Thermostat Design Plan

Engineer:

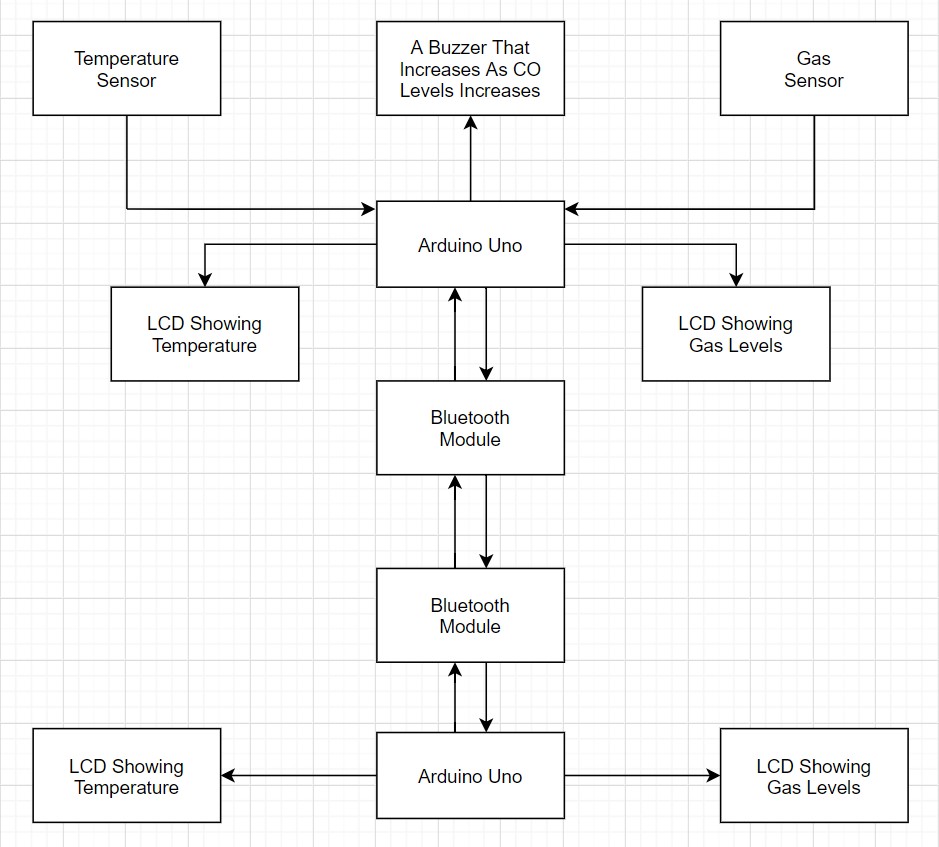
Yours Truly

Andy Garcia

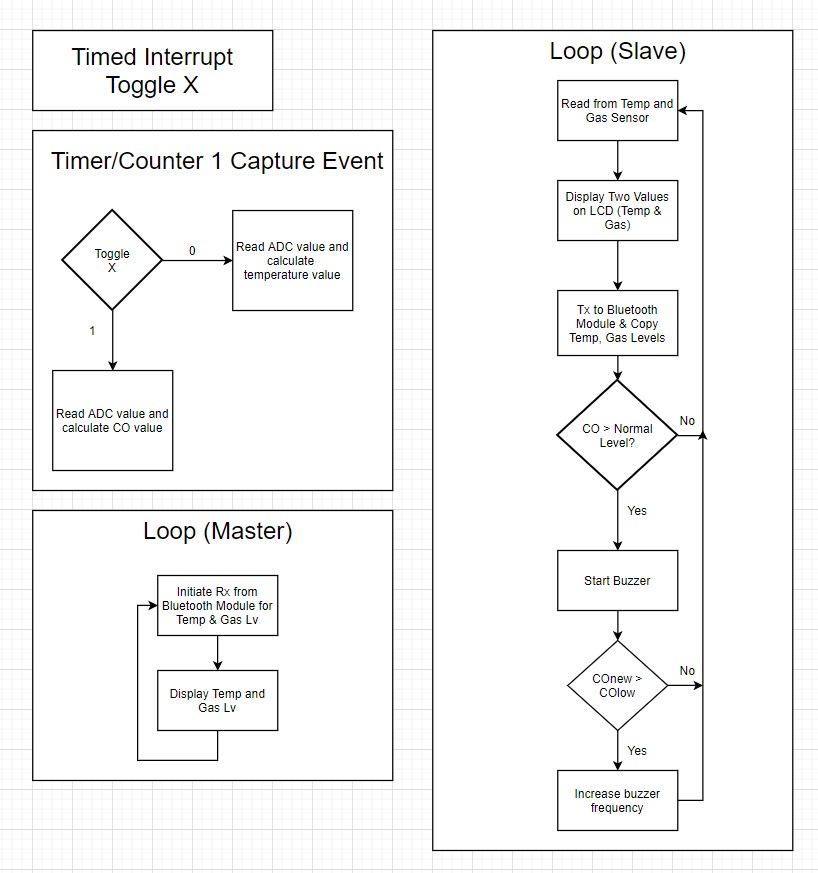
**Introduction:**

I want to showcase you my design project, my wireless thermostat. My thermostat measures 2 components; temperature and gas levels that comprised of carbon dioxide, methanol, and liquefied petroleum gas (LPG), and to display it on an LCD, connected to a physical Arduino. When the gas level gets too high, where it becomes dangerous for a person to inhale, it will start to buzz to warn the person to get out of the toxic environment. The buzzer will start to beep faster as the CO level gets higher, emphasizing the severity. The thermostat will be equipped with a Bluetooth module that will transmit the temperature and gas levels to a remote Arduino that is away from harm’s way and within Bluetooth range (20 meters) and display both values to another LCD.

# Block Diagram

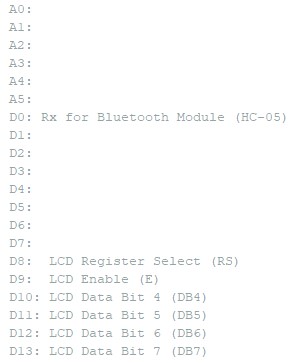


# Flowchart

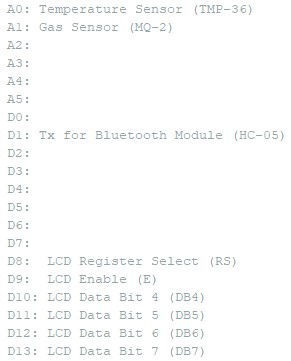


# I/O Pins Used

Master Arduino (Remote Arduino)



Slave Arduino (Arduino with Sensors)



# Hardware Description

The temperature sensor used in the slave Arduino is a TMP-36 (as shown in Appendix), which is connected in parallel with a bypass capacitor and it is connected to pin A0. It is used for detecting temperature from -50 to 450 degrees Celsius.

The gas sensor used in the slave Arduino is the MQ-2 (as shown in Appendix), which is connected to pin A1. It is used for detecting carbon dioxide, methanol, and liquefied petroleum gas (LPG). It outputs a voltage from 0.1V to 0.3V.

The Bluetooth module used in each Arduino’s is the HC-05 (as shown in Appendix), which is connected to pins D0 & D1. On the slave Arduino, it only uses pin D1 to send data to the other Arduino. On the master Arduino, it only uses pin D0 for receiving data. Its default baud rate is 9600, but I modified it to use a baud rate of 38,400 for faster transmission.

The LCD module (as shown in Appendix) is used in both Arduinos’ to display the temperature and gas values. Both of them have the same pinout. D8 is for the register select, D9 is for the enable, D10-D13 for data bit 4-7.

# Software Description

In both Arduinos’, both utilized the LCD library that contains all the LCD functions and constants, and this was downloaded from the classes BlackBoard page. In the master Arduinos code, in the setup, it starts off by disabling all interrupts to allow the following code to be uninterrupted. It first starts off by initializing the LCD, then setup the USART for the Bluetooth module to function. I setup the baud registers UBRR0H = 0x00 & UBRR0L = 0x67. Beforehand, when I was pairing both Bluetooth modules together, I changed the baud rate from 9600 to 38400 to increase the transfer speed in the AT mode. In order to pair the Bluetooth modules, in the AT mode, I had to do the following; First connect the Rx ports of the Bluetooth module and the Arduino and Tx ports as well. Then power on the modules and power the enable pin with 5V, and press the AT command button on the module, and using the serial monitor, with a series of AT commands, I had to use the AT+ADDR? command on the slave module, I received the address of the module. By default, the module is programmed as slaves. On the other module, using the AT+ROLE=1 command, I set the role of the module as master and then after that I wrote AT+BIND=”Address” to bind with the slave module. When both modules are on, the master module will connect with the slave module automatically. Then to change the baud rate,

I used the AT+UART=”BaudRate” to command to change it from 9600 to 38400 for both modules. These commands were referenced from the HC-05 AT command set guide from the manufacturer of the module. I set UBRR0 to 25, and this was calculated by using the formula provided in lab 14 and reversed engineer it (UBRR0 = (16 \* 106) / (16 \* 38400) – 1). Then I configured the UCSR0B register to enable to receiver and enable an interrupt when receiving data through the USART. In the UCSR0C register, I enabled asynchronous USART with an 8bit character size. Then I reenabled interrupts. A subroutine used in the loop function was creating a decimal point for the temperature and gas level reading, and using a series of LCD commands, I display the information on the LCD. In the USART\_RX ISR, I stored the contents from UDR0 to the gas variable, then wait for the USART I/O register to clear, then read from UDR0 again and store it in the temperature variable. In the slave Arduino code, I use an external function to calculate the average of an array by the number of samples. The parameters is the array and a constant passed by value. In the setup function, it contains the registers for the ADC to function, and setup timed interrupts to trigger every 350ms. 350ms was used since that provided the most stabled results. I initially used 300ms but was resulted of that was inaccurate readings. 350ms was achieved by using the OCR1A value of 21875 by using the following formula that was provided from Lab 6 (OCR1A = (300ms \* 16 X 106) / (256) – 1). I configured the register to use fast PWM and the top value is OCR1A and enable an interrupt to happen using timer/counter 1. Then I configured the register to configure the frequency of the active buzzer. I used fast PWM and used OCR0A as the top value, which was chosen to be 255 to achieve the lowest frequency when no gas is present. When configuring the USART registers, I enabled transmission and allow 8-bit communication. Two subroutines were used in the main function. One subroutine was used to collect the gas reading from the MQ-2 sensor, using the average and convert it into a data we can understand and the other to collect the temperature reading from the TMP-36 and convert it using the average function to get crucial data. Then just like the master code, it used the same subroutine to convert that data so that the LCD can display those numbers. Then the next subroutine is used to transmit the temperature data via the Bluetooth module, then wait for the USART I/O register to clear, then send the gas value. Then in the TIMER1 COMPA ISR, every 350ms, it switches the subroutine that determines which port to gather from the sensors.

# Peripherals

Peripherals used was the ADC, TCNT0(PWM), TCNT1(Timed Interrupts), & the USART.

Hardware used with the ADC is with the TMP-36 & MQ-2(Gas Sensor). Hardware used with TCNT0 to activate PWM is connect with an active buzzer. TCNT1 indirectly uses the TMP-36 and MQ-2. The USART relates to the HC-05 (Bluetooth module).

# Functionality

The project is a success. Not only were the main objectives achieved (ADC, Timed Interrupts/PWM), but an addition that I noted in the progress report (USART) by adding wireless support via Bluetooth was implemented successfully. The only addition that didn’t make it was the real-time clock. Getting the Bluetooth module took a very long time to work since the first set of Bluetooth modules came in defective/didn’t work (HC-10) & took me a couple of days to figure out why, I had to get them returned an order a new different set (HC05) of modules and figure out how to use those and configure it to get the proper data to transfer without having any issues, and the LCD that was supposed to give me a bigger display to place the time didn’t work. I found out I needed to solder the pins. Since I didn’t have the knowledge to execute, and that required to buy more equipment on my budget, I scratched that idea out. Even without the clock aspect, my project is still a fully functioning thermostat with wireless capabilities.

# Challenges

A huge challenge to overcome was to learn how to utilize the Bluetooth modules. I first started out with HC-10 modules which had Bluetooth 4.0 support & came with an app that can transfer the data you want onto your phone, which I found attractive. That’s why I picked it.

Unfortunately, the instructions that came with it wasn’t very clear how to use them. My phone couldn’t even connect to it, which I concluded after two days of tinkering with it, I returned them and order a new set, which I found was more popularly used (HC-05). It only used Bluetooth 2.0 support, but I only cared if it worked. Since there were plenty of guides on how to program it using cheater code, I thought I could translate the code to C and make it work. Another issue that relates to the first was pairing them. In one of the guides, it told me I need to be in AT mode to configure them to communicate with each other. I had to look through 2 different guides and 4 different YouTube tutorials to get it to work. I first had to learn how to connect one module to my phone and turn off/on an LED wireless using an app, then learn how to pair two modules together. It took a bit of time to figure out how to configure it, but I was successful. The next obstacle was to relearn how to use the USART, since I couldn’t get the second circuit in lab 14 to work, I had no idea how to get two Arduinos to communicate with each other. It took a while after trying to get the second circuit to work with both of my physical Arduinos to work, and from a little help from a classmate, I got it to work and used that knowledge to my project. The next obstacle I had to face was how to transfer two variables via the USART. I previously learned how to transfer one thing only, but not two different things simultaneously. I had to look through both my circuits from lab 14 and my first lab had a comment that stated to checked for an empty data buffer before sending data to UDR0. I then had an idea an implemented a while loop in the USART\_RX ISR that after sending the first variable, wait for the buffer to clear, then send the other variable again. That solved the problem.

# Labs/Activities

Several labs help contribute to my project. The following labs that helped me were lab 3

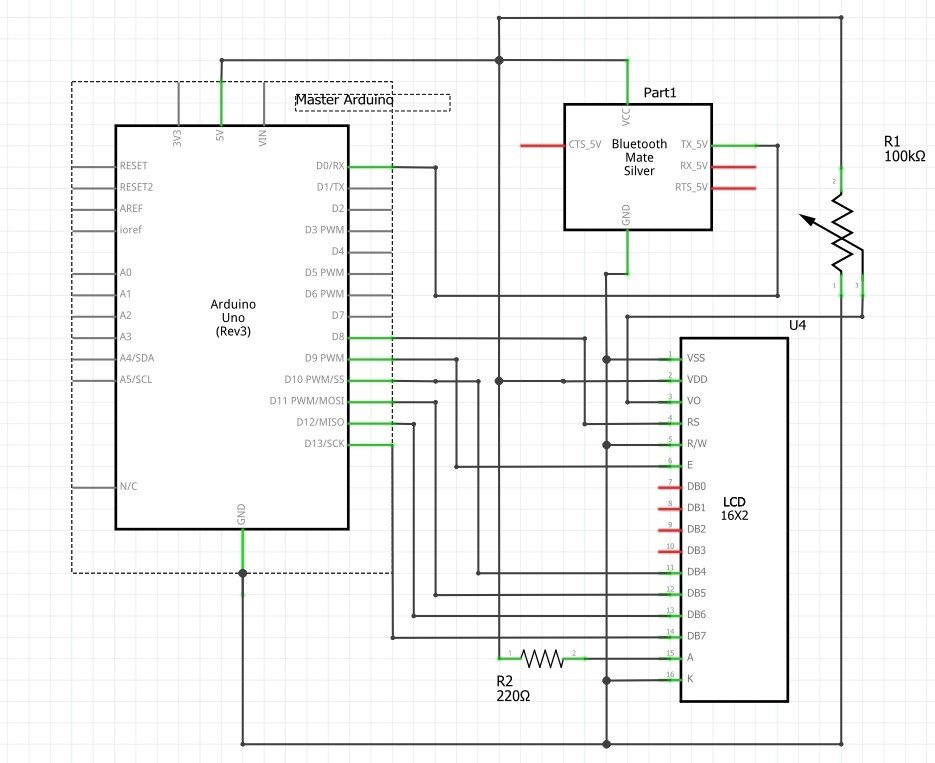
(ADC Lab), lab 4 (Sensors & Calibration), lab 6 (Timed Interrupts), lab 7 (PWM), and lab 14 (USART). To be specific, lab 4 helped me have a deeper understanding on how to develop a formula based on parameters of a sensor (for example, using excel to plot two points and getting a formula from it) & it played a part for developing the formula to control the frequency of the buzzer depending on the gas reading. Lab 6 gave me a refresher on how to calculate the period of the interrupt that I wanted to achieve, lab 7 helped me figure out what registers will help me achieve how to configure the voltage amount from a PWM port which will determine the frequency of the buzzer & how to configure the registers for fast PWM. The variable LED circuit played the other part of developing the formula which will determine the frequency of the buzzer depending on the value from the gas sensor. And lab 14, circuit 2 to be specific, helped me configure how to setup the communication to facilitate two Arduinos to communicate with each other, just that instead of having two cables connecting each other, it would be wirelessly connected.

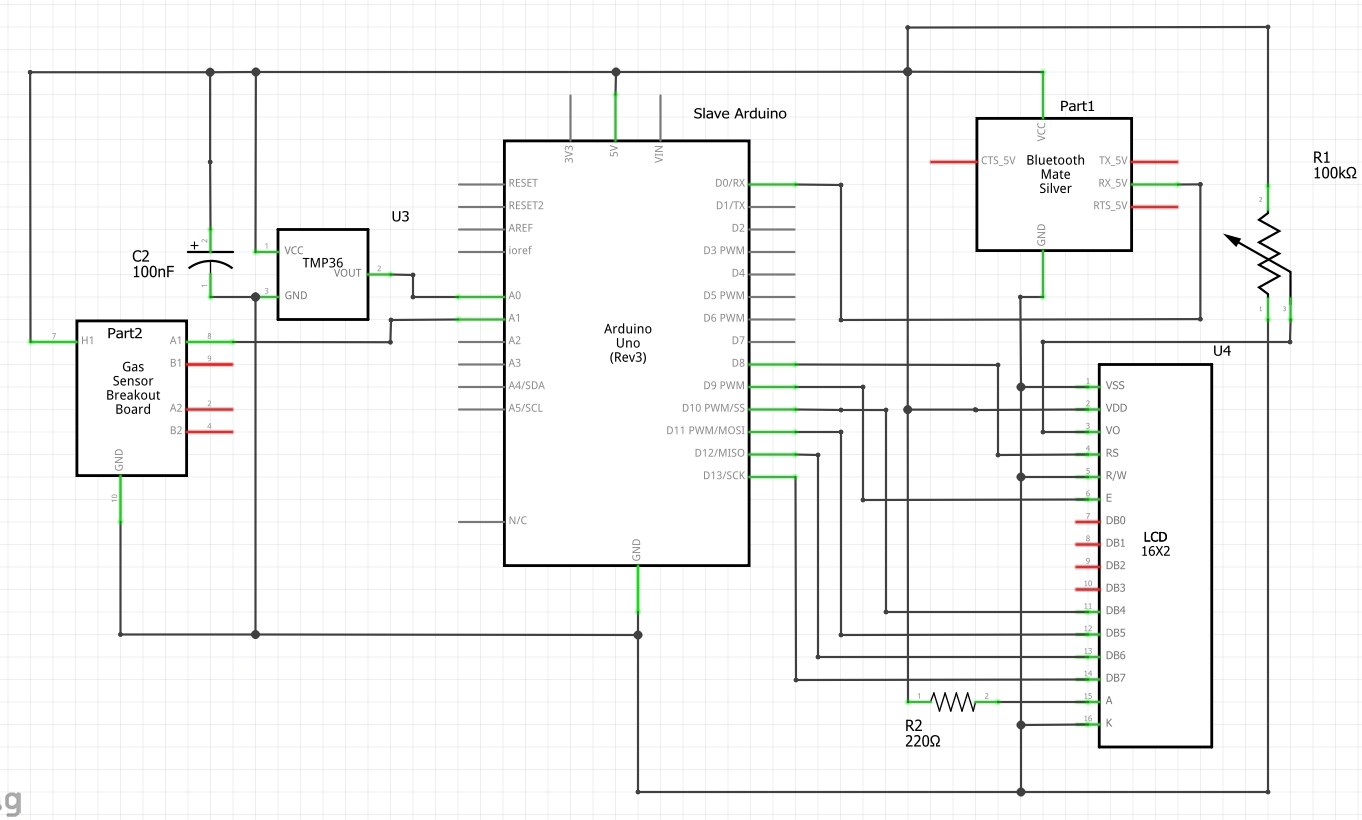
# Conclusion

In conclusion, my design project, my wireless thermostat measures 2 components; temperature and gas levels that comprised of carbon dioxide, methanol, and liquefied petroleum gas (LPG), and to display it on an LCD, connected to a physical Arduino. When the gas level gets too high, where it becomes dangerous for a person to inhale, it will start to buzz to warn the person to get out of the toxic environment. The buzzer will start to beep faster as the CO level gets higher, emphasizing the severity. The thermostat will be equipped with a Bluetooth module that will transmit the temperature and gas levels to a remote Arduino that is away from harm’s

way and within Bluetooth range (20 meters) and display both values to another LCD.

# Appendix: Code





/\*

Design Project - Master Code

This is a wireless thermostat that consist of 4 parts, a temperature sensor, a gas sensor, a buzzer that indicates the severity of the gas level, and using the USART to add Bluetooth functionality.

By: Andy Garcia

Written: May 1, 2020

Edited: July 14, 2020

I/O Pins A0:

A1:

A2:

A3:

A4:

A5:

D0: Rx for Bluetooth Module (HC-05)

D1:

D2:

D3:

D4:

D5:

D6:

D7:

D8: LCD Register Select (RS)

D9: LCD Enable (E)

D10: LCD Data Bit 4 (DB4)

D11: LCD Data Bit 5 (DB5)

D12: LCD Data Bit 6 (DB6) D13: LCD Data Bit 7 (DB7)

\*/

// include the library that contains all of the LCD functions and constants

#include "hd44780.h"

void setup() {

cli();

//Initialize LCD lcd\_init();

//Setting the USART for Bluetooth

//Setting up the Baud Registers

UBRR0H = 0x00;

UBRR0L = 0x67;

UBRR0 = 25;

UCSR0A = 0x00;

UCSR0B = 0x90; //Reciever UCSR0C = 0x0E;

sei();

}

volatile int temperature = 0; //Declare temperature variable volatile unsigned int gas = 0; //Formula to calculate gas levels void loop() { lcd\_clrscr(); //Clear LCD

// creating a decimal point for temp reading unsigned char tempLeft = temperature / 10; unsigned char tempRight = temperature % 10;

// Converting values of temp into a string char charBuffer[4];

char charBuffer2[2]; lcd\_puts("Temp: "); itoa(tempLeft, charBuffer, 10); itoa(tempRight, charBuffer2, 10);

// Displaying values of temperature on LCD lcd\_puts(charBuffer); lcd\_puts("."); lcd\_puts(charBuffer2); lcd\_putc((char)223); lcd\_puts("F");

// creating a decimal point for gas reading unsigned char gasLeft = gas / 10 ; unsigned char gasRight = gas % 10;

// Converting values of CO2 into a string lcd\_goto(40); lcd\_puts("CO2: "); itoa(gasLeft, charBuffer, 10); itoa(gasRight, charBuffer2, 10);

// Displaying values of CO2 on LCD lcd\_puts(charBuffer); lcd\_puts("."); lcd\_puts(charBuffer2); lcd\_puts("%");

\_delay\_ms(24);

}

ISR(USART\_RX\_vect)

{

//Needs to be multiplied by 2 for uncompression gas = UDR0 \* 2; while (!(UCSR0A & 0x20)){

}

//Needs to be multiplied by 4 for uncompression temperature = UDR0 \* 4;

}

/\*

Design Project - Slave Code

This is a wireless thermostat that consist of 4 parts, a temperature sensor, a gas sensor, a buzzer that indicates the severity of the gas level, and using the USART to add bluetooth functionality.

By: Andy Garcia

Written: May 1, 2020

Edited: July 14, 2020

I/O Pins

A0: Temperature Sensor (TMP-36)

A1: Gas Sensor (MQ-2)

A2:

A3:

A4:

A5:

D0:

D1: Tx for Bluetooth Module (HC-05)

D2:

D3:

D4:

D5:

D6:

D7:

D8: LCD Register Select (RS)

D9: LCD Enable (E)

D10: LCD Data Bit 4 (DB4)

D11: LCD Data Bit 5 (DB5)

D12: LCD Data Bit 6 (DB6) D13: LCD Data Bit 7 (DB7)

\*/

// include the library that contains all of the LCD functions and constants

#include "hd44780.h"

volatile unsigned int result = 0;

unsigned long ave(unsigned int arr[], unsigned char k) { //Calculates the average of n temperature readings unsigned long sum = 0; for (unsigned char j = 0; j < k; j++)

{

sum = sum + arr[j];

}

sum = sum / k; return sum;

}

void setup() {

cli();

//initialize LCD lcd\_init();

//Setting ADC registers

ADCSRA = 0xEF;

ADCSRB = 0x00;

ADMUX = 0x40;

//Configure Interrupt to trigger every 350ms when TCNT1 = OCR1A

TIMSK1 = 0x02;

TCCR1A = 0x00;

TCCR1B = 0x0C;

TCCR1C = 0x00;

OCR1A = 21875;

//Setting Active Buzzer PWM

TCCR0A = 0x23;

TCCR0B = 0x0B;

OCR0A = 255;

DDRD = 0x20;

//Setting the USART for Bluetooth

//Setting up the Baud Registers

UBRR0H = 0x00;

UBRR0L = 0x67;

UBRR0 = 25; //Baud rate is 38400

UCSR0A = 0x00;

UCSR0B = 0x08; //Transmitter UCSR0C = 0x0E;

sei();

}

//Variable that determines which analog pin to read from static unsigned char z = 0;

void loop() { lcd\_clrscr(); //Clear LCD int temperature; //Declare temperature variable unsigned int gas; //Formula to calculate gas levels static unsigned long x = 0; //Start of the loop

if (z == 0){

// Offset is 2 w/o smoke sensor // 6.6 w/ smoke sensor unsigned char offset = 4; const unsigned char n = 2; //Number of samples static unsigned int ADCvalues[n] = {}; //Array where the temperature reading will go ADCvalues[x%n] = result; //Result is the ADC reading (temperature sensor) unsigned int avg = ave(ADCvalues, n);

//Temperature Formula & multiplied by 10 to get decimal int temperatureCelcius = 500L \* avg/1023 - 50 - offset;

// Converting from celsius into Fahrenheit temperature = (temperatureCelcius \* 90L / 5) + 320;

} if (z == 1) { const unsigned char m = 5; //Number of samples static unsigned int ADCvalues2[m] = {}; //Array where the gas reading will go

ADCvalues2[x%m] = result; //Result is the ADC reading (MQ-2 sensor) unsigned int avg2 = ave(ADCvalues2, m); gas = (avg2 \* 122L/100 - 17073/100); if (avg2 > 400) { //If Gas levels exceed 30%

OCR0B = avg2 \* 2656L/10000;

} else {

OCR0B = 0;

}

}

// creating a decimal point for temp reading unsigned char tempLeft = temperature / 10; unsigned char tempRight = temperature % 10;

// Converting values of temp into a string char charBuffer[4]; char charBuffer2[2];

lcd\_puts("Temp: "); itoa(tempLeft, charBuffer, 10); itoa(tempRight, charBuffer2, 10);

// Displaying values of temperature on LCD lcd\_puts(charBuffer); lcd\_puts("."); lcd\_puts(charBuffer2); lcd\_putc((char)223); lcd\_puts("F");

// creating a decimal point for gas reading unsigned char gasLeft = gas / 10 ; unsigned char gasRight = gas % 10;

// Converting values of CO2 into a string lcd\_goto(40); lcd\_puts("CO2: "); itoa(gasLeft, charBuffer, 10); itoa(gasRight, charBuffer2, 10);

// Displaying values of CO2 on LCD lcd\_puts(charBuffer); lcd\_puts("."); lcd\_puts(charBuffer2); lcd\_puts("%");

\_delay\_ms(24);

if (UCSR0A & 0x20){

//Needs to be divided by 4 for compression UDR0 = temperature / 4; while (!(UCSR0A & 0x20)){

}

//Needs to be divided by 2 for compression

UDR0 = gas / 2;

}

x++;

}

//For ADC

ISR(ADC\_vect) { result = ADC;

}

//Timed Interrupt to toggle between A0/A1 pin

ISR(TIMER1\_COMPA\_vect) {

//Toggle z z ^= 0x01; //Read from A0

if (z == 0){

ADMUX = 0x40;

}

//Read from A1

if (z == 1){

ADMUX = 0x41;

}

}