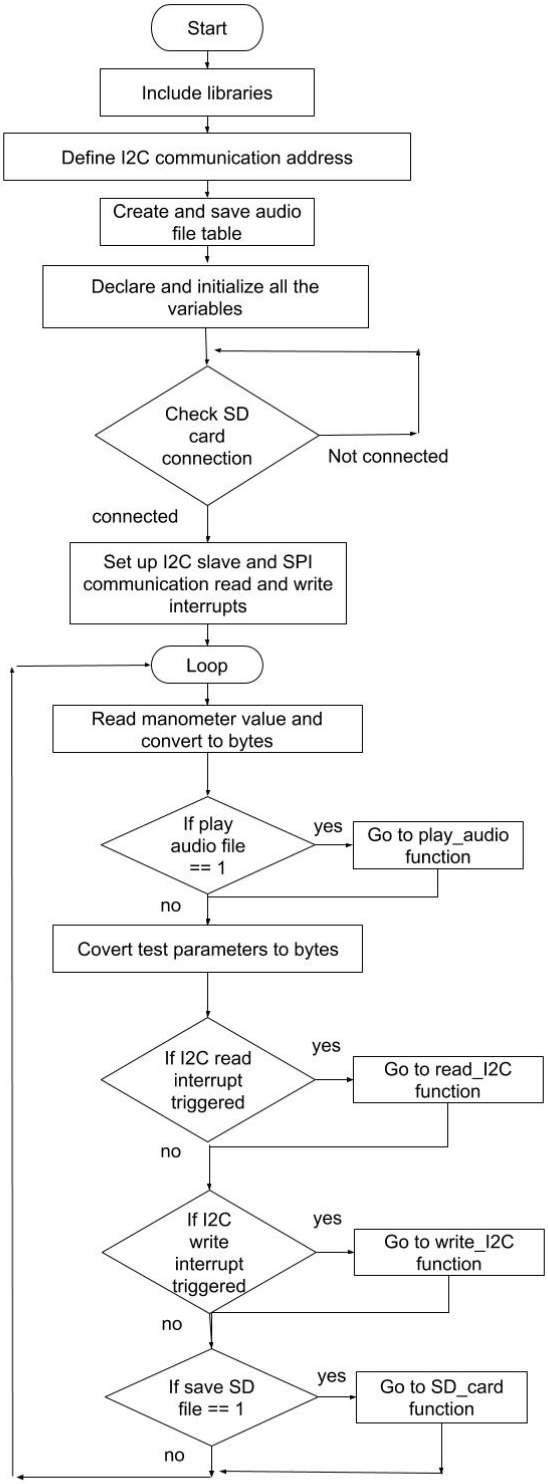


Figure D2. Arduino main flowchart pt. 1

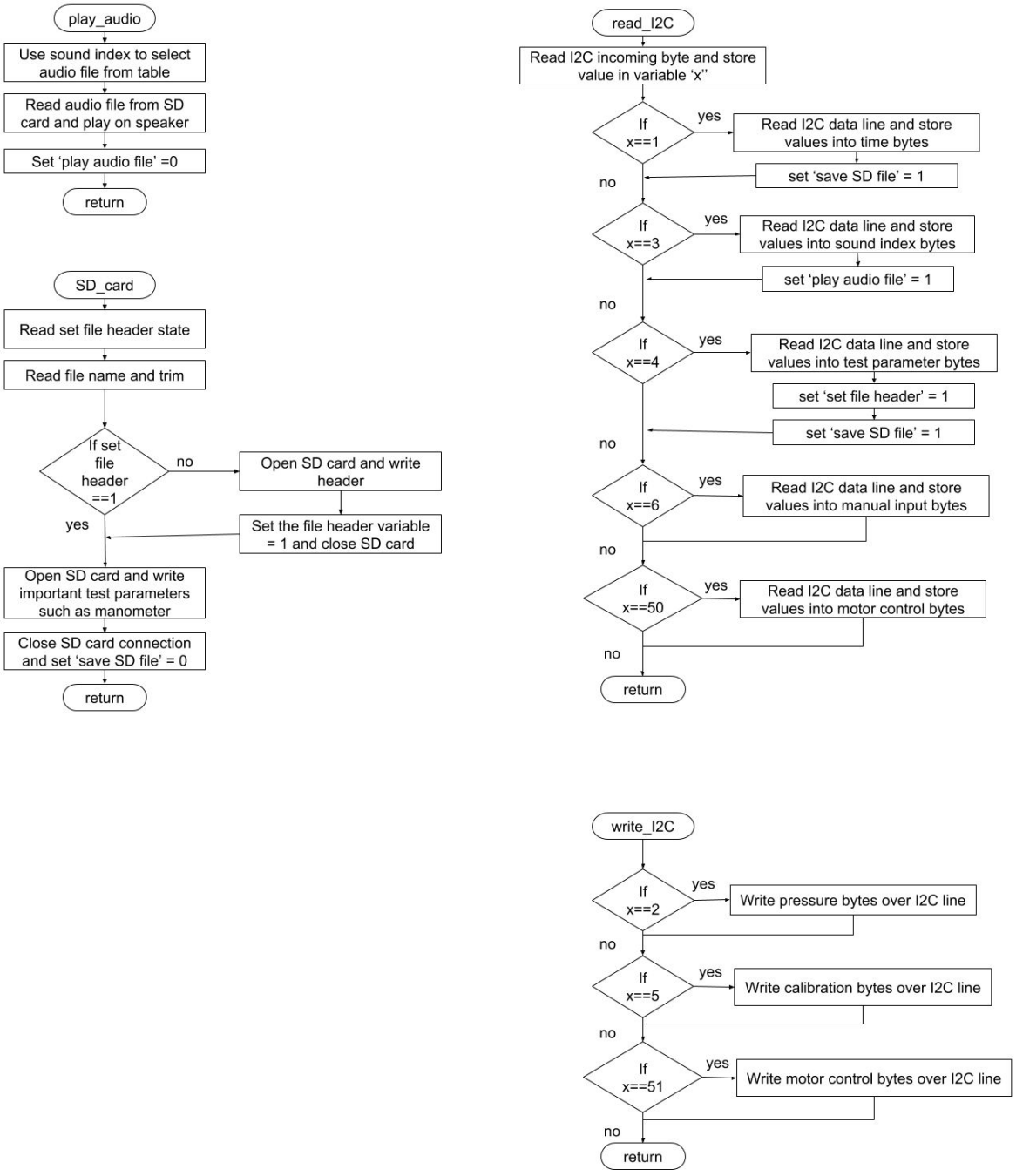


Figure D3. Arduino flowchart for I2C interrupts and sub SPI functions pt. 2

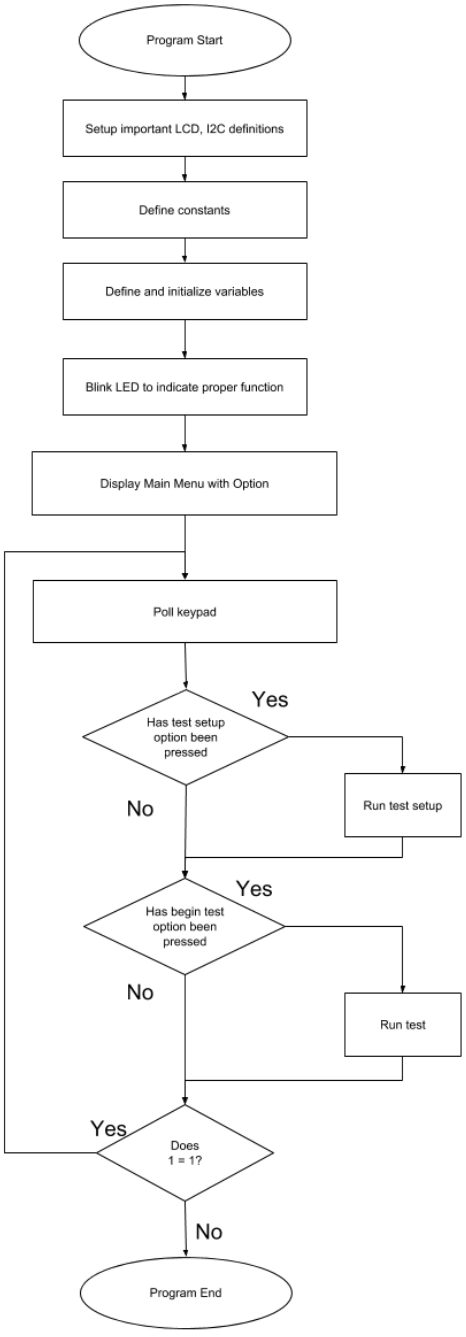


Figure D4. PIC18F2550 main program flowchart

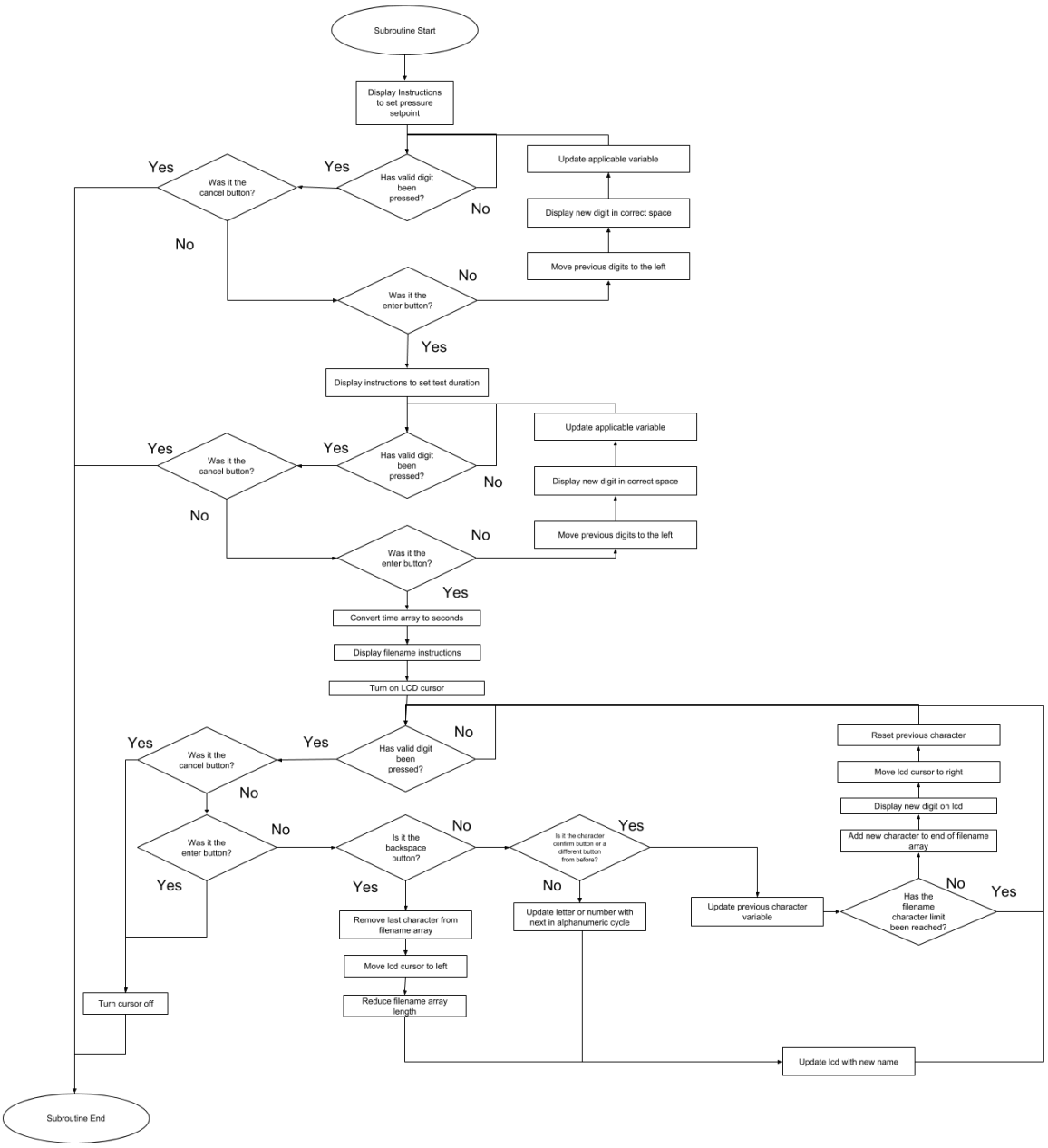


Figure D5. PIC18F2550 test setup flowchart

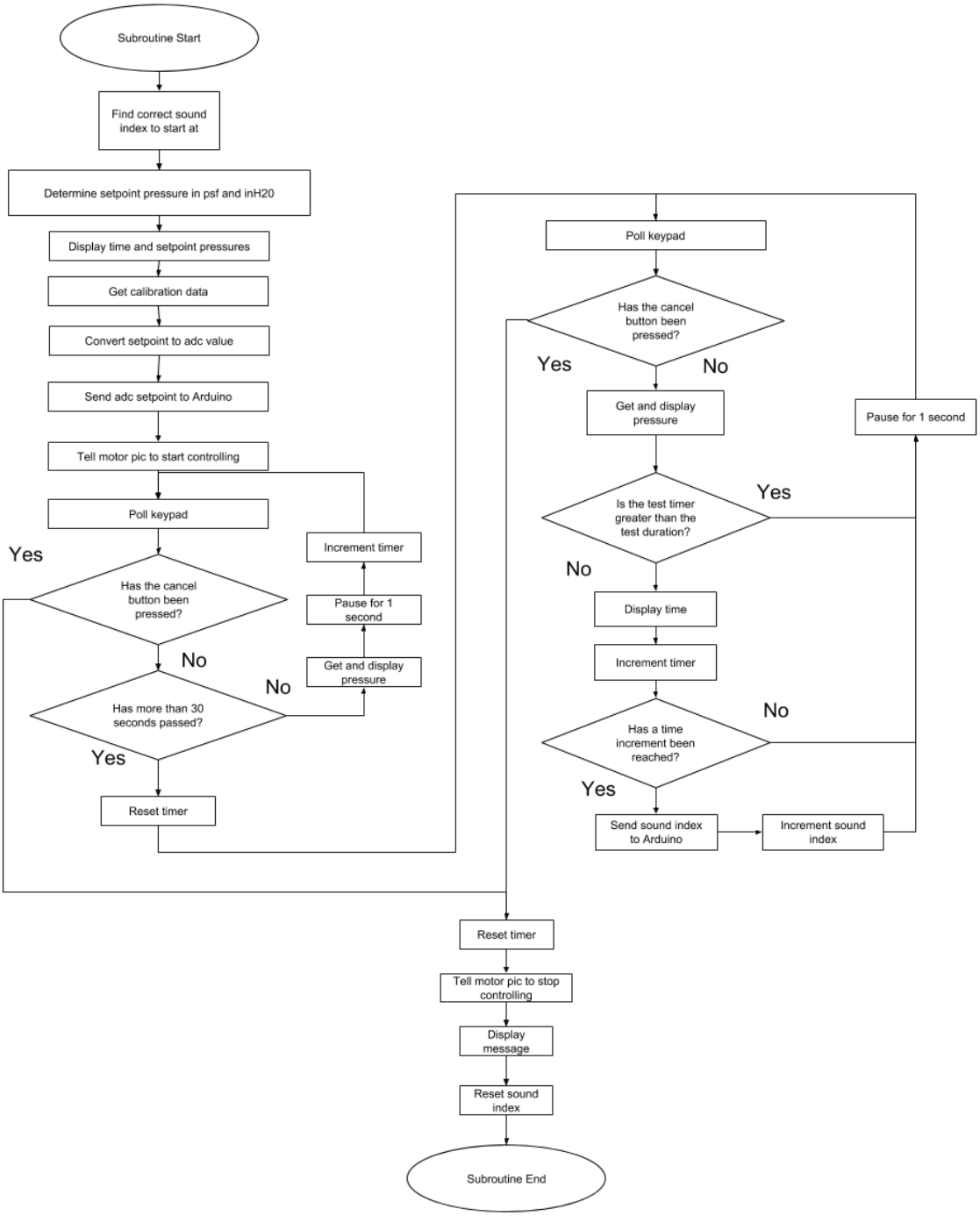


Figure D6. PIC18F2550 run test flowchart

E. Project Code

1. Arduino nano main code

```
/*
Code below reads Pressure using analog port 0. For Arduino microcontroller
Mechatronics 2018
*/
//Logs pressure data and reads voice data
#include <SPI.h>  //SD card serial communication
#include <SD.h>   //For talking to SD Card
#include <pcmConfig.h>
#include <pcmRF.h>
#include <TMRpcm.h>
#include <avr/pgmspace.h>
#include <Wire.h>
#include <String.h>

//Define pins
#define SLAVE_ADDRESS 0x08 //slave address, any number between 0x01 to 0x7F

//SD Card is on Standard SPI Pins
/*SD card SPI pin assignments
  MOSI = PIN 11 (module pin DI)
  MISO = PIN 12 (module pin DO)
  CLK = PIN 13  (module pin CLK)
  CS_PIN = CS pin on module
*/

/*This the I2C communication connections between Arduino and
 * PIC with the arduino as the slave device.
 * A4 = SDA
 * A5 = SCL
 * shift PIC register from 8 bit to 7 bit
 */

const int CS_PIN      =10; //SS for SD Shield
int pressure          =A3; //Analog Input 3 for pressure from manometer
int pressure_val      =0;  //used to store analog voltage value from manometer
int sound_index_array = 0;  //used to store PIC index value to select .WAV file from table
int slope             =167;   //hold calibration slope parameter
int intercept         =1;   //hold calibration intercept parameter
int SD_flag           =0;   //flag for sd card save
int sound             =0;    //flag sound play function
char c                =0;
unsigned long last_sound_time = 0;      //is set when a sound file is played


TMRpcm wav;   // create an object for use in this sketch
```

```
//set up audio file tables for time testing
static const char wav_0[] PROGMEM = "20.wav";       //20 minute audio file
static const char wav_1[] PROGMEM = "15.wav";       //15 minute audio file
static const char wav_2[] PROGMEM = "10.wav";       //10 minute audio file
static const char wav_3[] PROGMEM = "5.wav";        //5 minute audio file
static const char wav_4[] PROGMEM = "1.wav";        //1 minute audio file
static const char wav_5[] PROGMEM = "45.wav";       //45 second audio file
static const char wav_6[] PROGMEM = "30.wav";       //30 second audio file
static const char wav_7[] PROGMEM = "155.wav";      //15 second audio file
static const char wav_8[] PROGMEM = "55.wav";       //5 second audio file
static const char wav_9[] PROGMEM = "4.wav";       //4 second audio file
static const char wav_10[] PROGMEM = "3.wav";       //3 second audio file
static const char wav_11[] PROGMEM = "2.wav";       //2 second audio file
static const char wav_12[] PROGMEM = "11.wav";      //1 second audio file
static const char wav_13[] PROGMEM = "100.wav";     //end of test audio file

//create .WAV file table
const char *wav_table[] =
{
  wav_0,
  wav_1,
  wav_2,
  wav_3,
  wav_4,
  wav_5,
  wav_6,
  wav_7,
  wav_8,
  wav_9,
  wav_10,
  wav_11,
  wav_12,
  wav_13
};

//Initialize pressure and SD Card variables
int update_time =0;
int set_sd = 0;
int run_time = 0;  //returns the run time since the board has been powered on
int sd_reset = 0;   //used to reset excel header function once every time new parameters sent from PIC
int test_setpoint = 0;    //used to store the user test pressure set point on SD card
int test_duration = 0;    //used to store the user test duration on SD card
int HM_setpoint_p = 0; //used to store the human set pressure
String File_name_true = "";

//initializer I2C variables
    byte x = 0;  //stores incoming non character bytes from PIC
    //int x2 = 0;   //store incoming byte transfer into integer
```

```
//char c = 0;  //stores incoming characters from PIC

//pressure bytes
   byte pressure_byte1 = 0;    //used to split manometer voltage into two bytes
   byte pressure_byte2 = 0;
//time bytes
   byte time_byte1 = 0;        //used to store time from PIC and save into SD card
   byte time_byte2 = 0;
//sound index byte
   byte sound_index = 0;       //used to select audio file form table
//Test parameters send to the Arduino from PIC
   byte tst_setpoint_byte1 = 0;
   byte tst_setpoint_byte2 = 0;

   byte tst_duration_byte1 = 0;
   byte tst_duration_byte2 = 0;
  //strings used to create one string from 20 filename bytes received
   String File_name = "                    ";
//Calibration bytes that will be Sent from Arduino to PIC
   byte slope_byte1 = 0;
   byte slope_byte2 = 0;

   byte intcp_byte1 = 0;
   byte intcp_byte2 = 0;
//Human input for pressure from PIC to Arduino
   byte hm_set_byte1 = 0;
   byte hm_set_byte2 = 0;
//motor control from PIC adc to Arduino
   byte motor_cn_byte1 = 0;
   byte motor_cn_byte2 = 0;
   int count_send = 0; //to know how many byte have been sent from Arduino to PIC
   int count_receive = 0; //to know how many bytes have been received from PIC to Arduino

void setup()
{
 Serial.begin(9600);

 //SD card SPI library setup
 Serial.println(F("Initializing Card"));
 //Initialize SD card
 if (!SD.begin(CS_PIN))
 {
  Serial.println(F("Card Failure"));
  return;
 }
 Serial.println(F("Card Ready"));

 //I2C communication setup with interrupt functions
```

```
  Wire.begin(SLAVE_ADDRESS);              // join i2c bus with address #8
  Wire.onReceive(receiveEvent);           // register event
  Wire.onRequest(requestEvent);           //master PIC triggers an event to send data

  //speaker connection for TMRpcm library setup
  wav.speakerPin = 9; //5,6,11 or 46 on Mega, 9 on Uno, Nano, etc
  //notone
  //reset ADC voltage to 5V for real time clock inputs
  analogReference(DEFAULT);
  delay(2000);

  //CS pin, and pwr/gnd pins are outputs
  //pinMode(CS_PIN,   OUTPUT);
  pinMode (pressure, INPUT);

}

void loop()
{
  //Manometer reading and I2C preparations
  //Serial.println("im in the general loop");
  pressure_val = analogRead(pressure);      //store manometer voltage
  pressure_byte1 = pressure_val/256;        //acquire first byte from voltage reading
  pressure_byte2 = pressure_val % 256;      //store modulo (remainder) in second byte

  if (sound == 1)
  {
    last_sound_time = millis();
    play_audio();   //play indexed audio .WAV file
    sound=0;
  }

  /*if ((SD_flag  == 1) && (millis() - last_sound_time) > 3000)
  {
   Serial.println("SAVING TO SD CARD");
   sd_card();      //go to the sd card writing function
   SD_flag = 0;      //reset flag
  }*/

  HM_setpoint_p = (hm_set_byte1*256)+hm_set_byte2;
  test_setpoint = (tst_setpoint_byte1*256)+tst_setpoint_byte2;   //recombines test setpoint bytes into integer
  test_duration = (tst_duration_byte1*256)+tst_duration_byte2;   //recombines test duration bytes into
integer

}
//_____________________________________
//I2C communication code
  // function that executes whenever data is received from master
```

```
// this function is registered as an event, see setup()
  void receiveEvent(byte howMany) {
    Serial.println("I received an event loop");
    Serial.println(x);

     if (x == 1)              //time bytes are being sent to Arduino COMMAND CODE = 1
     {
      if (count_receive == 0)   //receive first time byte
      {
       Serial.println("receiving time bytes");
       time_byte1 = Wire.read();
       count_receive=count_receive+1;    //increment byte receiving counter
      }
      else{                    //receive second time byte
       time_byte2 = Wire.read();
       count_receive = 0;   //reset counter
       SD_flag = 1;      //save a new pressure value based on time measurement
       x = 0;    //reset global integer byte variable
       //convert time bytes into a single integer
       run_time = (time_byte1*256)+time_byte2;
      }

     }

     else if (x == 3)             //sound index is being sent to the Arduino COMMAND CODE = 3
     {
      Serial.println("receiving sound index bytes");
      sound_index = Wire.read();
      sound_index_array = (sound_index);      //set sound index byte to an integer value
      Serial.println("sound index");
      Serial.println(sound_index_array);
      x = 0;    //reset global integer byte variable
      sound = 1;
     }

     else if (x == 4)             //test parameter data is being sent to the Arduino COMMAND CODE = 4
     {
      Serial.println("Receiving test parameters from PIC");
      if (sd_reset == 0)
      {
       set_sd = 0;     //reset the excel header function because new test parameters being sent
       sd_reset = sd_reset+1;    //increment header reset only once every time
      }

     //begin byte collection for new parameters
      if (count_receive == 0)
      {
       tst_setpoint_byte1 = Wire.read();   //read test pressure set point bytes
```

```
  count_receive=count_receive+1;    //increment byte receiving counter
 }
 else if (count_receive == 1)
 {
  tst_setpoint_byte2 = Wire.read();
  count_receive=count_receive+1;
 }
 else if (count_receive == 2)
 {
  tst_duration_byte1 = Wire.read();   //read test duration time bytes
  count_receive=count_receive+1;
 }
 else if (count_receive == 3)
 {
  tst_duration_byte2 = Wire.read();
  count_receive=count_receive+1;
 }

 else if ((count_receive >= 4) && (count_receive < 25))  //read all file name bytes and store in string
 {
  File_name[count_receive-4] = Wire.read();
  //Serial.println(File_name);

  count_receive = count_receive+1;
 }

 else{

 }
 SD_flag =1;   //triggers the SD card saving function
}

else if (x == 6)            //Human input pressure being sent to the Arduino COMMAND CODE = 6
{
 if (count_receive == 0)   //receive first time byte
 {
  Serial.println("receiving human input bytes");
  hm_set_byte1 = Wire.read();
  count_receive=count_receive+1;    //increment byte receiving counter
 }
 else{                  //receive second time byte
  hm_set_byte2 = Wire.read();
  count_receive = 0;   //reset counter

 //convert time bytes into a single integer
 HM_setpoint_p = (hm_set_byte1*256)+hm_set_byte2;
 x = 0;    //reset global integer byte variable
 }
```

```
    }
  //********* motor control case **********
    else if (x == 50)
    {
     if (count_receive == 0)   //receive first time byte
     {
      Serial.println("receiving the motor control bytes");
      motor_cn_byte1 = Wire.read();
      count_receive=count_receive+1;    //increment byte receiving counter
     }
     else{                    //receive second motor control byte
      motor_cn_byte2 = Wire.read();
      count_receive = 0;   //reset counter
      Serial.println("motor bytes 1 then 2");
      Serial.println(motor_cn_byte1);
      Serial.println(motor_cn_byte2);

      x = 0;    //reset global integer byte variable
     }
    }

    else
    {
     x = Wire.read();                // receive byte as an integer
     Serial.println(x);              // print the integer
     count_receive = 0;               //reset the counter for all receive events
    }
  }

//function that executes whenever data is requested from the master
//this function is registered when master wants Arduino to send data
  void requestEvent(){
   Serial.println("sending data to PIC request loop");
   Serial.println(x);
   //manometer data send if COMMAND CODE =2
   if (x == 2) {
     if (count_send == 0) {
      Wire.write(pressure_byte1);
      count_send=count_send+1;
     } else {
      Wire.write(pressure_byte2);
      count_send = 0;   //reset counter

      Serial.println(pressure_byte1);
      Serial.println(pressure_byte2);
      x = 0;    //reset x
     }
   } else if (x == 5) { //Calibration data transfer to PIC COMMAND CODE = 5
```

```
      Serial.println("sending Calibration data");
      slope_byte1 = slope/256;      //convert integer slope value into bytes
      slope_byte2 = slope % 256;

      intcp_byte1 = intercept/256;    //convert integer intercept value into bytes
      intcp_byte2 = intercept % 256;

      if (count_send == 0) {
        Wire.write(slope_byte1);
        count_send=count_send+1;
      } else if (count_send == 1) {
       Wire.write(slope_byte2);
       count_send=count_send+1;
      } else if (count_send == 2) {
       Wire.write(intcp_byte1);
       count_send=count_send+1;
      } else {
       Wire.write(intcp_byte2);
       count_send= 0;    //reset counter
       x = 0;            //reset x
       Serial.println("calibration data:");
       Serial.println(slope_byte1);
       Serial.println(slope_byte2);
       Serial.println(intcp_byte1);
       Serial.println(intcp_byte2);
      }
    } else if (x == 51) {
       Serial.println("sending Motor control data");
       if (count_send == 0) {
         Wire.write(motor_cn_byte1); //write first byte to motor PIC
         count_send=count_send+1;
       }
       else {
         Wire.write(motor_cn_byte2); //write second byte to motor PIC
         count_send = 0;   //reset counter
         Serial.println("sent motor data");
         Serial.println(motor_cn_byte1);
         Serial.println(motor_cn_byte2);
       }
    } else { //if no data can be sent to PIC
      Serial.println("im not in");
      x = 0; //reset x
      bogus();  //go to error function used for debugging
    }
  }

//_____________________________________________________
```

```
//Function called when needing to save data to SD card
void sd_card(){
  if(Serial.available()){
 if(sound == 0){

    //start file header__perform only once per run
    if (set_sd == 0){
     //Serial.println(File_name.length());
      File_name.trim();           //trims off unused char spaces
      File_name_true = String(File_name +".csv");
     //Serial.println(File_name.length());
     //Write Column Headers
     File dataFile = SD.open((File_name_true), FILE_WRITE);  //File_name
     if (dataFile)
      {
      dataFile.println(F("\nNew Log Started!"));
      dataFile.println(F("Run Time,Pressure (mm H20), human setpoint pressure, test setpoint, test
duration, Active"));
      dataFile.close(); //Data isn't actually written until we close the connection!

      //Print same thing to the screen for debugging
      Serial.println(F("\nNew Log Started!"));
      Serial.println(F("Run Time,Pressure (mm H20),human setpoint pressure, test setpoint, test duration,
Active"));
      }
     else
      {
      Serial.println(F("Couldn't open log file"));
      }
     set_sd = set_sd +1;    //reset variable that determines if file header has been written to the SD card
    }

   //Open a file and write to it.
     File dataFile = SD.open((File_name_true), FILE_WRITE);
      if (dataFile)
       {
       dataFile.print(time_byte1);
       dataFile.print(time_byte2);
       dataFile.print(F(","));
       dataFile.print(pressure_val);  //saves voltage reading from manometer to SD card
       dataFile.print(F(","));
       dataFile.print(HM_setpoint_p);
       dataFile.print(F(","));
       dataFile.print(test_setpoint);
       dataFile.print(F(","));
       dataFile.print(test_duration);
       dataFile.print(F(","));
       dataFile.println("active");
```

```
        dataFile.close(); //Data isn't actually written until we close the connection!

        //Print same thing to the screen for debugging
        Serial.print(time_byte1);
        Serial.print(time_byte2);
        Serial.print(F(","));
        Serial.print(pressure_val);  //prints voltage reading from manometer to serial monitor
        Serial.print(F(","));
        Serial.print(HM_setpoint_p);
        Serial.print(F(","));
        Serial.print(test_setpoint);
        Serial.print(F(","));
        Serial.print(test_duration);
        Serial.print(F(","));
        Serial.println("active");
        }
      else {
        Serial.println(F("Couldn't open log file"));
        }
      SD_flag = 0;      //reset SD_card flag
      }
  }
}
//_____________________________________________________________

void bogus()  //function used for error's in code execution
  {
    Serial.println("did not send data or receive data or connect to SD card");
  }
//_____________________________________________________________

void play_audio()
  {
    //if(Serial.available()){
    Serial.println("I AM IN PLAYING AUDIO");
    Serial.println("sound index");
    Serial.println(sound_index_array);

     char wavFile[sound_index_array];
     //based on sound index sent over I2C, will play a certain .WAV file
     switch(sound_index_array){
       case 0 : strcpy_P(wavFile, wav_table[0]);
              wav.play(wavFile);
              break;
       case 1 : strcpy_P(wavFile, wav_table[1]);
              wav.play(wavFile);
              break;
       case 2 : strcpy_P(wavFile, wav_table[2]);
```

```
            wav.play(wavFile);
            break;
        case 3 : strcpy_P(wavFile, wav_table[3]);
            wav.play(wavFile);
            break;
        case 4 : strcpy_P(wavFile, wav_table[4]);
            wav.play(wavFile);
            break;
        case 5 : strcpy_P(wavFile, wav_table[5]);
            wav.play(wavFile);
            break;
        case 6 : strcpy_P(wavFile, wav_table[6]);
            wav.play(wavFile);
            break;
        case 7 : strcpy_P(wavFile, wav_table[7]);
            wav.play(wavFile);
            break;
        case 8 : strcpy_P(wavFile, wav_table[8]);
            wav.play(wavFile);
            break;
        case 9 : strcpy_P(wavFile, wav_table[9]);
            wav.play(wavFile);
            break;
        case 10 : strcpy_P(wavFile, wav_table[10]);
            wav.play(wavFile);
            break;
        case 11 : strcpy_P(wavFile, wav_table[11]);
            wav.play(wavFile);
            break;
        case 12 : strcpy_P(wavFile, wav_table[12]);
            wav.play(wavFile);
            break;
        case 13 : strcpy_P(wavFile, wav_table[13]);
            wav.play(wavFile);
            //set_sd = 0;         //reset the excel file header function for future run
            break;

      //}

    }
}
```

Arduino nano TMPpcm.h library configuration. Code is used in the pcmConfig.h library inside the main Arduino pressure regulation code:

```
*
This library was intended to be a simple and user friendly wav audio player using standard
Arduino libraries, playing bare-bones standard format WAV files.
Many of the extra features have been added due to user request, and are enabled
optionally only to preserve the out of the box simplicity and performance initially
intended.
Code/Updates: https://github.com/TMRh20/TMRpcm
Wiki: https://github.com/TMRh20/TMRpcm/wiki
Blog: https://tmrh20.blogspot.com/
*/
#ifndef pcmConfig_h   // if x.h hasn't been included yet...
#define pcmConfig_h   //   #define this so the compiler knows it has been included

#include <Arduino.h>
/****************** GENERAL USER DEFINES ******************************
 See https://github.com/TMRh20/TMRpcm/wiki for info on usage
  Override the default size of the buffers (MAX 254). There are 2 buffers, so memory usage will be double this number
   Defaults to 64 bytes for Uno etc. 254 for Mega etc. note: In multi mode there are 4 buffers*/
//#define buffSize 128  //must be an even number

 /* Uncomment to run the SD card at full speed (half speed is default for standard SD lib)*/
#define SD_FULLSPEED

 /* HANDLE_TAGS - This options allows proper playback of WAV files with embedded metadata*/
 #define HANDLE_TAGS

 /*Ethernet shield support etc. The library outputs on both timer pins, 9 and 10 on Uno by default. Uncommenting this
  will disable output on the 2nd timer pin and should allow it to function with shields etc that use Uno pin 10 (TIMER1 COMPB).*/
#define DISABLE_SPEAKER2

 /* Use 8-bit TIMER2 - If using an UNO, Nano, etc and need TIMER1 for other things*/
//#define USE_TIMER2

//#define debug
```

2. PIC16F88 Control Code:

```
'*****************************************************************
'*  Name    : ControlMaster.pbp                                  *
'*  Author  : Groupie Dimes                                      *
'*  Notice  : Copyright (c) 2018 [select VIEW...EDITOR OPTIONS]  *
'*          : All Rights Reserved                                *
'*  Date    : 3/28/2018                                          *
'*  Version : 1.0                                                *
'*  Notes   :                                                    *
'*          :                                                    *
'*****************************************************************
 ' Motor Wire Colors
 'Yellow/Orange - open
 'Blue - close
 'Green - OLS
 'Red - CLS
 'Black - ref (put at 5V)
 'PortB2 - Close
 'PortB3 - Open

'Define ADCIN parameters
DEFINE  ADC_BITS        10      ' Set number of bits in result
DEFINE  ADC_CLOCK       3       ' Set clock source (3=rc)
DEFINE  ADC_SAMPLEUS    15      ' Set sampling time in uS

'--- set device  ----------------------------------------------------------

DEFINE I2C_HOLD 1

'-------------------------------------------------------------------------

'Set PORTA to all input
TRISA = %11111111
'Initialize the I/O pins (RB2 & RB3 outputs and rest as inputs)
TRISB = %11110011

'Set up ADCON1
ADCON1 = %10000000          ' Right-justify results (lowest 10 bits)
ADCON0 = %00000001          ' PORTA .4 analog, rest pins to digital
```

```
ANSEL = 0
ANSEL.6 = 1
CMCON = $7

'--- Define here your variables and alias  ------------------------------------

setpoint    VAR WORD        'Setpoint input  result
pressure    var word        'Pressure input Variable
controlling var PORTB.0     'For control from master
led         var PORTA.2     'LED used for debugging
fullyopen   var PORTB.6     'Motor fully open
open        var PORTB.3     'Open motor output
close       var PORTB.2     'Close motor output
SDA         var PORTB.1     'I2C data line
SCL         var PORTB.4     'I2C clock
tmp         var byte        'Temporary byte for I2C

fullyopen = 0

low open
low close

'--- Define constants  ---------------------------------------------------------

I2CTmpaddress           CON 8 ' Make address = 8
I2Caddress              con I2CTmpaddress << 1

'--- Initialization Done!  -----------------------------------------------------

Main:
   'Wait for boot up
   pause 10
   low open
   low close
   high led
   pause 5000
   low led
   pause 5000
```

```
    'Fully Open at start
    gosub FullOpen
    pause 3000

    'Wait until control command from Master
    while (controlling != 1)
        high led
        pause 500
        low led
        pause 250
    wend

    'Start controlling when the command is received from the Master
    if (controlling == 1) then
        'I2C Communication
        gosub i2c
        'Begin controlling
        gosub controlloop
    endif
    goto main
end

controlloop:
'Control the pressure
while(controlling == 1)
    high led
    ADCIN 6, pressure

    'Open if P>Pset
    if (pressure > setpoint+6) then
    low  close
    high open
        if (pressure > setpoint+200) then
            pause 18
        elseif (pressure > setpoint+100) then
            pause 15
        elseif (pressure > setpoint+15) then
            pause 12
        else
```

```
            pause 10
        endif
    low open
    low close

    'Close if P<Pset
    elseif (pressure < setpoint-6) then
    low  open
    high close
    if (pressure < setpoint-200) then
            pause 20
        elseif (pressure < setpoint-100) then
            pause 17
        elseif (pressure < setpoint-15) then
            pause 14
        else
            pause 10
        endif
    low open
    low close

    'Don't move if within range
    else
        low open
        low close
    endif

    pause 250
wend
low led
'Fully open the valve
gosub FullOpen
pause 50
return


'Blink the LED 5 times ( for debugging)
blink:
high led
```

```
pause 50
low led
pause 50
high led
pause 50
low led
pause 50
high led
pause 50
low led
pause 50
high led
pause 50
low led
pause 50
high led
pause 50
low led
pause 50
return


'I2C Communication
i2c:
SSPCON.5 = 1
pause 25
I2Cwrite SDA, SCL, I2Caddress, [51], blink
pause 100
i2cread SDA, SCL, I2Caddress, [tmp], blink
setpoint = tmp << 8
pause 100
i2cread SDA, SCL, I2Caddress, [tmp], blink
setpoint = setpoint + tmp
SSPCON.5 = 0
low led
return

'Fully open the valve
FullOpen:
```

```
while(fullyopen < 1)
   high led
   high open
   low close
wend
if (fullyopen < 1) then
   goto FullOpen
else
   low led
   low open
   low close
   return
endif
```

3. PIC18F2550 User Interface Code

```
'****************************************************************
'*  Name    : PIC18F2550_UI_Code.BAS                            *
'*  Author  : Nathan Richter                                    *
'*  Notice  : Copyright (c) 2018 [select VIEW...EDITOR OPTIONS] *
'*          : All Rights Reserved                               *
'*  Date    : 3/24/2018                                         *
'*  Version : 1.0                                               *
'*  Notes   :                                                   *
'*          :                                                   *
'****************************************************************
'configure oscillator options---------------------------------------------
'set oscillator to internal-----------------------------------------------
#CONFIG
 CONFIG FOSC = INTOSC_EC
#ENDCONFIG

'set internal oscillator to 4 MHz-----------------------------------------
OSCCON.6 = 1
OSCCON.5 = 1
OSCCON.4 = 0


'start of definitions-----------------------------------------------------
'allows any slave to hold clock, essential for I2C communication----------
define I2C_HOLD 1

'definitions for LCD screen-----------------------------------------------
' Set LCD Data port
DEFINE LCD_DREG PORTA
```

```
' Set starting Data bit (0 or 4) if 4-bit bus
DEFINE LCD_DBIT 0
' Set LCD Register Select port
define LCD_RSREG    PORTB
' Set LCD Register Select bit
DEFINE LCD_RSBIT    3
' Set LCD Enable port
DEFINE LCD_EREG PORTB
' Set LCD Enable bit
define LCD_EBIT 2
' Set LCD bus size (4 or 8 bits)
DEFINE LCD_BITS 4
' Set number of lines on LCD
DEFINE LCD_LINES    2
' Set command delay time in us
DEFINE LCD_COMMANDUS    3000
' Set data delay time in us
DEFINE LCD_DATAUS  95


'start of constant declarations----------------------------------------------
'set Arduino address---------------------------------------------------------
ardaddresstmp con 8
'shifts address one bit left, essential for I2C communication to it----------
arduino_address con ardaddresstmp << 1

max_cycle_time con 60 'timeout for cycling to correct pressure


'start of port aliasing declarations-----------------------------------------
'keypad variables
col1 Var PORTB.7
col2 Var PORTB.6
col3 Var PORTA.5
col4 Var PORTB.4
row1 Var PORTC.6
row2 Var PORTC.5
row3 Var PORTC.4
row4 Var PORTC.2

led var PORTA.4    'status led

control_mot var PORTC.7 'motor control line

scl var PORTB.1  'i2c clock line, to portb.1 on 16f88
sda var PORTB.0  'i2c data line, to portb.4 on 16f88
```

```
'start of pic register configurations----------------------------------------
ADCON1 = %00001111 'configure all analog channels to digital channels

CVRCON.6 = 0 'turn off comparator

'usb configuration
UCON.3 = 0
UCFG.3 = 1

'set pins 0-output, 1-input
TRISA = %11100000
TRISB = %11101111
TRISC = %01111111
SSPCON1 = %00101000'set pic as an I2C master, with ports B1 and B0 as I2c ports
SSPSTAT.7 = 1  'slewrate disabled for standard speed transfer


'start of variable declarations----------------------------------------------
pressed_key_num var byte     'number of key in keypad array
pressed_key var byte         'variable to hold keypad character
previous_key var byte
available_keys var byte[65] 'array holding all possible keys

timeIntervals var word[14]  'array holding time values of alert times
timeIntervalIndex var byte 'index indicating which sound file should be
                    'played next

fullyOpen var bit 'ball valve is fully open

calibPressArray var byte[3]

'calibration curve m and b, sent from Arduino
calibration_slope var word
calibration_interc var word

'variables to hold pressure in decimaless form
testPressSetpoint_psf var byte[4] 'array holds numbers in setpoint entry

'variables that hold pressure (used to display to LCD)
testPressSet_psf var word
testPressSet_h20 var word
testPressSet_adc var word
currentPress_adc var word
currentPress_psf var word
currentPress_h20 var word
calibPress_h20 var word

'temporary word and byte, used dfor various purposes
```

```
tmp_byte var byte
tmp var word

testTime var byte[4] 'array holds test duration entry
testFileName var byte[20]'array hold filename entry
testFileNameLength var byte  'array keeps track of filename length
keypadDepth var byte  'depth of keypad, i.e. maximum number of possible letters/
                'numbers per key

test_timer_sec var word 'number of seconds since test start
test_duration_sec var word 'number of seconds in test

'loop counter variables
ct var byte
j var byte
i vaR BYTE


'start of variable initialization----------------------------------------------
timeIntervalIndex = 0
timeIntervals[0] = 1200
timeIntervals[1] = 900
timeIntervals[2] = 600
timeIntervals[3] = 300
timeIntervals[4] = 60
timeIntervals[5] = 45
timeIntervals[6] = 30
timeIntervals[7] = 15
timeIntervals[8] = 5
timeIntervals[9] = 4
timeIntervals[10] = 3
timeIntervals[11] = 2
timeIntervals[12] = 1
timeIntervals[13] = 0

pressed_key_num = 0
testFileNameLength = 0
previous_key = "%"
pressed_key = "%"
keypadDepth = -1
test_timer_sec = 0
test_duration_sec = 0
tmp = 0

J = 0
I = 0
CT = 0
```

```
FOR J = 0 TO 2
   calibPressArray[j] = " "
NEXT J

for j = 0 to 3
   testPressSetpoint_psf[j] = "0"
next j

for j = 0 to 20
   testFileName[j] = " "
next j

for j = 0 to 64
lookup j,["123A456B789C*0#D1adAgjmBptwC*0#D1beAhknBruxC*0#D1cfAiloBsvyC*0#D%"],i
   available_keys[j] = i
next j

i = 0

low control_mot

fullyOpen = 0

'Let LCD startup
   low led
   pause 500
   high led
   pause 500

gosub main_menu_disp 'display main menu

main:
gosub keypad  'go get number of key pressed
pressed_key = available_keys(pressed_key_num) 'get the character associated
                                    'with that number


   if(pressed_key == "A") then
      gosub test_setup 'setup test option
      pressed_key_num = 64
      ct = 0
      gosub main_menu_disp
   else
      if(pressed_key == "B") then
         gosub run_test 'run test option
         pressed_key_num = 64
         ct = 0
         gosub main_menu_disp
```

```
        'else
           'if(pressed_key == "C") then
              '  gosub calibration  'calibration option(not fully implemented)
              '  pressed_key_num = 64  'reset pressed key num
              '  ct = 0                'reset main loop counter (just in case)
              '  gosub main_menu_disp  'display menu again
           'else
              'if(pressed_key == "D") then
           '    gosub i2c_test 'test I2C communication (used for debugging I2C)
              '    pressed_key_num = 64
              '    ct = 0
              '    gosub main_menu_disp

    endif : endif': endif': endif
Goto main   ' Do it forever
End

main_menu_disp:
'display main menu text to lcd screen
    Lcdout $FE, 1, "A: Test Setup"
    lcdout $FE, $C0, "B: Start Test"
    'lcdout $FE, $94, "C: Calibrate"
    'lcdout $FE, $D4, "D: I2C Test "
return

'calibration:'not implemented fully
'display calibration information
'open valve steadily, recording voltage measurements for inputed pressures

'       Lcdout $FE, 1,   "Enter The Manometer "
'       lcdout $FE, $C0, "Pressure:  .   inH20"
'       lcdout $FE, $94, "Calibration Point # "
'       lcdout $FE, $D4, "*Cancel    #Confirm "

'       calibloop1:
'       gosub keypad 'poll for pressed keys
'       pressed_key = available_keys(pressed_key_num)
'       lcdout $FE, $94+19, dec (ct+1)
        'was a valid key pressed?
'       if((pressed_key != "A") and (pressed_key != "B") and
'          (pressed_key != "C") and (pressed_key != "D") and
'          (pressed_key != "%")) then

'           if(pressed_key == "*") then 'Was it the cancel button?
'              FOR J = 0 TO 4
'                 calibPressArray[j] = "0"
'              NEXT J
'              Lcdout $FE, 1, "Canceling"
```

```
'            lcdout $FE, $C0, "Calibration"
'            pause 3000
'            return

'         else
'            if(pressed_key == "#") then 'Was it the enter button?
'               calibPress_h20 = (calibPressArray[0]-48)*100 +
                '(calibPressArray[1]-48)*10 + (calibPressArray[2]-48)
'               gosub send_Calibration_Press_To_Arduino 'send manually
                                                        'entered manometer value
'               'tell slaved pic to increment ball valve
'                Lcdout $FE, 1, "Saving to SD card,"
'                lcdout $FE, $94, "please wait"
'                ct = ct +  1
'                FOR J = 0 TO 2
'                  calibPressArray[j] = "0"
'                NEXT J
'                pressed_key_num = 64
'                pause 500
'                'determine if valve is full open from slaved pic.
'                if(ct < 4) then
'                  fullyOpen = 0
'                else
'                  fullyOpen = 1
'                endif
'
'                if(fullyOpen) then'Is ball valve completely open?
'                  Lcdout $FE, 1, "Completing"
'                  lcdout $FE, $94, "Calibration"
'                  pause 4000
'                  return
'                else
'                  Lcdout $fe, 1,   "Enter The Manometer "
'                  lcdout $FE, $C0, "Pressure:  .   inH20"
'                  lcdout $FE, $94, "Calibration Point # "
'                  lcdout $FE, $D4, "*Cancel     #Confirm"
'                  goto calibloop1
'                endif
'            else
'               for j = 0 to 1
'                  i = j+1
'                  calibPressArray[j] = calibPressArray[i]'update array
                                                'with shifted digits
'               next j
'
'               calibPressArray[2] = pressed_key
'
'                lcdout $FE, $C0+10, calibPressArray[0] 'display entered
```

```
                    'digit onto screen with previous digits shifted
'
'                for j = 12 to 13
'                 i = j - 11
'                 lcdout $FE, $C0+j, calibPressArray[i] 'display entered
                        'digit onto screen with previous digits shifted
'                next j
'
'                pause 500
'                pressed_key_num = 64
'
'                goto calibloop1
'             endif
'          endif
'       else
'          goto calibloop1
'       endif
'return

test_setup: 'subroutine used to set the system up for testing----------
'display test setup instructions
'poll for valid key presses, changing setpoint and time

      Lcdout $fe, 1,   "Enter The Setpoint  "
      lcdout $FE, $C0, "Pressure For The    "
      lcdout $FE, $94, "Test in Psf:  .  Psf"
      lcdout $FE, $D4, "*Cancel     #Confirm"
      for j = 12 to 13
               i = j - 12
      lcdout $FE, $94+j, testPressSetpoint_psf[i] 'Display updated setpoint
      next j

      lcdout $FE, $94+(j+1), testPressSetpoint_psf[2] 'Display updated setpoint
      lcdout $FE, $94+(j+2), testPressSetpoint_psf[3] 'Display updated setpoint

  testSetPressLoop: 'loop to enter the setpoint pressure
       gosub keypad 'poll the keypad
       pressed_key = available_keys(pressed_key_num)
       if((pressed_key != "A") and (pressed_key != "B") and
         (pressed_key != "C") and (pressed_key != "D") and
         (pressed_key != "%")) then 'Has valid digit been pressed
         if(pressed_key == "*") then 'Was the cancel button pressed?
           return
         else
            if(pressed_key == "#") then 'was the enter button pressed?
               pressed_key_num = 64
               pause 500
               gosub test_setup_time 'go to time setup subroutine
```

```
            return
          else
            for j = 0 to 2
              i = j+1
              testPressSetpoint_psf[j] = testPressSetpoint_psf[i]
              'Update setpoint array with shifted digits
            next j

            testPressSetpoint_psf[3] = pressed_key 'insert new number

            for j = 12 to 13
             i = j - 12
             lcdout $FE, $94+j, testPressSetpoint_psf[i]
             'Display updated setpoint on lcd
            next j

            lcdout $FE, $94+(j+1), testPressSetpoint_psf[2]
            'Display updated setpoint on lcd
            lcdout $FE, $94+(j+2), testPressSetpoint_psf[3]
            'Display updated setpoint on lcd

            pause 500
            pressed_key_num = 64

            goto testSetPressLoop 'loop
          endif
        endif
      else
        goto testSetPressLoop 'loop
      endif
return


test_setup_time: 'subroutine to enter in the duration of the test
'display test setup instructions
'poll for valid key presses, changing setpoint and time

  Lcdout $fe, 1,   "Enter The Test      "
  lcdout $FE, $C0, "Duration in Minutes "
  lcdout $FE, $94, "And Seconds:   :    "
  lcdout $FE, $D4, "*Cancel     #Confirm"

  'update testTime with the previous test duration
  testTime[0] = (test_duration_sec/60)/10
  testTime[1] = (test_duration_sec/60)//10
  testTime[2] = (test_duration_sec//60)/10
  testTime[3] = (test_duration_sec//60)//10
```

```
testSetTimeLoop: 'loop to enter in new number
for j = 13 to 14
   i = j - 13
lcdout $FE, $94+j, dec testTime[i] 'Display updated time on lcd
next j

for j = 16 to 17
   i = j - 14
lcdout $FE, $94+j, dec testTime[i] 'Display updated time on lcd
next j

   gosub keypad 'poll the keypad
   pressed_key = available_keys(pressed_key_num)
   if((pressed_key != "A") and (pressed_key != "B") and
     (pressed_key != "C") and (pressed_key != "D") and
     (pressed_key != "%")) then 'Has valid digit been pressed
      if(pressed_key == "*") then 'Was the cancel button pressed?
        return
      else
         if(pressed_key == "#") then 'Was it the enter button?
            pressed_key_num = 64
            test_duration_sec = testTime[0]*10*60+testTime[1]*60+
            testTime[2]*10+testTime[3] 'calculate length of
                                  'the test in seconds
            test_timer_Sec = 0
            pause 500
            gosub test_setup_filename ' go to filename entry subroutine
            returN
         ELSE
            for j = 0 to 2
               i = j+1
               testTime[j] = testTime[i]
               'Update time array with shifted digits
            next j

            testTime[3] = pressed_key-48

            for j = 13 to 14
               i = j - 13
               lcdout $FE, $94+j, dec testTime[i]
               'Display updated time on lcd
            next j


            for j = 16 to 17
               i = j - 14
               lcdout $FE, $94+j, dec testTime[i]
               'Display updated time on lcd
```

```
                next j

                pause 500
                pressed_key_num = 64
                goto testSetTimeLoop
              endif
            ENDIF
        else
            goto testSetTimeLoop
        endif
return

test_setup_filename: 'subroutine to enter in filename using alphanumeric keypad
    'print filenaming insturctions
    Lcdout $fe, 1,   "Enter The Filename: "

    lcdout $FE, $94, "Cnext Chr Bbackspace"
    lcdout $FE, $D4, "*Cancel     #Confirm"

    lcdout $FE, $0E
    gosub update_lcd_wfilename 'go to subroutine to print out current filename

    nameloop: 'main loop to get new filename not needed in final version
  '     gosub keypad 'poll keypad
  '     pressed_key = available_keys(pressed_key_num)
  '
  '     if((pressed_key != "A") and (pressed_key != "D") and
          '(pressed_key != "%")) then'has a valid digit been pressed?
  '         if(pressed_key == "*")then'was it the cancel button?
  '             lcdout $FE, $0C 'turn off cursor
  '             return
  '         else
  '             if(pressed_key == "#") then'was it the enter button?
  '                 lcdout $FE, $0C 'turn off cursor
  '                 Lcdout $FE, 1, "Saving Filename"
  '                 pause 1000
  '                 Lcdout $FE, 1, "Filename Saved"
  '                 pause 1000
  '                 return
  '             else
  '                 if(pressed_key == "B") then 'Was it the backspace button?
  '                     testFileName[testFileNameLength] = " "
                      'remove last character from filename array
  '                     if(testFileNameLength > 0)then
  '                         testFileNameLength = testFileNameLength - 1
                          'reduce array length
  '                     endif
  '                         keypadDepth = -1 'reset keypad depth
```

```
  '                    pressed_key_num = 64
  '                    pause 500
  '                    gosub update_lcd_wfilename 'update lcd with current name
  '                    goto nameloop
  '                else
  '                   if(((pressed_key == "C") or (pressed_key != previous_key)) and previous_key != "%")then'Was it
the character confirm button or a different button from before?
  '                    if(testFileNameLEngth < 19) then 'is the length less than maximum?
  '                       testFileNameLength = testFileNameLength + 1 'increment filename length
  '                       if((pressed_key != previous_key) and (pressed_key != "C")) then
  '                          previous_key = pressed_key 'Update previous character variable
  '                          testFileName[testFileNameLength] = pressed_key 'Update letter or number with next in
alphanumeric cycle
  '                          gosub update_lcd_wfilename
  '                          keypadDepth = 0
  '                       else
  '                          previous_key = "%" 'reset previous key
  '                          keypadDepth = -1 'reset to initial layer
  '                       endif
  '                       pressed_key_num = 64
  '                     endif
  '                    pause 500
  '                    gosub update_lcd_wfilename 'display new character on lcd
  '                    goto nameloop
  '
  '                  else
  '                     previous_key = pressed_key
  '                     keypadDepth = (keypadDepth + 1) // 4 'Update keypad depth to next layer
  '                     testFileName[testFileNameLength] = available_keys(keypadDepth*16+pressed_key_num)'Update
letter or number with next in alphanumeric cycle
  '                     pressed_key_num = 64
  '                     pause 500
  '                     gosub update_lcd_wfilename
  '                     goto nameloop
  '                   endif
  '                endif
  '             endif
  '          endif
  '     else
  '        goto nameloop
  '     endif
return

update_lcd_wfilename: 'subroutine to update lcd with the new filename
   for j = 0 to 19
     lcdout $FE, $C0+j, testFileName[j]
   next j
   lcdout $FE, $C0+testFileNameLength
```

```
return

run_test: 'subroutine to run the test
'display test setpoints while system finds pressure
'ensure pressure setpoint remains in valid range while test is conducted.
      while(timeIntervals[timeIntervalIndex] >= test_duration_sec and timeIntervalIndex < 14)
      timeIntervalIndex = timeIntervalIndex + 1
      wend
      testPressSet_psf = (testPressSetpoint_psf[0]-48)*1000 + (testPressSetpoint_psf[1]-48)*100 +
(testPressSetpoint_psf[2]-48)*10 + (testPressSetpoint_psf[3]-48)
      testPressSet_h20 = testPressSet_psf * 10 / 52
      Lcdout $fe, 1,   "Timer   :    *Cancel"   'display test information to lcd
      lcdout $FE, $C0, "        Psf    inH20"
      lcdout $FE, $94, "Set:      .      .  "
      lcdout $FE, $D4, "Actual:   .      .  "

      gosub display_time
      gosub display_setpointPress
      gosub get_Calibration_Data_From_Arduino
      'gosub send_Test_Params_To_Arduino
      gosub pressSet_to_adc
      testPressSet_Adc = 500
      gosub send_Arduino_adc_Setpoint 'send setpoint to Arduino
      pause 500
      gosub start_PIC_Control
      ct = 0
      testloop:
         gosub keypad 'poll keypad for key
         pressed_key = available_keys(pressed_key_num)
         pressed_key_num = 63
         if(pressed_key == "*") then 'has the cancel button been pressed
               test_timer_sec = 0'reset the timer
               gosub stop_PIC_Control
               Lcdout $FE, 1,   "Regulation Canceled "   'display regulation ended message
               lcdout $FE, $94, "Returning to Menu   "
               pause 2000
               return 'return to main menu
         else
            'if(testPressSet_adc > currentPress_adc) then
            '   tmp = (testPRessSet_adc - currentPress_adc)
            'else
            '   tmp = ( currentPress_adc - testPRessSet_adc)
            'endif

            'if(tmp <= 30) then
            '  ct = ct + 1
            'else
            '  ct = 0
```

```
            'endif

            if((test_timer_sec > 30)) then'has the motor control pic cycled to within 1% of the setpoint?
               test_timer_sec = 0 'reset the timer
               gosub active_test
               gosub stop_PIC_Control
               Lcdout $FE, 1,   "Regulation Canceled "   'display regulation ended message
               lcdout $FE, $94, "Returning to Menu   "
               test_timer_sec = 0
               timeIntervalIndex = 0
               pause 2000
               return 'return to main menu
            else
               'if(test_timer_sec > max_cycle_time) then'has more than a minute passed?
               '    test_timer_sec = 0
               '    gosub stop_PIC_Control
               '    Lcdout $fe, 1,   "Failed to cycle to  "   'display test information to lcd
               '    lcdout $FE, $C0, "Correct Setpoint     "
               '    lcdout $FE, $94, "Pressure             "
               '    lcdout $FE, $D4, "Returning to Menu   "'display "failed to cycle to correct pressure"
               '    timeIntervalIndex = 0
               '    pause 2000
               '    return
              ' else
                   gosub get_Press_from_Arduino
                   lcdout $FE, $C0, "    "
                   lcdout $FE, $C0, dec testPressSet_Adc
                   lcdout $FE, $C0+5, "    "
                   lcdout $FE, $C0+5, dec currentPress_Adc
                   gosub adc_to_inH20_psf 'convert adc value to pressure in inH20 and psf
                   gosub display_currentPress 'print current pressure to lcd
                   pause 1000 'wait for a second
                   test_timer_sec = test_timer_sec + 1 'increment timer by one
                   goto testloop:
               'endif
            endif
         goto testloop
         endif
return

active_test:
  activetestloop:
    gosub keypad'poll keypad for pressed key
    pressed_key = available_keys(pressed_key_num)
    pressed_key_num = 63
    if(pressed_key == "*") then 'was the key the cancel button
       return 'Yes it was                          n
    else 'No it was not
```

```
      gosub get_Press_from_Arduino

      lcdout $FE, $C0, "    "
      lcdout $FE, $C0, dec testPressSet_Adc
      lcdout $FE, $C0+3, "    "
      lcdout $FE, $C0+3, dec currentPress_Adc
      gosub adc_to_inH20_psf 'convert adc value to pressure in inH20 and psf
      gosub display_currentPress 'print current pressure to lcd

      if((test_duration_sec - test_timer_Sec) > 0) then 'is the timer_duration-timer greater than 0?
         test_timer_sec = test_timer_Sec + 1 'increment timer by 1
         gosub display_time 'display remaining time to lcd
         if((test_duration_sec - test_timer_Sec) == timeIntervals[timeIntervalIndex]) then'has a sound increment been
reached yet?
            gosub send_Sound_Index_To_Arduino'Tell sound controller to play corresponding sound.
            timeIntervalIndex = timeIntervalIndex + 1
         endif
          'gosub send_Time_To_Arduino 'send time_duration-time to sound controller
      endif

      pause 1000 'wait 1 second
      goto activetestloop
   endif

return

display_time: 'subroutine to display remaining time on lcd
   lcdout $FE, 2  'move  cursor to second line
   tmp = test_duration_sec - test_timer_sec  'calc remaining time
   testTime[0] = (tmp/60)/10
   testTime[1] = (tmp/60)//10
   testTime[2] = (tmp//60)/10
   testTime[3] = (tmp//60)//10
   for j = 1 to 6  'shift cursor right six spaces
   lcdout $FE, $14
   next j

   lcdout dec testTime[0]
   for j = 1 to 3
   lcdout dec testTime[j]
    if j == 1 then
        lcdout $FE, $14 'move cursor right
    endif
   next j
return

display_setpointPress: 'subroutine to display setpoint pressure on lcd during test
  if ((testPressSet_psf/100) < 10) then
```

```
   lcdout $FE, $94+8, " "   'display setpoint in psf (integar)
   lcdout $FE, $94+9, dec TestPressSet_psf/100   'display setpoint in psf (integar)
  else
   lcdout $FE, $94+8, dec TestPressSet_psf/100   'display setpoint in psf (integar)
  endif

  if ((testPressSet_psf//100) < 10) then
   lcdout $FE, $94+11, dec 0
   lcdout $FE, $94+12, dec TestPressSet_psf//100 'display setpoint in psf (decimal)
  else
   lcdout $FE, $94+11, dec TestPressSet_psf//100 'display setpoint in psf (decimal)
  endif

   lcdout $FE, $94+16, dec TestPressSet_h20/100 'display setpoint in inH20 (integar)

  if ((testPressSet_h20//100) < 10) then
   lcdout $FE, $94+18, dec 0
   lcdout $FE, $94+19, dec TestPressSet_h20//100 'display setpoint in inH20 (decimal)
  else
   lcdout $FE, $94+18, dec TestPressSet_h20//100 'display setpoint in inH20 (decimal)
  endif
return

display_currentPress:
  if ((currentPress_psf/100) < 10) then
   lcdout $FE, $D4+8, " "   'display setpoint in psf (integar)
   lcdout $FE, $D4+9, dec currentPress_psf/100   'display currentPress in psf (integar)
  else
   lcdout $FE, $D4+8, dec currentPress_psf/100   'display currentPress in psf (integar)
  endif

  if ((currentPress_psf//100) < 10) then
   lcdout $FE, $D4+11, dec 0
   lcdout $FE, $D4+12, dec currentPress_psf//100 'display currentPress in psf (decimal)
  else
   lcdout $FE, $D4+11, dec currentPress_psf//100 'display currentPress in psf (decimal)
  endif


   lcdout $FE, $D4+16, dec currentPress_h20/100 'display currentPress in inH20 (integar)

  if ((currentPress_h20//100) < 10) then
   lcdout $FE, $D4+18, dec 0
   lcdout $FE, $D4+19, dec currentPress_h20//100 'display currentPress in inH20 (decimal)
  else
   lcdout $FE, $D4+18, dec currentPress_h20//100 'display currentPress in inH20 (decimal)
  endif
return
```

```
i2c_test: 'subroutine for testing I2C connection to Arduino/any slaved microcontroller
Lcdout $fe, 1,   "Testing I2C Connect-"   'display test information to lcd
lcdout $FE, $C0, "ion:                "
lcdout $FE, $94, "Send Ctrl # to Pic  "
lcdout $FE, $D4, "ADC:                "

  'i2cwrite sda, scl, slave1address, [6], blink
  'pause 10
  'testPressSet_psf = (testPressSetpoint_psf[0]-48)*1000 + (testPressSetpoint_psf[1]-48)*100 +
(testPressSetpoint_psf[2]-48)*10 + (testPressSetpoint_psf[3]-48)
  'testPressSet_psf = 1000
  'testPressSet_h20 = testPressSet_psf * 10 / 52
  'gosub get_Calibration_Data_From_Arduino
  'gosub pressSet_to_adc
  'gosub send_Arduino_adc_Setpoint
  'gosub start_PIC_Control
  gosub get_Press_from_Arduino
  lcdout $FE, $D4+5, dec currentPress_adc
  pause 4000
  'gosub stop_PIC_Control
  'i2cwrite sda, scl, arduino_address, [51], blink
  'pause 10
  'i2cread sda, scl, arduino_address, [tmp_byte], blink
  'pause 10
  'i2cread sda, scl, arduino_address, [tmp_byte], blink

  'i2cwrite sda, scl, slave1address, [testWrite], blinktwice
  'i2cread sda, scl, slave1address, test, blink
  'pause 10
  'i2cwrite sda, scl, arduino_address, ["2"], blink
  'i2cread sda, scl, arduino_address, [tmp_byte]
  'pause 10000
  'gosub stop_PIC_Control
  'for j = 0 to 20
  'gosub get_Press_from_Arduino
  'lcdout $FE, $D4+4, dec currentPress_Adc
  'pause 1000
  'gosub keypad'poll keypad for pressed key
  'pressed_key = available_keys(pressed_key_num)
  'pressed_key_num = 63
  'if pressed_key = "*" then
    'j = 20
  'endif
  'next j
  'pause 500
  'gosub stop_PIC_Control
return
```

```
adc_to_inH20_psf: 'convert from adc to inH20 and psf
   currentPress_h20 = currentPress_adc*50/calibration_slope*2
   currentPress_psf = currentPress_h20*52/10
return

get_Press_from_Arduino: 'subroutine to get the current pressure from the Arduino through I2C communication
   i2cwrite sda, scl, arduino_address, [2], blink
   pause 10
   i2cread sda, scl, arduino_address, [tmp_byte], blink 'receive most significant byte of information first
   currentPress_adc = tmp_byte << 8                    'shift byte left 8 spaces into a word sized variable
   pause 10
   i2cread sda, scl, arduino_address, [tmp_byte], blink 'receive second byte
   currentPRess_adc = currentPRess_Adc + tmp_byte       'add to word
   pause 10
return

send_Time_To_Arduino 'subroutine to send the current time to the Arduino through I2C communication
   i2cwrite sda, scl, arduino_address, [1], blink
   pause 10
   tmp_byte = (test_duration_sec - test_timer_Sec)>> 8
   i2cwrite sda, scl, arduino_address, [tmp_byte], blink
   pause 10
   tmp_byte = (test_duration_sec - test_timer_Sec)//256
   i2cwrite sda, scl, arduino_address, [tmp_byte], blink
   pause 10
return

send_Sound_Index_To_Arduino 'subroutine to send the sound index to the Arduino through I2C communication
when a sound increment is reached
   i2cwrite sda, scl, arduino_address, [3], blink
   pause 10
   i2cwrite sda, scl, arduino_address, [timeIntervalIndex], blink
   pause 10
return

send_Test_Params_To_Arduino 'subroutine to send the test parameters to the Arduino through I2C communication
(for data logging)
   i2cwrite sda, scl, arduino_address, [4], blink
   pause 10

   'send the test setpoint pressure in psf
   tmp_byte = testPressSet_psf >> 8
   i2cwrite sda, scl, arduino_address, [tmp_byte], blink
      pause 10
   tmp_byte = testPressSet_psf // 256
   i2cwrite sda, scl, arduino_address, [tmp_byte], blink
      pause 10
```

```
    'send the test duration in seconds
    tmp_byte = test_duration_sec >> 8
    i2cwrite sda, scl, arduino_address, [tmp_byte], blink
    tmp_byte = test_duration_sec // 256
       pause 10
    i2cwrite sda, scl, arduino_address, [tmp_byte], blink
       pause 10

    'send the filename
    for i = 0 to 19
       i2cwrite sda, scl, arduino_address, [testFileName[i]], blink
       pause 10
    next i
    pause 2000
return

get_Calibration_Data_From_Arduino 'subroutine to get the calibration curve info from the Arduino
    i2cwrite sda, scl, arduino_address, [5], blink
    pause 10

    'get slope of calibration curve
    i2cread sda, scl, arduino_address, [tmp_byte], blink
       pause 10
    calibration_slope = tmp_byte << 8

    i2cread sda, scl, arduino_address, [tmp_byte], blink
       pause 10
    calibration_slope = calibration_slope + tmp_byte

    'get intercept of calibration curve
    i2cread sda, scl, arduino_address, [tmp_byte], blink
       pause 10
    calibration_interc = tmp_byte << 8

    i2cread sda, scl, arduino_address, [tmp_byte], blink
       pause 10
    calibration_interc = calibration_interc + tmp_byte
return


send_Calibration_Press_To_Arduino 'subroutine to send calibration information to Arduino (not fully implemented
in project)
    i2cwrite sda, scl, arduino_address, [6], blink
    pause 10

    'return the calibration pressure in h20
    tmp_byte = calibPress_h20 >> 8
```

```
    i2cwrite sda, scl, arduino_address, [tmp_byte], blink
       pause 10
    tmp_byte = calibPress_h20 // 256
    i2cwrite sda, scl, arduino_address, [tmp_byte], blink
       pause 10
return

pressSet_to_adc 'subroutine to the convert the pressure in inH20 into an adc value.
  testPressSet_adc = (testPressSet_h20/2*calibration_slope)/50
return

send_Arduino_adc_Setpoint 'subroutine to send the adc setpoint to the Arduino
  i2cwrite sda, scl, arduino_address, [50], blink
  pause 10

  tmp_byte = testPressSet_adc >> 8
  i2cwrite sda, scl, arduino_address, [tmp_byte], blink
  pause 10

  tmp_byte = testPressSet_adc // 256
  i2cwrite sda, scl, arduino_address, [tmp_byte], blink
  pause 30

return

stop_PIC_Control 'subroutine to stop the motor controlling pic from controlling
  low control_mot
return

start_PIC_Control 'subroutine to start the motor controlling pic controlling at a given adc value
 SSPCON1.5 = 0  'configure scl and sda lines as normal I/O lines
 high control_mot 'allow pic to start controlling
 pause 100  'wait for motor controlling pic to grab adc setpoint
 SSPCON1.5 = 1 'con figure the scl and sda lines again.
return

keypad:
'poll to figure out which key, if any has been pressed
     'check column 1
     Low col1 : High col2 : High col3  : high col4
     if(row1 == 0) then
     pressed_key_num = 0
     endif
     if (row2 == 0) then
     pressed_key_num = 4
     endif
     if (row3 == 0) then
     pressed_key_num = 8
```

```
endif
if (row4 == 0) then
pressed_key_num = 12
endif

'check column 2
high col1 : low col2 : High col3 : high col4
if(row1 == 0) then
pressed_key_num = 1
endif
if (row2 == 0) then
pressed_key_num = 5
endif
if (row3 == 0) then
pressed_key_num = 9
endif
if (row4 == 0) then
pressed_key_num = 13
endif

'check column 3
high col1 : High col2 : low col3 : high col4
if(row1 == 0) then
pressed_key_num = 2
endif
if (row2 == 0) then
pressed_key_num = 6
endif
if (row3 == 0) then
pressed_key_num = 10
endif
if (row4 == 0) then
pressed_key_num = 14
endif

'check column 4
high col1 : High col2 : high col3 : low col4
if(row1 == 0) then
pressed_key_num = 3
endif
if (row2 == 0) then
pressed_key_num = 7
endif
if (row3 == 0) then
pressed_key_num = 11
endif
if (row4 == 0) then
pressed_key_num = 15
```

```
    endif

    high col4
return

blink:
  low led
  pause 50
  high led
  pause 50
  low led
  pause 50
  high led
  pause 50
  low led
  pause 50
  high led
  pause 50
  low led
  pause 50
  high led
  pause 50
RETURN
```