

RASPBERRY PI WITH DS18B20  
TEMPERATURE SENSOR TO  
LCD 16X2 DISPLAY



Zachary Sanford

Dan Kelly

*Dordt University*

March 25, 2020



# RASPBERRY PI WITH DS18B20 TEMPERATURE SENSOR TO LCD 16X2 DISPLAY

Zachary Sanford and Dan Kelly

March 25, 2020

## I. INTRODUCTION

A system on a chip is the combination of a CPU, memory, and input and outputs ports all integrated into one circuit. The project uses a Raspberry Pi, hereafter referred as RPi, which is an example of a system on a chip. The intentions of using a chip platform is to interact and read data from the given surroundings and put them into a digital form for the chip to understand. The process of getting this done is with different sensors and in this instance a DS18B20 Digital Thermometer Sensor. The information read by the sensor of the surrounding environment needs to be displayed through an LCD. For the components to connect with one another, two breadboards were combined with wires going from each pin in the RPi, to the sensor, and the LCD. It is not possible for these components to communicate without some programming, which was done in Python language. There were two separate codes that were used to get the RPi to communicate to the thermometer sensor and another to get the LCD to communicate to the sensor. While the temperature is being read, the RPi will write the temperature of the surrounding environment to the LCD using the RPLCD library, given in Python. The goal was achieved after completing thorough testing of wire connections and debugging the Python code. Presentation to Professor De Boer of the working circuit allowed the project to move forward to the second part of the project [1].

Why did you choose to use a Raspberry Pi?

This part is pretty banal. See example introductions in my booklet, "How to Write a Laboratory Report" [https://dordt.instructure.com/courses/2662877/files/165368760/download?ad=pages 15, 16.](https://dordt.instructure.com/courses/2662877/files/165368760/download?ad=pages%2015%2C%2016)

## II. CIRCUIT DESIGN

The preliminary design consisted of one Raspberry Pi 3 [2], one DS18B20 Digital Thermometer Sensor [3], a 16x2 LCD Display [4], and two potentiometers. The breadboard was used for attaching the thermometer sensor, a 4.7 k $\Omega$  resistor and LCD display piece. The design was borrowed from a hobby project by Circuits Basics online [5]. The LCD display came with pins that needed to be soldered into place. This was done on the soldering table in the electronics room. A parts list of the project can be found in Figure 1.

RPi Temp. LCD Display project parts.	
Raspberry Pi 3 B+	1
DS18B20 Temperature Sensor Digital	1 Part Number: DS18B20-ND
Blue & White 16x2 LCD + Keypad Kit	1 Part Number: ADA1115
4.7 k $\Omega$ resistor	1
10 k $\Omega$ potentiometer	2
Jumper wires	19

*Figure 1 – List of parts used for project.*

The DS18B20 sensor looks much like a transistor with a flat side and curved side covered in a black 'hat'. Looking at the flat side will give direction as to connection pins. Starting with the DS18B20 on the breadboard the Vcc pin, right, is connected to the 4.7 k $\Omega$  resistor. The resistor is connected to the 3.3V pin on the RPi via jumper wire. The data pin, center, of the sensor is connected to pin seven of the RPi (GPIO4). The ground pin, left, of the sensor is connected to pin six of the RPi (GND). The sensor requires less than 1 mA of current to work. The following shows the connections between the sensor and the RPi in Figure 2.

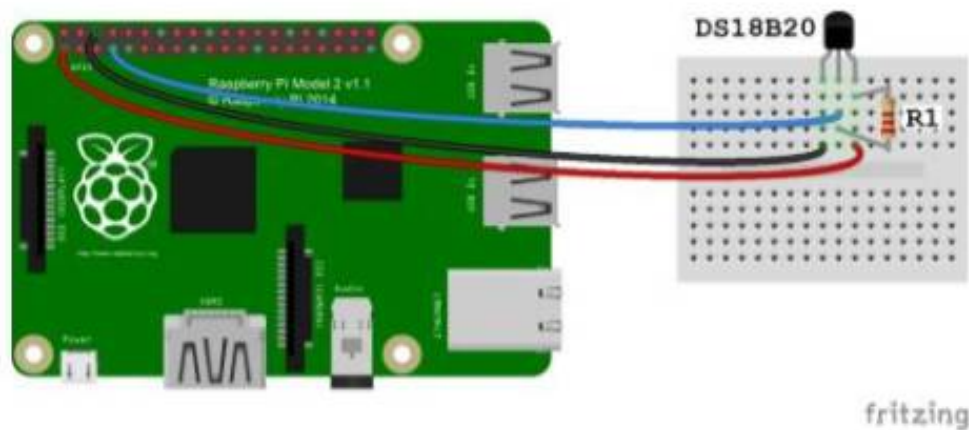


Figure 2 – Fritzing schematic of connection between DS18B20 sensor and Raspberry Pi 3 [5].

Next, wires connected the LCD display to the Vcc, GND, and GPIO pins to the RPi using the breadboard. Two connections were made from the LCD to the potentiometers before connecting to the positive and negative rails in the breadboard. The positive rail was connected from the breadboard to the +5V pin on the RPi. The negative rail was connected from the breadboard to the ground (GND) pin on the RPi. More wires connected the VDD pin and the LED Positive pin 15 to the positive rail. The following, Figure 3, lists the connections between the LCD display pins to the RPi GPIO pins. Figure 4 shows the Circuit Basics Fritzing schematic of the entire project [5].

16x2 LCD	Raspberry Pi 3
Register Select; pin 4	GPIO26
Enable; pin 6	GPIO19
Data Pin four; pin 11	GPIO13
Data Pin five pin 12	GPIO6
Data Pin six; pin 13	GPIO5
Data Pin seven; pin 14	GPIO11

Figure 3 – Wiring between LCD to Raspberry Pi 3.



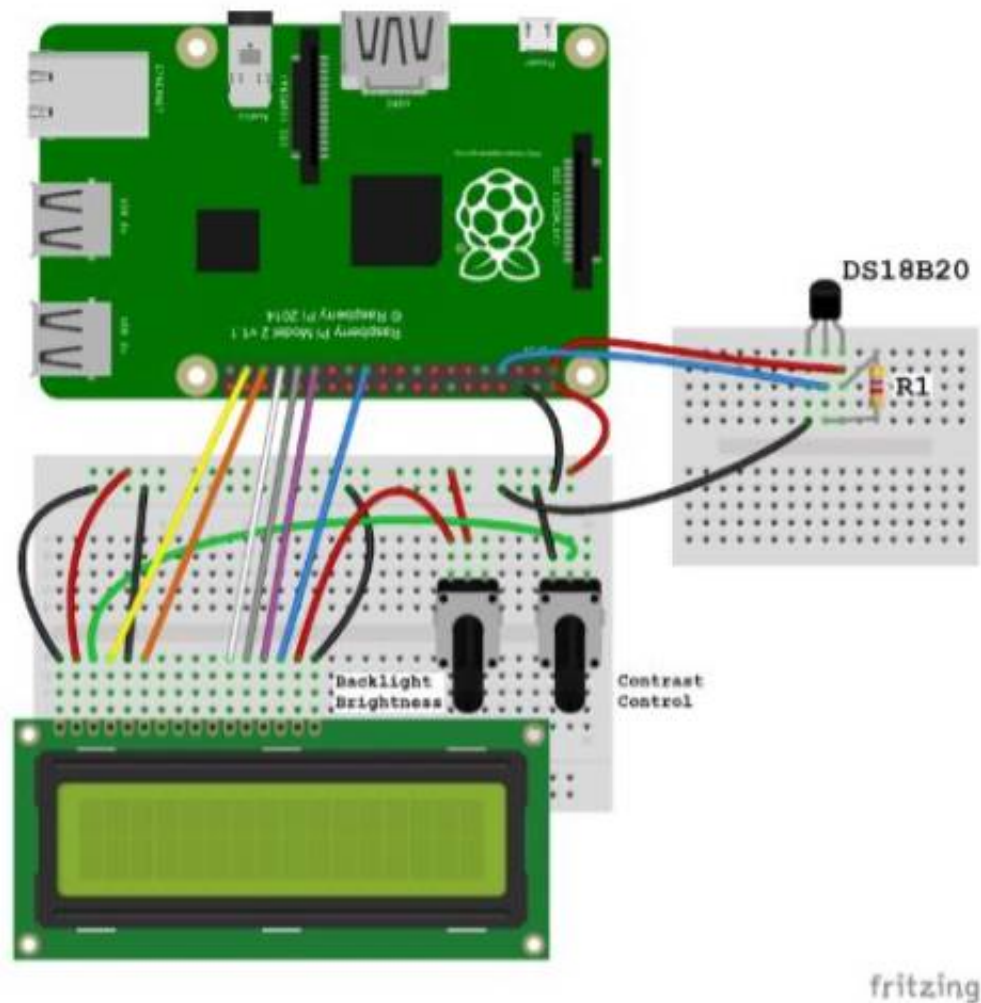
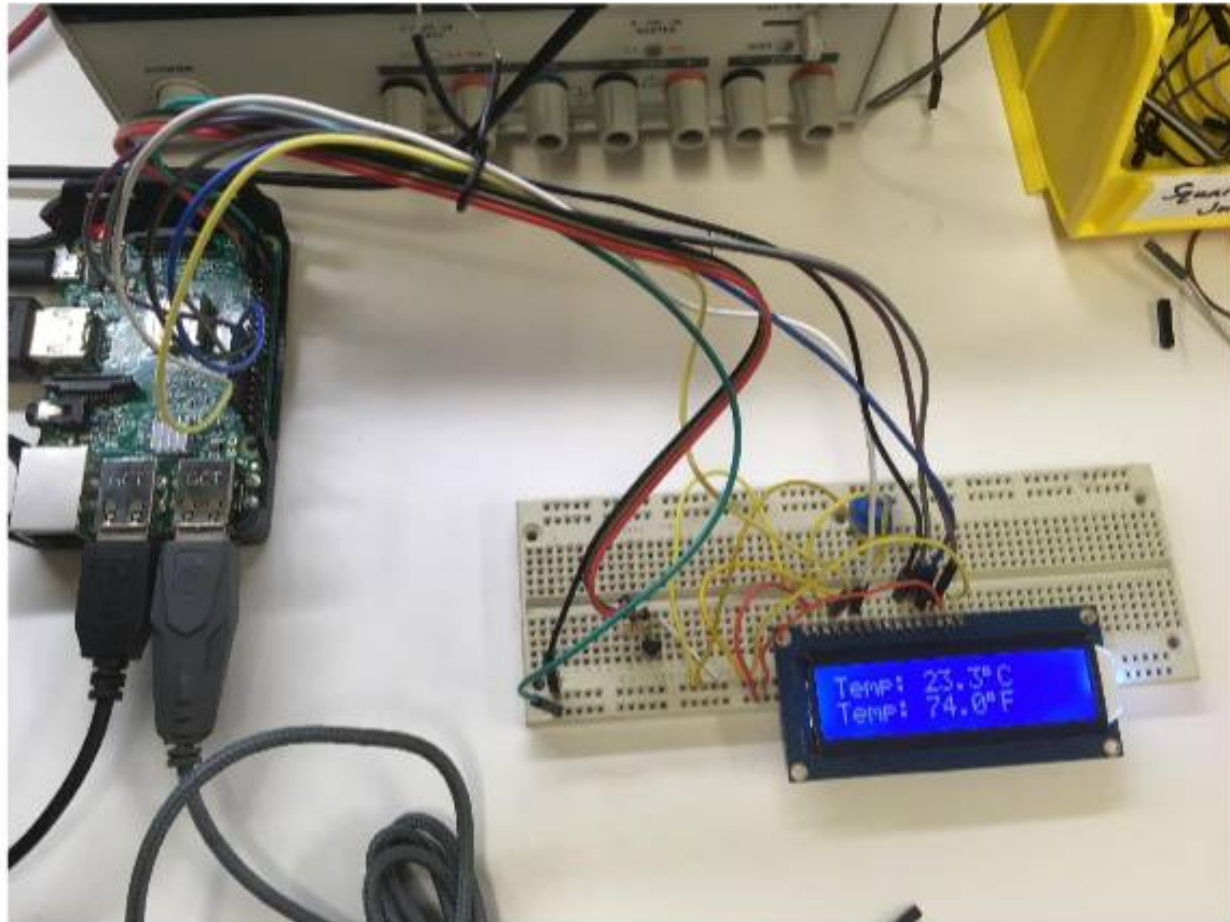


Figure 4 – Fritzing schematic of connection between LCD and Raspberry Pi in Fritzing program [5].

The connection between the LCD and the Raspberry Pi used a 4-bit mode. This was to simplify the circuit rather than using an 8-bit mode connection. Because the LCD will only be showing the temperature and not writing to the RPi it is unnecessary to have 8-bit mode. On the Circuit Basics project, it is suggested to have two 10 k $\Omega$  potentiometers for controlling the contrast and the brightness of the LCD [5]. These potentiometers were helpful for controlling the variants of the screen and displayed the temperature values under certain lighting conditions. The LCD came purchased with a keypad. It was decided that after inspection the keypad would not be needed for this project. It could be added in the second part of the project for further interactive information or

display. With the keypad taken out it will help with simplification and ease of project for the first part of the lab project. The followings, Figure 5 and 6, show the connection between the Raspberry Pi, DS18B20 sensor, and LCD both as a final product and schematic. The schematic was created original.



*Figure 5 – Final product displaying current room temperature in lab.*

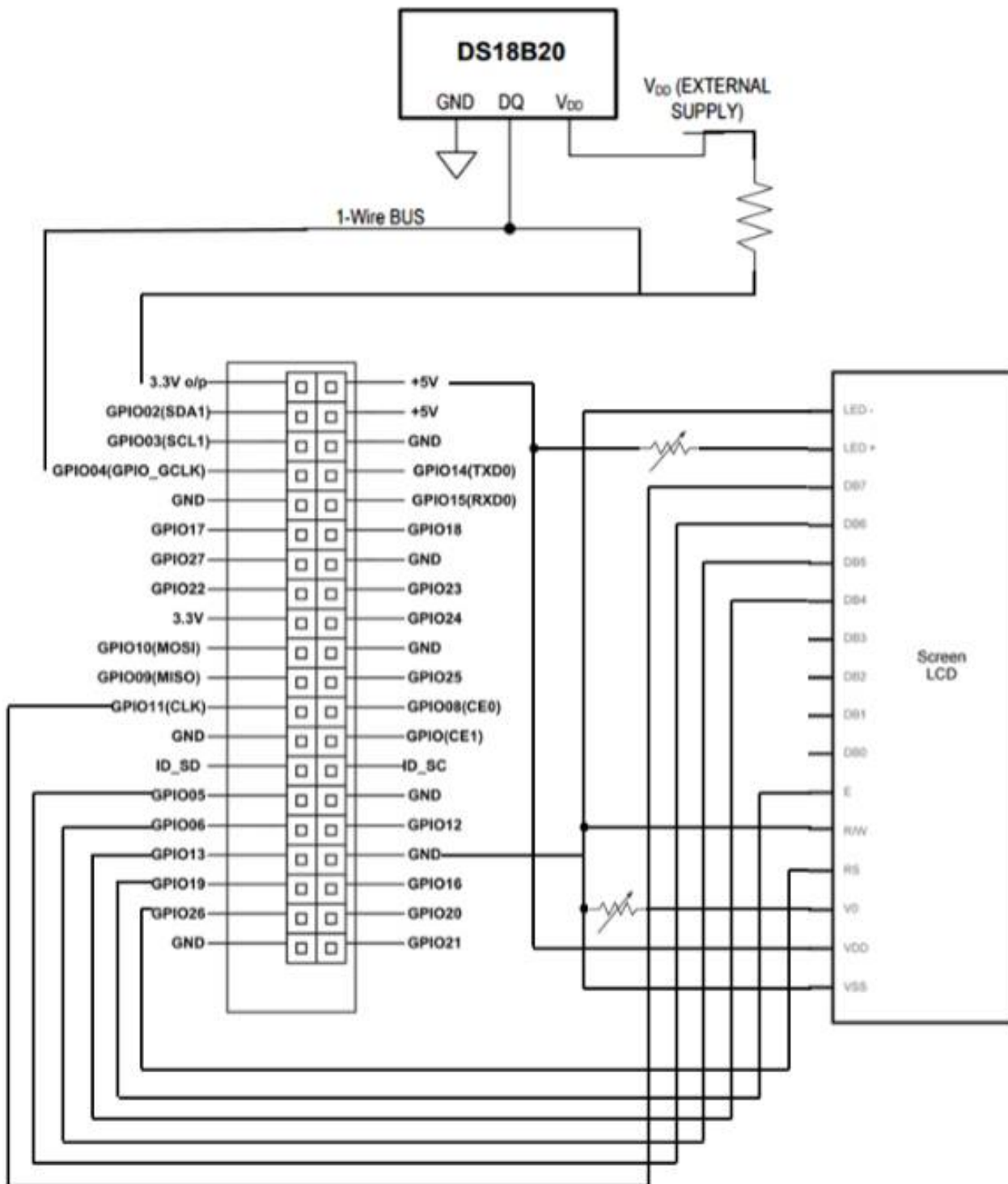


Figure 6 -- Schematic of Raspberry Pi connected to DS18B20 and LCD.



### III. CODING/LIBRARY

Two python scripts were used for talking with each accessory. The first one called, “temperature.py”, takes digital data from the DS18B20 sensor to store and display the temperature on the RPi terminal. The second script, “temp\_output.py”, talked to the LCD through GPIO pins and displayed the temperature from the sensor on the LCD. These scripts were accessed through the SSH terminal to support the operating system in the RPi. For the RPi to successfully talk between the DS18B20 Thermometer Sensor, a sub-library of commands, labeled ‘CharLCD’, was imported from the RPLCD library. The programming used for this project was RPLCD, which is a Python based library used to communicate between the RPi and the LCD [5]. RPLCD also supports GPIO which is what is being used to connect the RPi to the LCD. The RPi comes with some features already implemented on it, but others were needed to make it fully function.

The code itself had different blocks of code for the different functions that are being performed. There is the `read_temp_raw()` function that takes in the raw data temperature and is stored in the system. The next function `read_temp()` perform the calculations based upon the given temperature and converts it into either Celsius or Fahrenheit. Both values are stored within the program as variables. The temperature is taken and displayed every second. The python code uses the raw temperature data taken and displays a float point the to SSH terminal. The following code shows how it was implemented upon the terminal below in Figure 7.

```
1 import os
2 import glob
3 import time
4
5 os.system('modprobe w1-gpio')
6 os.system('modprobe w1-therm')
7
8 base_dir = '/sys/bus/w1/devices/'
9 device_folder = glob.glob(base_dir + '28*')[0]
10 device_file = device_folder + '/w1_slave'
11
12 def read_temp_raw():
13     f = open(device_file, 'r')
14     lines = f.readlines()
15     f.close()
16     return lines
17
18 def read_temp():
19     lines = read_temp_raw()
20     while lines[0].strip()[-3:] != 'YES':
21         time.sleep(0.2)
22         lines = read_temp_raw()
23     equals_pos = lines[1].find('t=')
24     if equals_pos != -1:
25         temp_string = lines[1][equals_pos+2:]
26         temp_c = float(temp_string) / 1000.0
27         temp_f = temp_c * 9.0 / 5.0 + 32.0
28         return temp_c, temp_f
29
30 while True:
31     print(read_temp())
32     time.sleep(1)
33
```

Figure 7 – Python Code for talking to DS18B20 Thermometer Sensor

Next, getting the LCD to display the given variables for temperature also needed coding. The difference between these codes is that this one provides an output to the LCD. This is accomplished through the `write_string()` command allowing an actual visible output. The return value for `read_temp()` gives the temperatures in both Celsius and Fahrenheit and proceeds to print those values upon the LCD. The temperature is updated on the LCD every second while the screen refresh rate is 200 ms. The last section of the code outputs the variables of temperature and displays them upon the LCD. The following python code for the LCD display can be seen below in Figure 8.

```

1 import os
2 import glob
3 import time
4 from RPLCD import CharLCD
5
6 lcd = CharLCD(cols=16, rows=2, pin_rs=37, pin_e=35, pins_data=[33, 31, 29, 25])
7
8 os.system('modprobe wl-gpio')
9 os.system('modprobe wl-there')
10
11 base_dir = '/sys/bus/wi/devices/'
12 device_folder = glob.glob(base_dir + '28*')[0]
13 device_file = device_folder + '/w1_slave'
14
15 def read_temp_raw():
16     f = open(device_file, 'r')
17     lines = f.readlines()
18     f.close()
19     return lines
20
21 #CELSIUS CALCULATION
22 def read_temp_c():
23     lines = read_temp_raw()
24     while lines[0].strip()[-3:] != 'YES':
25         time.sleep(0.2)
26         lines = read_temp_raw()
27     equals_pos = lines[1].find('=')
28     if equals_pos != -1:
29         temp_string = lines[1][equals_pos+2:]
30         temp_c = int(temp_string) / 1000.0 # TEMP_STRING IS THE SENSOR INPUT, MAKE SURE IT'S AN INTEGER TO DO THE PART!
31         temp_c = str(round(temp_c, 1)) # ROUND THE RESULT TO 1 PLACE AFTER THE DECIMAL, THEN CONVERT IT TO A STRING
32         return temp_c
33
34 #FAHRENHEIT CALCULATION
35 def read_temp_f():
36     lines = read_temp_raw()
37     while lines[0].strip()[-3:] != 'YES':
38         time.sleep(0.2)
39         lines = read_temp_raw()
40     equals_pos = lines[1].find('=')
41     if equals_pos != -1:
42         temp_string = lines[1][equals_pos+2:]
43         temp_f = (int(temp_string) / 1000.0) * 9.0 / 5.0 + 32.0 # TEMP_STRING IS THE SENSOR INPUT, MAKE SURE IT'S AN INTEGER TO DO THE PART!
44         temp_f = str(round(temp_f, 1)) # ROUND THE RESULT TO 1 PLACE AFTER THE DECIMAL, THEN CONVERT IT TO A STRING
45         return temp_f
46
47 while True:
48
49     lcd.cursor_pos = (0, 0)
50     lcd.write_string("Temp: " + read_temp_c() + unichr(223) + "C")
51     lcd.cursor_pos = (1, 0)
52     lcd.write_string("Temp: " + read_temp_f() + unichr(223) + "F")

```

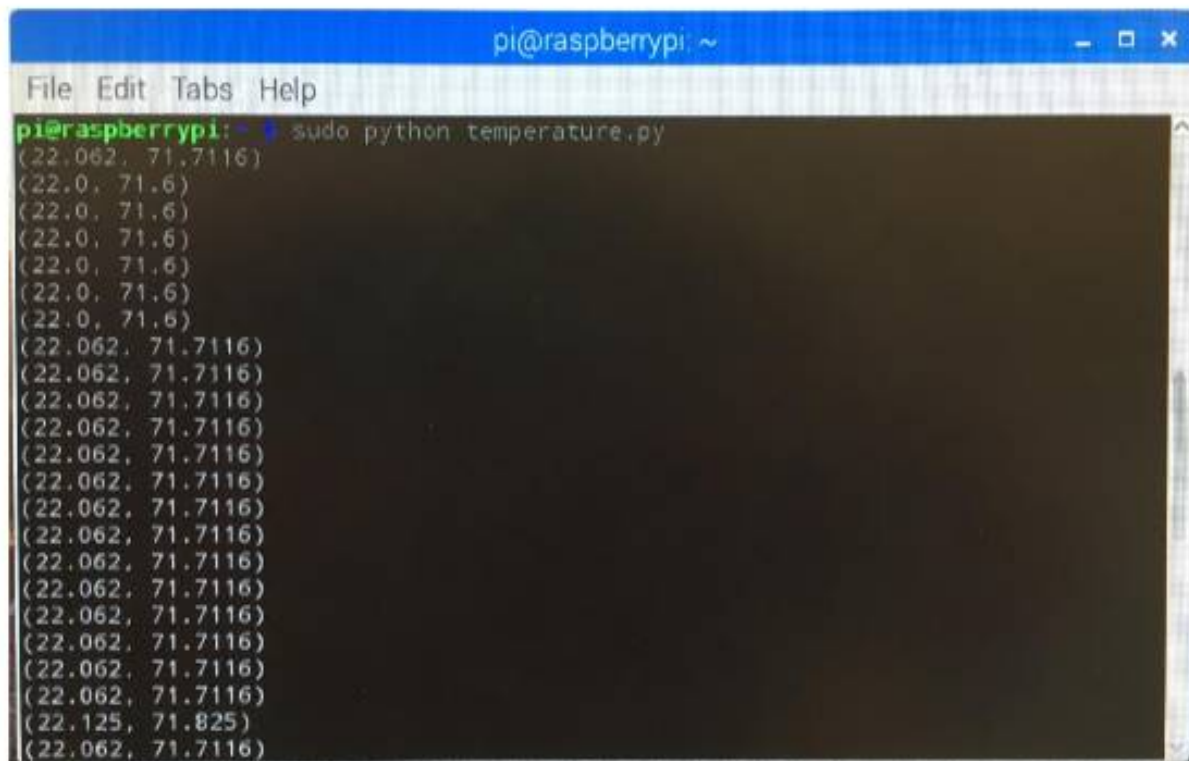
Figure 8 – Python Code for communicating to 16x2 LCD Module

## IV. RESULTS

The result of the design and functionality of this specific temperature sensor and compatible python code was displaying the current room temperature. Further design of the python code allowed for us to print the temperature on the RPi terminal. The design of the python code and library import allowed the RPi to correctly display the temperature reading, brightness and contrast onto the LCD using basic Celsius and Fahrenheit conversion equations. The sensor read the variant temperature in digital format as raw data which was translated into floating decimal points. These



floating-point values displayed on the RPi terminal validated that the sensor was reading accurate current room temperature. This was compared to the thermostat control in the room for comparison. To access the temperature the following command was executed, “sudo python temperature.py”. This command prints first Celsius followed by the Fahrenheit equivalent. The terminal output of temperature can be seen below in Figure 9.

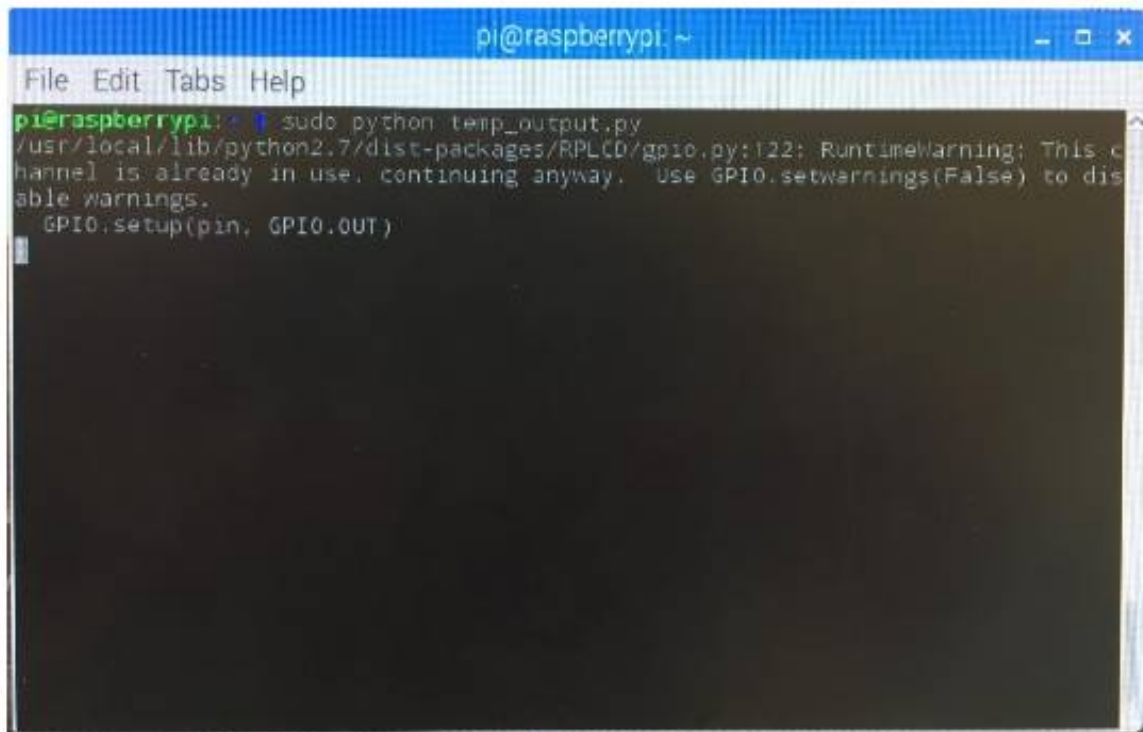


```
pi@raspberrypi: ~  
File Edit Tabs Help  
pi@raspberrypi: ~$ sudo python temperature.py  
(22.062, 71.7116)  
(22.0, 71.6)  
(22.0, 71.6)  
(22.0, 71.6)  
(22.0, 71.6)  
(22.0, 71.6)  
(22.0, 71.6)  
(22.062, 71.7116)  
(22.062, 71.7116)  
(22.062, 71.7116)  
(22.062, 71.7116)  
(22.062, 71.7116)  
(22.062, 71.7116)  
(22.062, 71.7116)  
(22.062, 71.7116)  
(22.062, 71.7116)  
(22.062, 71.7116)  
(22.062, 71.7116)  
(22.062, 71.7116)  
(22.125, 71.825)  
(22.062, 71.7116)
```

*Figure 9 -- Temperature reading from sensor given python code displayed in terminal.*

Correct wiring and python code was used for accurate display of temperature to the LCD. To access the LCD the following command was executed, “sudo python temp\_output.py”. This command activates the GPIO pins as outputs on the RPi and transfers them to the data pins in the LCD. The terminal command of “temp\_output.py” can be seen below in Figure 10.





```
pi@raspberrypi: ~  
File Edit Tabs Help  
pi@raspberrypi:~$ sudo python temp_output.py  
/usr/local/lib/python2.7/dist-packages/RPLCD/gpio.py:122: RuntimeWarning: This channel is already in use, continuing anyway. Use GPIO.setwarnings(False) to disable warnings.  
  GPIO.setup(pin, GPIO.OUT)
```

Figure 10 – Terminal command program for temperature display to LCD.

The result of the above command in the terminal prints the current temperature onto the LCD display. The code from the python program called, “temp\_output.py”, tells how the two-temperature reading will be displayed. Because the LCD is a 16x2 character array it allows for two lines of text at 16 characters long to be printed on the screen. The following is an example of the current temperature reading in Figure 11 below.



Figure 11 – Temperature display on the 16x2 LCD in Celsius and Fahrenheit.

## V. CONCLUSION

The purpose of this project was to take measurement of an analog variable, the room temperature of the surrounding environment, and display that information upon an LCD using a Raspberry Pi as the micro-controller. As seen, the project meets those requirements through providing constant feedback from the sensor. The sensor takes in the temperature and displays in both Celsius and Fahrenheit every second. The accuracy of the DS18B20 sensor was compared to the thermostat control of the current room of testing. The LCD display was accurate to within two tenths of a degree Fahrenheit with the thermostat. This conclude reasonable analysis that the sensor is working properly. The connection of the components would not have been possible without using the Python code. The code allowed the components to interact with one another through GPIO pins to take in the environmental temperature and retrieve it, translating the data into readable digital output. Overall, this design was straight forward and easy to follow, thanks to step-by-step instructions given by Circuit Basics online [5] and the guidance of the class instructor, Professor Douglas De Boer [1]. The second part of this lab project will be to upload the temperature readings to an accessible website online through the RPi wifi or ethernet port and retrieve anytime or place the current room temperature that the sensor resides.

## VI. REFERENCES

- [1] Professor De Boer, *Professor of Engineering*, Dordt University Spring 2020 (conversation with).
- [2] Components 101, *Raspberry Pi 3 Pin Configuration Features Datasheet*, Accessed Feb. 19, 2020.  
<https://components101.com/microcontrollers/raspberry-pi-3-pinout-features-datasheet>
- [3] Maxim Integrated, *DS18B20: Programmable Resolution 1-Wire Digital Thermometer*, Accessed Feb. 19, 2020  
<http://www.circuitbasics.com/wp-content/uploads/2016/03/DS18B20-Datasheet.pdf>
- [4] Components 101, *16x2 LCD Module Pinout Datasheet*, Accessed Feb. 19, 2020.  
<https://components101.com/16x2-lcd-pinout-datasheet>
- [5] Circuit Basics, *Raspberry Pi DS18B20 Temperature Sensor Tutorial*, Accessed Feb. 19, 2020.  
<https://www.circuitbasics.com/raspberry-pi-ds18b20-temperature-sensor-tutorial/>

\_\_\_A\_\_\_ Style (The introduction is weak)

\_\_\_A\_\_\_ Completeness

\_\_\_A\_\_\_ Accuracy

Overall report grade: A

Lux Sensor and Digital Display

Patrick M. Munsey, Kyle D. Waas

Dordt University, EGR 304

March 24th, 2020



## I. Introduction

This report explores the power supply breadboarding and the void loop and float collection coding techniques used to create a lux sensor to display interface. The BH1750 lux sensor and 0.96" I2C digital display were constructed prior to this project and its designs and construction will not be covered in this report. An arduino system, breadboard, and accompanying wiring was also provided by Dordt University to complete the projects needed materials.

## II. Method

This project was greatly assisted by an Instructables project by nafisaanjum13 [1]. The link is included in the references and noted here since this page will be mentioned frequently throughout this section.

For our power supply breadboarding, the specifications and limitations of both instruments were taken into consideration. Since the BH1750 and digital display weren't both rated similarly in voltage power, they needed to be connected to different nodes coming out of the Arduino system. The digital display was the larger of the two and worked well on the stable

five volt output pin that the Arduino comes programmed with. The sensor was also just as easy to power up since Arduinos also come with a built in 3v3 pin that adheres well to the BH1750's "3v3" is a European slang code notation properly used only on small parts and 2.4-3.6 voltage needs.

silk-screenings where space is limited. (Like a color code.) You mean "3.3 V" which  
This deviated from the instructions put forth by Instructables as it was suggested to only is the way the Arduino Uno board is silk-screened. ("3v3" is an informal extension of use the 3v3 pin to power both instruments. However, no complications were found through our a legitimate code for labeling resistors and capacitors, the IEC 60062:2016 code.) methodology and it was decided to not fix a working design(Figure 1).

You imply that your circuit does not exactly match what is given in the Instructable. This means you should provide a schematic (a schematic was requested) of your circuit. Without it, your project cannot be duplicated by the reader of this report. Further, I suspect that you used one I2C port on your Arduino which operated at 5 V logic levels but was connected to a board operating from 3.3 V. I'm sure it worked, but this would be a high-stress, early failure type of connection.



For the interface programming, the Arduino IDE program was used to communicate with the arduino system and subsequently, the instrument (Figure 2,3,4). The set up of the program proved to be the most intricate process we encountered. The Instructable page provided a .ino file with a working program, however, the digital display did not seem to sync up within the program's set up. To troubleshoot this, we used a test program that searched for the I2C port connection and gave feedback on the connection status (Figure 5&6). Through this we found the correct communication pins and proceeded with the given .ino file. Libraries needed were also provided on the Instructables page [1].

The second half of the program is for the communication aspect between the sensor and display. A `void` loop was used as the basis of the Arduino's instructions. A `void` loop in the

That the loop is of the type, "void" is so unimportant in this context that your writing Arduino IDE instructs the system and its attached instruments to continue operating for as long losses authority when you mention it. See for one example the Wikipedia article as the Maker Board is active. Within this loop, our incoming lux value from the BH1750 is that explains what the type "void" means: [https://en.wikipedia.org/wiki/Void\\_type](https://en.wikipedia.org/wiki/Void_type) accepted at a 'float' in order to display a couple decimal places of precision that the sensor can You have not defined what a "Maker Board" is. The phrase also does not appear give us. This is important since leaving this out would under-utilize our purchased equipment. ✓ Good anywhere in your reference--the Instructables page.

We acknowledged that in circumstances where the two decimal precision isn't valuable to the user, a cheaper sensor could be purchased and the 'float' left out in order to keep a program cleaner and thinner. This thinning would then allow our system to run faster with less CPU usage. Had we been building a protective device for someone's eyes that needed to read the lux value of the user's room and polarize a set of lenses in response to higher values, a three to four millisecond delay could not be as acceptable. However, with our set up and program being very rudimentary, we decided to keep the feature and show as much precision as we were allowed.

### III. Results

Though unsuccessful at first due to a miswired I2C connection at the digital display, we did get a functioning light sensor that would read out its current lux intake through a digital 128x64 pixel display. I guess I'll just have to take your word for it--and the demo. However you could have performed some sort of testing, even just covering the sensor with your hand and reporting results, to prove that it works. Or use the iris of a camera lens and report on how changing f-stops changes the lux reading. . . . Think of how to prove goodness.

### IV. Conclusion

The project was successful and created a light sensor that fit our needs. The decision to keep the precision aspect of our project was deemed suitable since there was not a time related aspect to our needs. Because the device was created as a hobby piece and a skill building exercise in I2C communication buses, the delay in sensor readout is without consequence.

#### Progress seen:

In the lab--you get things working and have more confidence now.

On paper, the report shows a clear IMRAC (Introduction, methods, results, conclusion) presentation order which will be easily followed in industry and academia.

#### Recommended areas of improvement:

Writing with authority--understanding the meaning and origin of words and phrases used.

Manipulating a word processor, especially with respect to illustrations

(Figure 1 looks like it got pasted in on its side.

Figures 2, 3 and 4 are really one figure and could be made to look that way.

Same for Figures 5 and 6.)

Suggestion: Run your next lab report past me at least two days before it is due. (Send it to me by e-mail with a request for me to review it.) I'll get it back to you with comments. Then make improvements and turn it in.



## References

- [1] nafisaanjum13. "Mini Digital LUX Meter." Instructables.

<https://www.instructables.com/id/Mini-Digital-LUX-Meter/>. (accessed Feb. 12th, 2020)

## Figures

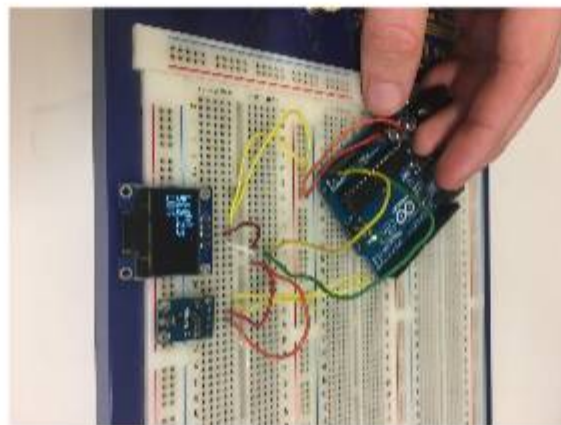


Fig. 1, Photo of the project as built.

All figures need captions, even if the subject of the figure seems obvious.

```

1 //
2 // Kyle Waus and Patrick Runney
3 // LGR 384 Coding
4 // Light Sensor Project
5 //
6 // This is the main program that will do the reading and display
7 //
8 // Found online help with libraries included
9 //
10
11 //List of all included Libraries
12 #include <BH1750FV1.h>
13 #include <Wire.h>
14 #include <Adafruit_GFX.h>
15 #include <Adafruit_SSD1306.h>
16 #include <SPI.h>
17
18 //Defining the variables that will never change
19 #define SCREEN_WIDTH 128 // OLED display width, in pixels
20 #define SCREEN_HEIGHT 64 // OLED display height, in pixels
21
22 #define OLED_RESET -4
23
24 //From import Adafruit
25 Adafruit_SSD1306 display(SCREEN_WIDTH, SCREEN_HEIGHT, &Wire, OLED_RESET);
26
27 // Create the LightSensor instance
28 //From import BH1750FV1
29 BH1750FV1 LightSensor(BH1750FV1::I2C_Addr);
30
31

```

Fig. 2 Missing caption. . . same for all following figures.



```

27 // Create the LightSensor instance
28 //Then import B01750FV2
29 B01750FV2 myLightSensor (B01750FV2::B01750FV2Default);
30
31 void setup() {
32
33   Wire.begin();
34
35   //Start serial with specific here statements to print for debugging purposes
36   Serial.begin(9600);
37
38   Serial.println("HERE1");//Debugging step
39
40   //If it can't find the display it will fail forward
41   if(!display.begin(SSD1306, 128*64, 0x03)) { // Address 0x03 for 128x64
42     //Found the address 0x03 from our other program
43     Serial.println("SSD1306 address failed");
44     for(;;);
45   }
46 }
47
48 // Clear the whatever is on the display from last time
49 display.clearDisplay();
50
51 //Begin the reading the Light Sensor
52 LightSensor.begin();
53 }
54
55

```

Fig. 3

```

56 void loop(){
57
58   Serial.println("HERE");//Debugging step
59
60   //Gets the light intensity as a float
61   float lux = LightSensor.GetLightIntensity();
62
63   //Sets up our text as white and 1:1 scale
64   //Also will start at top-left corner
65   display.setTextSize(2);
66   display.setTextColor(SSD1306_WHITE);
67   display.setCursor(0,0);
68   //What it will display
69   display.println(F("Light : "));
70   //Our current Light Intensity
71   display.println(lux);
72   //Units labeled
73   display.println(F("LUX"));
74   display.display();
75
76   Serial.println("HERE2");//Debugging step
77
78   //Leave it on the screen for some time before it will read again
79   delay(2000);
80 }
81

```

Fig. 4

```

13 #include <Wire.h>
14
15 void setup()
16 {
17   Wire.begin();
18
19   Serial.begin(9600);
20   while (!Serial); // Leonardo: wait for serial monitor
21   Serial.println("I2C Scanner");
22 }
23
24
25 void loop()
26 {
27   byte error, address;
28   int nDevices;
29
30   Serial.println("Scanning...");
31
32   nDevices = 0;
33   for(address = 1; address < 127; address++)
34   {
35     // The I2C scanner uses the return value of
36     // the Wire.endTransmission() to see if
37     // a device did acknowledge to the address.
38
39     Wire.beginTransmission(address);
40     error = Wire.endTransmission();
41
42     if (error == 0)

```

Fig. 5

```

43     {
44       Serial.print("I2C device found at address 0x");
45       if (address < 16)
46         Serial.print("0");
47       Serial.print(address, HEX);
48       Serial.println(" ");
49
50       nDevices++;
51     }
52     else if (error==4)
53     {
54       Serial.print("Unknown error at address 0x");
55       if (address < 16)
56         Serial.print("0");
57       Serial.print(address, HEX);
58       Serial.println(" ");
59     }
60   }
61   if (nDevices == 0)
62     Serial.println("No I2C devices found.");
63   else
64     Serial.println("done.");
65   delay(5000); // Wait 5 seconds for next scan
66 }
67

```

Fig. 6

\_\_B\_\_ Style: Most of the comments above have to do with style.

\_\_C+\_\_ Completeness: Lack of a schematic is a pretty strong weakness in this report

\_\_A\_\_ Accuracy: No problem here!

Overall Report grade: B

# MICROCONTROLLER DESIGN PROJECT

## TEMPERATURE CONTROLLER

Shane R. Tinklenberg, Charles J. Young, Matthew W. Van Eps, Lucas J. Nelson  
March 25, 2020

### I. Introduction

A microcontroller is a discrete computer typically employed in an embedded system to fulfill a specific function. One example of an embedded system that may utilize a microcontroller is a temperature control system. Temperature control systems can utilize a variety of types of sensors to measure the controlled environment's temperature. These types include nothing too important. There are a variety of control methodologies used to control temperatures including PI and PID control, however in this scenario we will use a <sup>bc</sup> Bang <sup>bc</sup> Bang method due to its simplicity, ease of implementation, and high functionality relative to low cost.

The purpose of the temperature control system was to create a working embedded microcontroller system that would regulate the ambient temperature within an environment. Even though temperature control systems exist, the goal was to create a working customizable temperature control system using the Arduino Development Environment. The Arduino platform was chosen over other systems like the Raspberry Pi for various reasons. The first reason being the fact that the Raspberry Pi does not have its own analog to digital converter which would prohibit us from reading analog temperature sensor values without a separate discrete analog to digital converter. The second reason we chose the Arduino platform over the Raspberry Pi is because the Raspberry Pi runs its own Linux distribution which <sup>is</sup> ~~seemed like~~ <sup>(Write with authority.)</sup> unnecessary considering the simple control scheme we were aiming to implement. Lastly, the third reason we



chose the Arduino over the Raspberry Pi is because of its ability to drive more current out of its general-purpose input output pins. In the end by using an Arduino Mega, an LM335 temperature sensor, a solid state relay, a dual contact electro-mechanical relay, a hair dryer<sup>or</sup> (any properly rated resistive heating element with a blower fan would suffice)<sup>space</sup>, and various resistors and capacitors, we were able to successfully implement a Bang Bang Temperature Control system via an embedded microcontroller system. It was demonstrated that we were able to control temperature within a definable range to a set temperature.

## II. Hardware

We aimed to accomplish our goal of constructing a working temperature controller using an Arduino Mega, hair dryer, a solid-state relay<sup>i</sup>, an Archer 120 VAC DPDT Relay Cat No. 275-207, a Texas Instruments LM335<sup>ii</sup>, two LED Seven Segment displays part number MAN6680<sup>iii</sup>, various resistors, various capacitors, two push buttons, and a single chip Schmitt Trigger integrated circuit part number SN74LS14N<sup>iv</sup>.

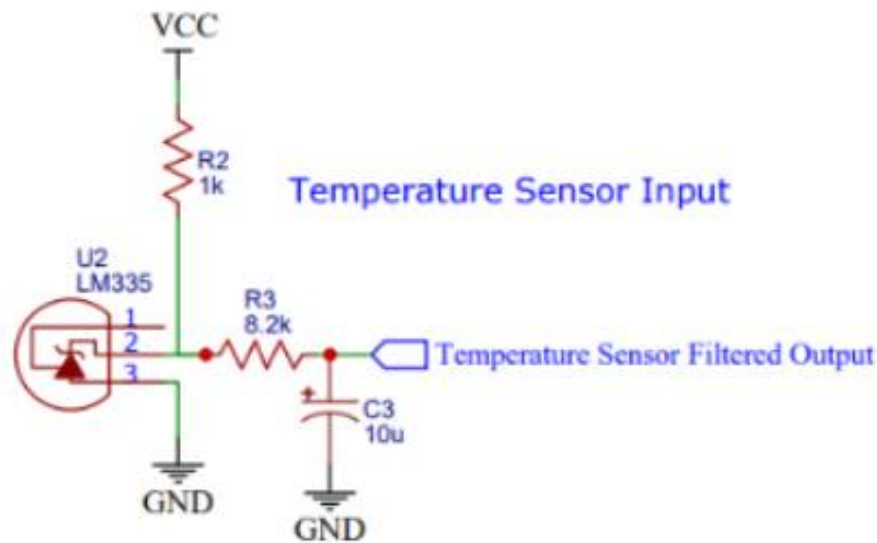
We used our LM335 to measure the ambient air temperature by measuring the voltage across the element using the Arduino Mega's analog to digital converter. The LM335 is similar to a reverse biased Zener diode who's break down voltage varies in a linear fashion roughly at the rate of 10 mV per degree C. Thus, we were able to use that relationship by placing the LM335 in a reverse biased configuration with a 1 kilo-ohm<sup>kΩ</sup> resistor in series to the Arduino's 5-volt power supply bus. We also added an RC lowpass anti-aliasing filter to limit the input frequency content of our temperature signal to 1.94 hertz<sup>Hz</sup> given that a first order lowpass RC filter has a -3 dB cutoff point gives the parameters found from Equation 1.1.

All units in the report must have the same style. I recommend engineering prefixes on abbreviated units, e.g. "10 mV"



$$F = \frac{1}{2 * \pi * R * C} \quad (\text{Equation 1.1})$$

A schematic of the temperature sensing portion of our microcontroller can be found below in Figure 1.1.

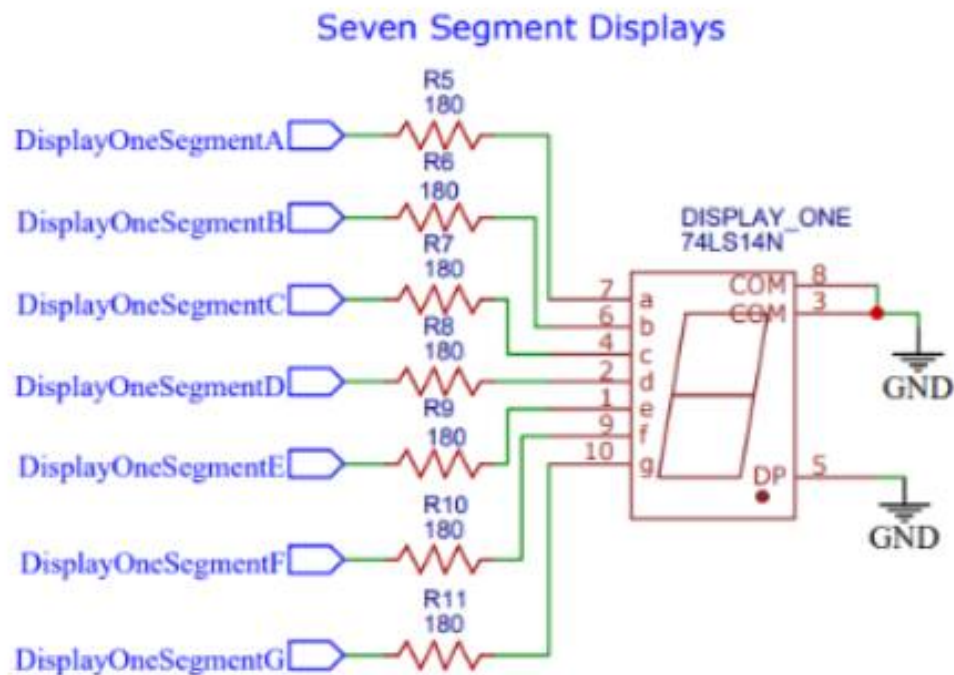


**Figure 1.1. Temperature Sensor Schematic**

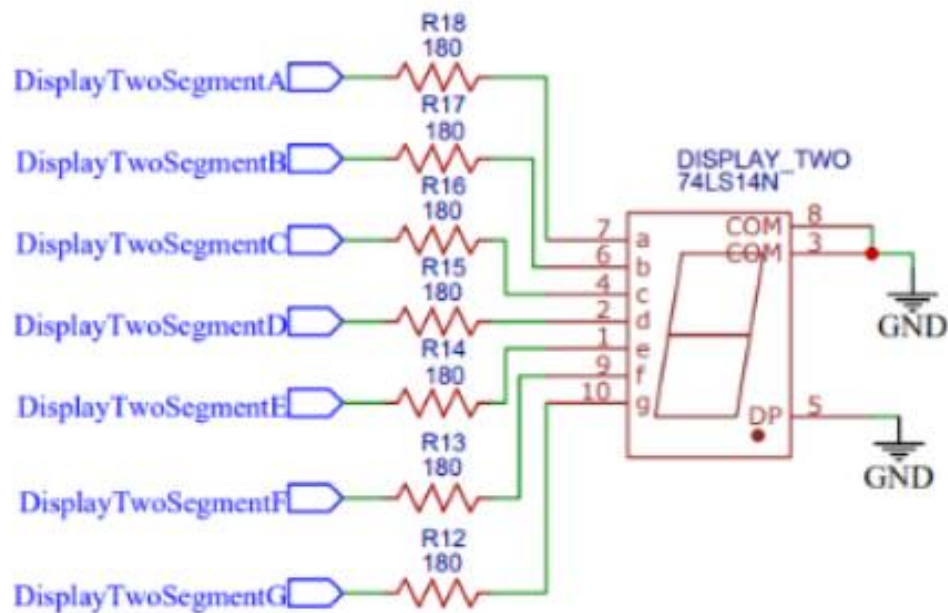
We used the LED 7-segment displays to display the set temperature. We used two push buttons to toggle the set temperature. One button incremented the set temperature up by one degree while the other button incremented the set temperature down by one degree. For each individual segment on the seven-segment display we had to calculate a current limiting resistor to put in series between the output of the Mega and the LED that was being driven. We calculated the resistance of the current limiting resistors using the following Equation 1.2.

$$R = \frac{(V_{cc} - V_f)}{I} \quad (\text{Equation 1.2})$$

The wiring schematics of the resistors and seven-segment displays can be found in Figure 1.2. and 1.3.



**Figure 1.2. 7 Segment Display #1 Schematic**



**Figure 1.3. 7 Segment Display #2 Schematic**

For the button used to toggle the set temperature we used both a Schmitt Trigger and a capacitor to debounce the button. A Schematic of the push buttons, Schmitt Triggers, and capacitors can be found below in Figures 1.4. and 1.5.

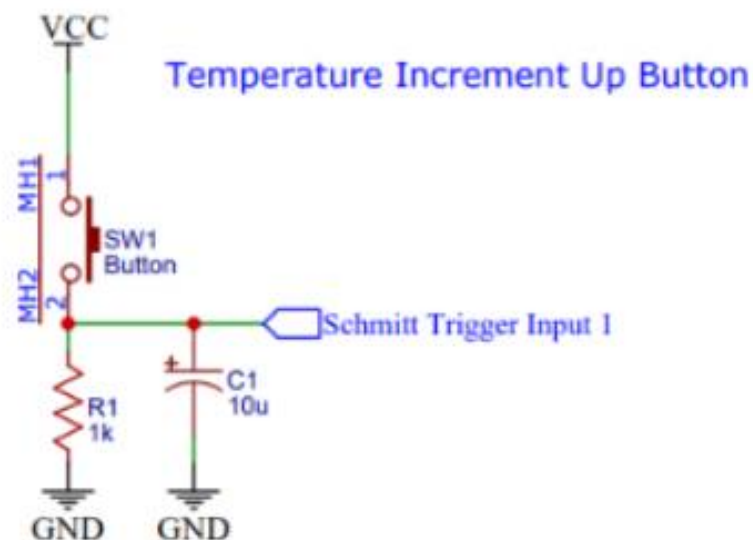


Figure 1.4. Schmitt Trigger Increment Up Schematic

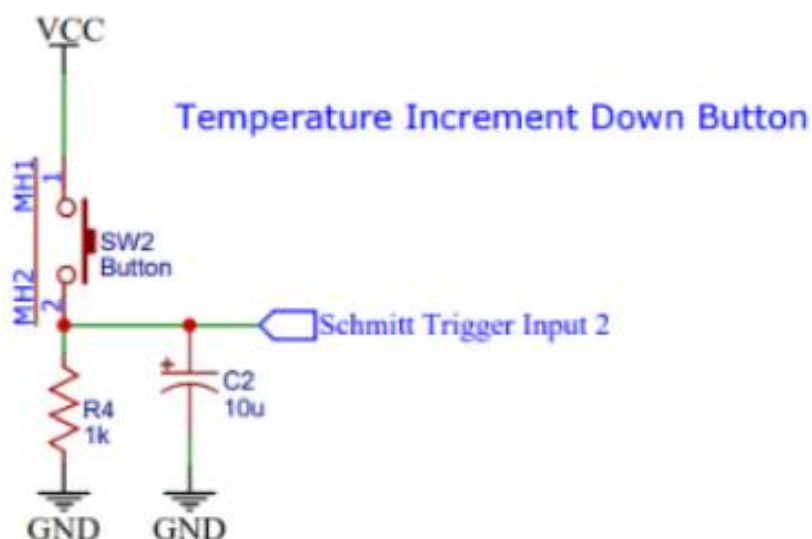
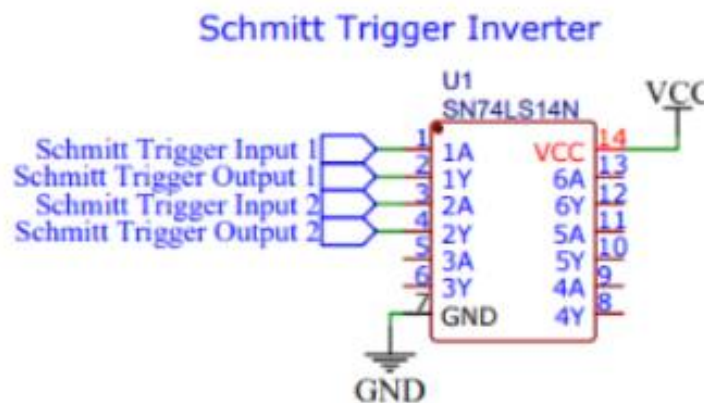


Figure 1.5. Schmitt Trigger Increment Down Schematic



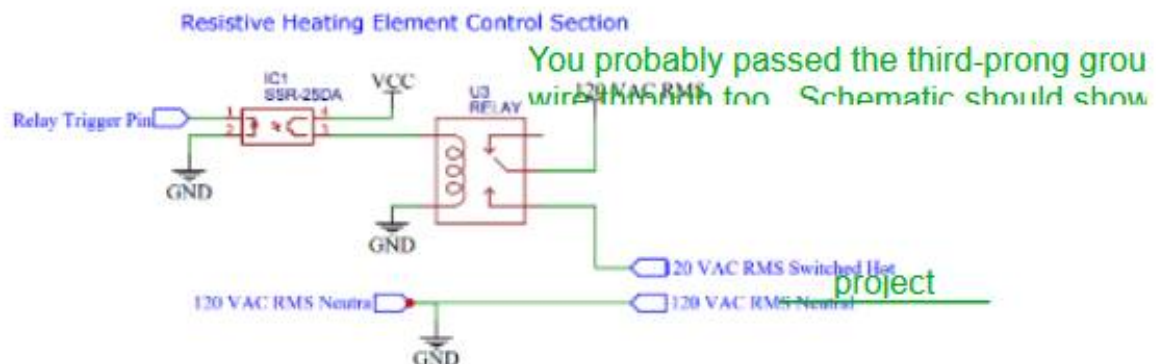
As mentioned before a single integrated chip was used. We used this chip for multiple Schmitt Triggers. The wiring schematic for the integrated chip can be found below in Figure 1.6.



**Figure 1.6. Schmitt Trigger Integrated Chip**

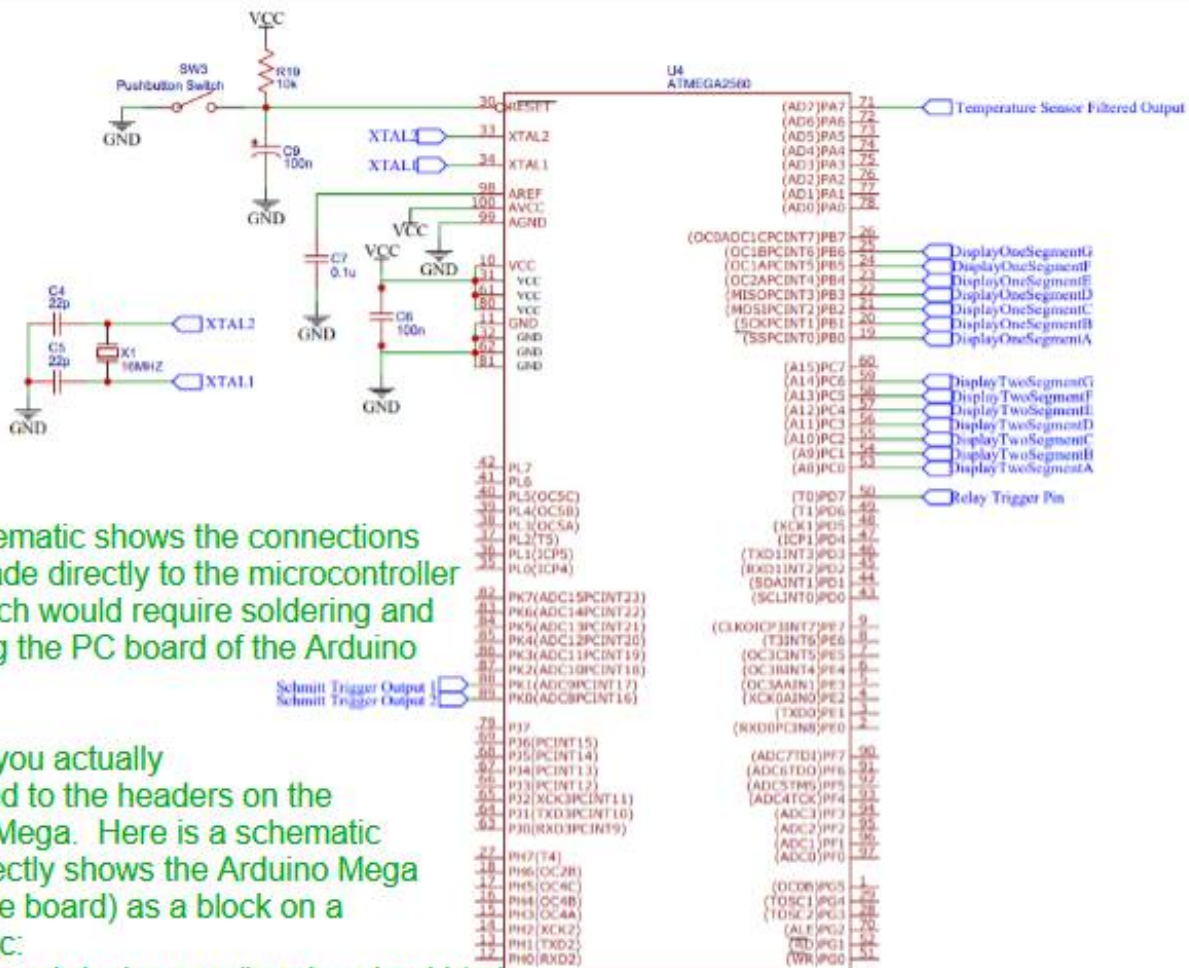
In order to power the hairdryer, we used two relays. Obviously, the Arduino does not put out enough power to run the hair dryer which is powered by 120 VAC. To achieve the end goal of using the Arduino to turn on the hair dryer, we used an output pin from the Arduino to a solid-state relay. From the steady state relay, we then went to a larger relay which then was connected to an outlet. We took a power cord and plugged it into the wall and brought the power to the larger of the two relays. When the Arduino sent a signal to turn the hairdryer on, the first relay stepped up the 5 volts to a larger voltage which then allowed the contacts on the second larger relay to close, which in turn allowed the power to flow from the wall outlet into our own outlet which the hairdryer was plugged into. Both relays and our own outlet were placed into a box to protect from short circuits and arc flashes. A schematic for the wiring of the relays can be found below in Figure 1.7.





**Figure 1.7. Relay Box Wiring Schematic.**

Each of the previous schematics are all a part of the entire microcontroller. All sections are necessary for the microcontroller to function properly. Each schematic has inputs that are connected to the Arduino Mega. Figure 1.8. below shows where all of the inputs and outputs are connected to the Arduino Mega board.



**Figure 1.8. Arduino Mega Inputs/Outputs Schematic**

### III. Software

Our code was relatively simple. Because the Arduino IDE uses a modified version of C++, we were able to use some object-oriented programming to avoid creating repetitive code. We created two separate class types to use for the code. The first class was a class that held the code for getting the temperature from our LM335. The class had two members and three methods. The members defined a calibration voltage for the sensor and an input pin for the sensor. The three methods were: `measureF()`, `measureC()`, and `measureK()`.

These allow for users to retrieve the temperature in either the Fahrenheit, Celsius, or Kelvin unit system.

The second-class type defined was a seven-segment display class which held member variables for the pin assignment of each individual segment on the display. The seven-segment display class used one method other than its constructor which was `WriteSegmentNumber(uint8_t value)` which took in a single unsigned 8-bit integer and displayed it to the seven-segment display assigned to the specific instance of that class. In the setup routine, we simply attached the interrupt service routines to the functions that would increment or decrement a volatile unsigned 8-bit integer that represented the set temperature in degrees Fahrenheit. The loop routine would then continually update and check whether the temperature was within the set allowable range for the <sup>he</sup> Bang <sup>he</sup> Bang controller. If the code determined that the temperature was below the set temperature it would drive the relay trigger pin high which would in turn trigger the hair dryer to run until the temperature on the junction was at least 10 degrees higher than the set temperature at which point it would turn the signal off and continue to monitor the temperature.

#### IV. Results

After uploading our code to the Arduino Mega, we were left with a functioning temperature microcontroller. Although the microcontroller worked, it was not a flawless piece of equipment and an improvement could be made. One issue we were having was the debouncing of the pushbuttons. We attempted to use a Schmitt Trigger and capacitor to ground to create a debounce in hardware. Another option would have been to write code that would act as a filter. We decided to use a Schmitt trigger because it is usually more reliable and does not slow down the code.



There were a few issues in our implementation. Primarily, we encountered a lot of issues with pushbutton bouncing. The pushbuttons used originally were out of date and did not provide a good electrical connection. Even though we were not able to make the debouncing technique work perfectly, once we took away the Schmitt Trigger<sup>1c</sup> and tested the microcontroller without the trigger, we found a difference between when the Schmitt Trigger was in the circuit and when it was not. Which worked better? With the Schmitt trigger or without?

## V. Conclusion

Once the breadboard was wired with all the components, the compiled program was uploaded into the Arduino Mega. The result of the upload was a fully functional temperature microcontroller. When the displayed temperature fell far enough below the setpoint, the relay would turn on from a signal from the Arduino Mega. The relay would then turn the hair dryer on and heat up the ambient temperature around the sensor until it reached a set distance above the temperature set point. The temperature microcontroller functioned by itself without any necessary human interaction.

## VI. References

<sup>i</sup> Solid State Relay Datasheet: <http://www.fotek.com.hk/solid/SSR-1.htm>

<sup>ii</sup> Temperature Sensor LM335 Datasheet: <http://www.ti.com/lit/ds/snis160e/snis160e.pdf>

<sup>iii</sup> 7-Segment Display Datasheet:

<https://media.digikey.com/pdf/Data%20Sheets/Fairchild%20PDFs/MAN6600%20Series.pdf>

<sup>iv</sup> Schmitt Trigger Datasheet: <http://ee-classes.usc.edu/ee459/library/datasheets/DM74LS14.pdf>

<sup>v</sup> Github Repository of Code: <https://github.com/shanetink/Thermostat>

IEEE style is to use integers in braces using regular text (Not italics, not superscript.) This style is not built into MS-Word, but it is easy enough to make MS-Word do it.

\_\_A\_\_ Style

\_\_A\_\_ Completeness

\_\_A\_\_ Accuracy

Overall report grade: A



# LCD and RTC-Based Arduino Clock

Nolan Vande Griend and Stefan Walicord

Professor De Boer

EGR 304 Lab, Dordt University

April 29th, 2020



## I. Introduction

This heading appears on the wrong page.

First-page header is missing.

Learning to integrate analog data and output displays in an embedded system is an invaluable skill for an engineer to have in our current technological age. To achieve this, Lab Project 1's requirement is to use a microcontroller or SoC (system-on-a-chip) to process an analog phenomenon using a sensor, and to use at least one display output. This team chose to use an Arduino Uno with a 16x2 LCD (liquid crystal diode) display and a DS1307+ RTC (real time clock). The intent is to have the RTC feed the current time to the Arduino, which then displays it on the LCD. Due to what are likely library and hardware roadblocks, the project is not successful at the time of this report's submission.

Banal. See example introductions in my booklet, "How to Write a Laboratory Report"  
<https://dordt.instructure.com/courses/2662877/files/165368760/download> pages 15, 16.

## II. Method

The project began with a significant period of research and planning that produced many resources for implementing the clock system on Arduino (see Resources). The team also ordered the LCD and RTC needed for the project (see Hardware), but because of an early oversight, the LCD purchased was only I<sup>2</sup>C (two wire) protocol capable. This meant the Arduino, RTC, and the LCD all needed to be connected with I<sup>2</sup>C, as opposed to using a 4-bit connection option that almost all tutorials used to interface the Arduino and LCD. The team connected both the SDA and SCL ports of all three devices to a bus node that had a 10 <sup>k $\Omega$</sup>  resistor running to ground. This allowed the Arduino to speak to both devices and see their I<sup>2</sup>C addresses.

Use the symbol menu to get a proper display of the "ohms" symbol. (The equation editor can also be used for this.)

Difficulty arose when the LCD did not react to common LCD libraries for Arduino. The vendor had not listed a library, blocking the team's ability to send commands to the LCD. The correct library was eventually found on the manufacturer's page and installed. A potentiometer was used so that the LCD screen would not burn out after running for an extended period of time.

Running header with page numbers, date, paper title is missing.

The RTC integration suffered from missing libraries and a forgotten quartz oscillator. After a separate quartz crystal (see Hardware) was acquired and installed as per specification, the RTC continually failed to respond to any commands, despite rewiring. Eventually, a somewhat functional library was installed that let the RTC give a simple text output that was written to the LCD (see Results).

### III. Results

The Arduino code was able to successfully write to the LCD screen while also running through both the RTC and quartz crystal. However, the RTC never transmitted the correct time to the Arduino, meaning timekeeping was impossible. After one final library switch, the number “22” kept printing repeatedly on the LCD screen without changing. At first this suggested that the RTC knew the current date; however, it did not change with the passage of time, meaning there was not conclusive evidence that the RTC was functioning. In summary, the Arduino could communicate with both the LCD and RTC, but the RTC failed to supply useful time data, stymying the project.

### IV. Conclusion

Neither the project requirements nor the team’s plans were met. Reading time from the RTC, which was the analog sensor requirement, could not be accomplished. The requirement for a display was achieved, but the LCD had nothing to display. The lack of lab access due to the 2019-2020 coronavirus quarantine cut the team off from hands on professorial help and critical oscilloscope equipment, forcing them to abandon the project without having achieved timekeeping.



The project showcased the complexity of working with unique plans that deviated from helpful internet guides, as shown by the I<sup>2</sup>C struggles. It also highlighted the difficulty of correctly understanding and purchasing hardware, such as buying an LCD with 4-pin capability or an RTC in a complete package including the quartz crystal and a battery, neither of which the team realized were important beforehand. Integrating disparate systems like the Arduino, LCD, and RTC with the appropriate libraries proved to be a difficult task. While a timekeeping apparatus was not created, the constant roadblocks and failures on the team's part highlighted the importance of planning ahead and researching products rigorously before buying the parts. This is an important lesson for future projects: understand what you need and what you are buying, and also ensure that the hardware and libraries requisite for proper integration in your project exist.

## V. Hardware

1. Arduino Uno w/ USB cable
2. 16x2 Sseed Studio Black on Yellow LCD, Mouser #713-104020113
3. DS1307+ Maxim Integrated RTC, Mouser #700-DS1307
4. 32.768 kHz Crystal by ECS Inc., Digi-Key Part #X1124-ND, Mfg. Part #32.ECS-.327-12.5-13X
5. x2 10 kOhm 5% resistors
6. 2 kOhm potentiometer
7. Breadboard

## VI. Resources

[https://www.youtube.com/watch?time\\_continue=350&v=xVC0X\\_PEXE&feature=emb\\_title](https://www.youtube.com/watch?time_continue=350&v=xVC0X_PEXE&feature=emb_title)

<https://www.arduino.cc/en/Reference/Wire>

<https://www.arduino.cc/en/Tutorial/MasterReader>

<https://electronics.stackexchange.com/questions/25278/how-to-connect-multiple-i2c-interface-devices-into-a-single-pin-a4-sda-and-a5>

<https://www.digikey.com/catalog/en/partgroup/32-768khz-crystals/4707>

<https://forum.arduino.cc/index.php?topic=576597.0>

<https://forum.arduino.cc/index.php?topic=569071.0>

<https://www.electronics-lab.com/project/real-time-clock-20x4-i2c-lcd-display/>

<https://www.instructables.com/id/Interfacing-DS1307-I2C-RTC-With-Arduino/>

<https://playground.arduino.cc/Main/I2cScanner/>

<https://www.seeedstudio.com/Grove-16x2-LCD-White-on-Blue.html>

<https://www.mouser.com/datasheet/2/256/DS1307-1513036.pdf>

<https://www.circuitlib.com/index.php/tutorials/product/91-arduino-based-alarm-clock>

<https://circuitdigest.com/microcontroller-projects/arduino-alarm-clock>

[https://www.electronicshub.org/arduino-alarm-clock/#Components\\_Required](https://www.electronicshub.org/arduino-alarm-clock/#Components_Required)

<https://create.arduino.cc/projecthub/the-underdog/arduino-alarm-clock-3388b0>

<https://maker.pro/arduino/projects/arduino-alarm-clock-using-real-time-clock-lcd-screen>

## VII. Appendix

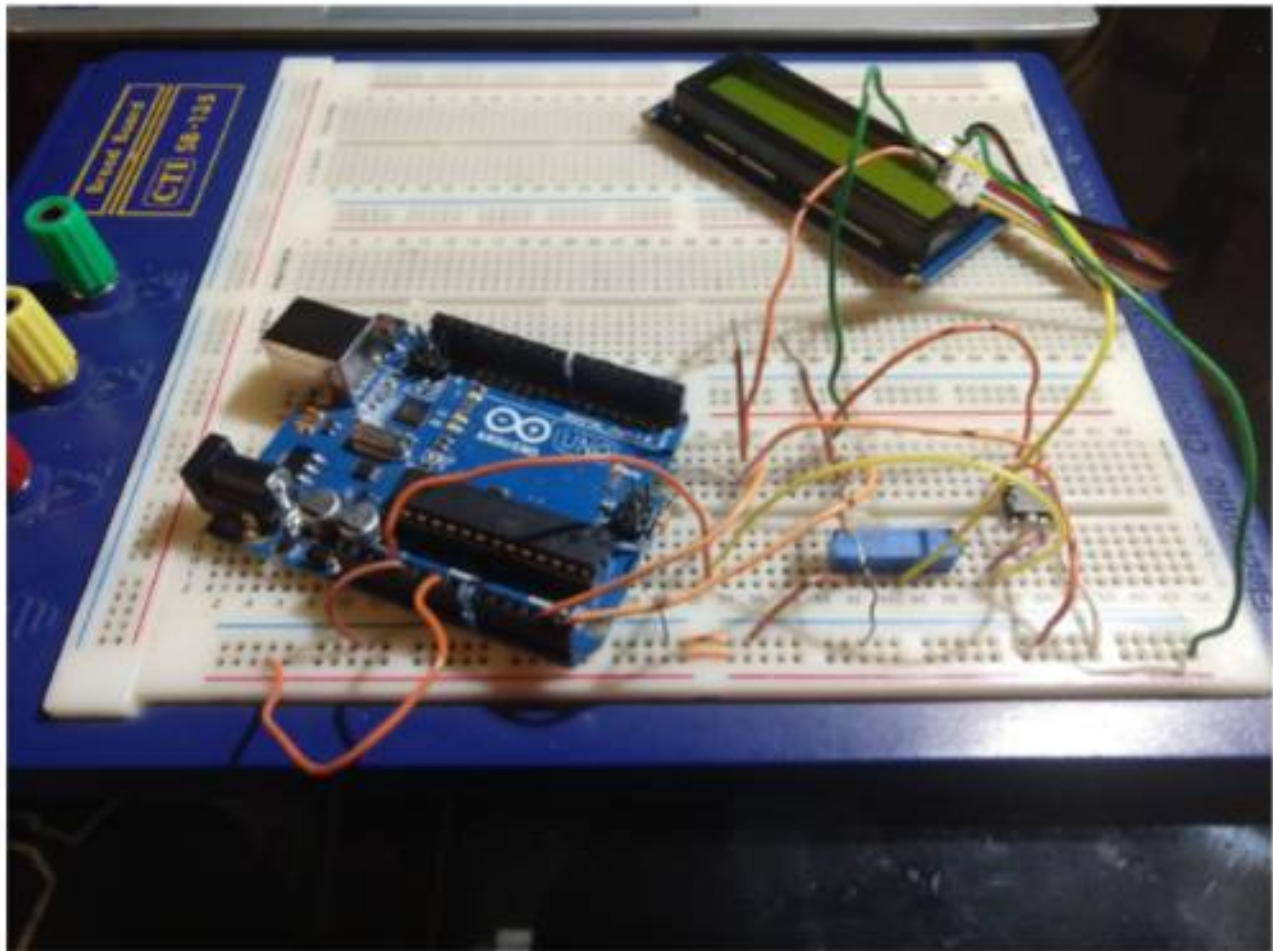


Figure 1: Project Hardware Set-Up

Figure 1 is somewhat helpful, but a SCHEMATIC would be much more helpful.

Learning to think symbolically in the style encouraged by looking at a schematic would probably have helped a lot on this project, especially in the early parts when you had not yet recognized the merits of a 4-wire display connection vs. the I2C connection. (Both types of connections are valid and useful, but not realizing your situation early on caused a lot of delay.)

Grading:

- B   Style (The elements of IEEE style show, but there are several deficiencies)
- C   Completeness (No schematic--incomplete diagnostics of the problems.)
- A   Accuracy (A good grade here mainly because what is written is true.)

Overall Report Grade: B-



```

void setup() {

  pinMode(BACKLIGHT_PIN, OUTPUT);
  digitalWrite(BACKLIGHT_PIN, HIGH);
  lcd.begin(16,2);
  lcd.setCursor(2,0);

  Serial.begin(9600);

  Wire.begin();

  rtc.begin();

  if (!rtc.isrunning()) {

    lcd.print("RTC is NOT running!");

    // following line sets the RTC to the date & time this sketch was compiled
    rtc.adjust(DateTime(__DATE__, __TIME__));

  }

}

```

Figure 2: Test Code Set-Up at End of Project (Non-conclusive)

```

void loop() {

  DateTime now = rtc.now();

  char buf[100];

  strncpy(buf, "DD.MM.YYYY hh:mm:ss", 100);

  lcd.print(Serial.println(now.format(buf)));

  delay(1000);

}

```

Figure 3: Looping Scan to Display to LCD Screen (Functional, but did not display properly)

# HEARING LOOP MAGNETIC SENSOR AND STRENGTH DETECTOR

Ty White

Ryan Zevenbergen

March 25, 2020





# Hearing Loop Magnetic Sensor and Strength Detector

Ty White and Ryan Zevenbergen (The LoopyTunes)

March 25, 2020

## I. INTRODUCTION

The Hearing Loss Association of America reports that 48 million Americans suffer from some form of hearing disability (Blackwell, et al). Many of these citizens require some form of hearing aid to communicate reliably. Many institutions have been implementing magnetic hearing loops to send electromagnetic signals directly to the hearing aid. For best performance, it is necessary to monitor the strength of the field generated by these hearing loops so that the levels can be adjusted. The goal of this lab was to design and build a microcontroller circuit to perform the monitoring purposes mentioned above. The Arduino provided the perfect interface for the detector circuit, as it is programmable and has both input and output pins. An inductor was used to detect the presence of a magnetic field. The Arduino determines the strength of the magnetic field based on the voltage difference across the inductor after amplification. Once the signal has been processed, it is outputted to a scale of light emitting diodes (LEDs). The LEDs are color coded to display the strength of the field in an intuitive manner. The whole circuit has been calibrated according to International Electrotechnical Commission (IEC) standards. The final circuit was tested by Professor De Boer, who confirmed that the results are accurate. One area of improvement comes in the system response time. Currently, the time constant is at an acceptable level in order to gain an accurate average value, but a loading effect exists meaning the system reacts slowly to rapid changes in signal intensity. For more information on hearing loops and similar devices, see the Appendices at the end of this report.

use a different word,  
such as "sent."  
"Outputted" is awkward.

## II. METHODS

### a. Idea and Proposal

After discussing several options with Professor Douglas De Boer, the hearing loop detector and level sensor became the definitive project. This design was decided on because it is a project that serves some useful purpose for Dordt University. Professor De Boer also required that a proposal document be written to approve of the project. The proposal was written so that the team could progress to the conceptual design phase.

### b. Conceptual Design

Professor De Boer provided informational resources for the design of the circuit. These documents were read and reviewed for background research for a better initial design. Next, the LED color code was decided upon, as well as the standard levels for hearing loops. A testing apparatus was also proposed in the conceptual design phase. Finally, the initial flow of the Arduino program was created.

## c. Calculations

The voltage drop across the inductor is on the order of millivolts, so amplification was necessary to properly scale it for the Arduino to read. To do this, the design utilized an operational amplifier (op amp) circuit. The gain formula for an inverting amplifier circuit, the type of amplifier used in the detector, is shown in Equation 1.1 below.

$$GM = -\frac{R2}{R1} \quad \text{Eq. 1.1}$$

$R1$  for the design was set at  $10\ \Omega$  and  $R2$  was a potentiometer ranging from  $10\ k\Omega$  –  $1\ M\Omega$ . This gives a variable gain in the range of 1,000-100,000. The gain range was determined to be a suitable level for amplification.

In addition to gain, the circuit required a half-wave rectifier to determine the average value of the incoming signal. The half-wave rectifier has a time constant, and it needed to be sufficiently large to gain an accurate reading. The calculation for the time constant is given below.

$$\tau = RC \quad R = 51\ k\Omega \quad C = 1\ \mu F \quad \text{Eq. 1.2}$$

$$\tau = 51\ k\Omega * 1\ \mu F = 0.051\ s$$

This time constant is sufficient for the purposes of the design.

## d. Design Implementation

The circuit was constructed after the conceptual design was completed. Fourteen LEDs are laid out on a breadboard. Eight of these LEDs are green in color, specifying an acceptable level of the magnetic field. Four LEDs are yellow in color, signifying that the magnetic field level is on the fringes of the acceptable standards. Two of the LEDs are red. These will activate when the magnetic field is not within the IEC standards. Finally, no LEDs will be displayed if no magnetic field is detected. Each of the LEDs is connected in series with a  $130\ \Omega$  resistor. These are then plugged into each of the fourteen digital pins of the Arduino. The inductor is connected to an LM741 op amp in the inverting input. In addition, a resistor is connected to the non-inverting input to pull down the bias. This concept is discussed more in the RESULTS section. A potentiometer acts as the gain control for the amplifier circuit. A half-wave rectifier is connected to the output of the amplifier to read the DC average level of the voltage. The DC average acts as the signal and is plugged into the analog input of the Arduino. The Arduino then maps the  $0 - 5\ V$  signal to a  $0 - 1023$  scale. Depending on the level in this range, an output LED is activated, giving the reading to the user. See Appendix II for the Arduino code used.

The average value of the signal is zero. Your schematic shows that your circuit is peak responding. "CAP 108" will charge rapidly through the diode, then discharge slowly through the resistors to the right of the capacitor.



### III. RESULTS

#### a. Experimental Data

To test the design, 120VAC to 12VAC transformer was used. The transformer converts an electrical signal into a magnetic signal through an iron core to another electrical circuit that reverts the magnetic signal into a lower voltage signal. This transformer was held in close proximity to the hearing aid of Professor De Boer to assure it worked. It was found through experimental testing that the loudest allowable magnetic signal transmitted from the transformer was less than or equal to one half of an inch. Any closer and the magnetic signal was intolerable.

Specifically, it caused saturation distortion inside my hearing aid.

When the signal was initially inputted to the device, the Serial Monitor showed an analog signal of 0. After realizing that an AC signal needs to be converted to a DC signal, a half-wave rectifier was implemented. Unfortunately, a negative DC bias was introduced with the op amp due to leakage current raising the floating ground voltage. As a result of this phenomenon, a Thevenin equivalent resistance of R1 and R2 was added to the non-inverting input of the op amp. The Thevenin equivalent resistance was calculated experimentally with a potentiometer, as the exact resistance of the R2 potentiometer was unknown. Unfortunately, this did not completely solve the negative DC bias issue. Briefly, a second op amp was added to introduce a positive DC bias to balance it out. After a few tests, this idea was abandoned due to inconsistent results. Eventually, the potentiometer connected to the non-inverting input had its value further refined and adjusted. This solved the issue.

Like "outputted," "inputted" is an awkward word. Try, "when the signals was initially connected. . ." or similar.

To protect the Arduino input pin from over-voltage, a 5V regulator was connected to the output signal of the half-wave rectifier. Unfortunately, this was the incorrect component for the job. After consulting Professor De Boer, the 5V regulator was replaced with a 4.7V Zener Diode. This introduced a maximum threshold that could be inputted to the Arduino input pin.

Not included in the schematic or in the photographs.

With the final design complete, it was tested once more with the transformer transmitter. The signal interpreted by the Arduino receiver at one half inch or closer was coded to be the high threshold value. Using this iteration, the design correctly displayed the intensity of the magnetic signal. If the signal was too weak, it would not be displayed on the LEDs. As the signal strength was increased, it would be shown properly on the display up until the signal was too strong and exceeded the highest tolerable value. Once the signal reached this limit, the red LED at the top of the display would remain illuminated until the signal strength was reduced.

"applied"



## b. Schematics and Figures

Below is an example of an unacceptable signal because it exceeded our maximum allowable threshold. The magnetic field is too strong, thus the red LED on the right indicates that it is unacceptable.

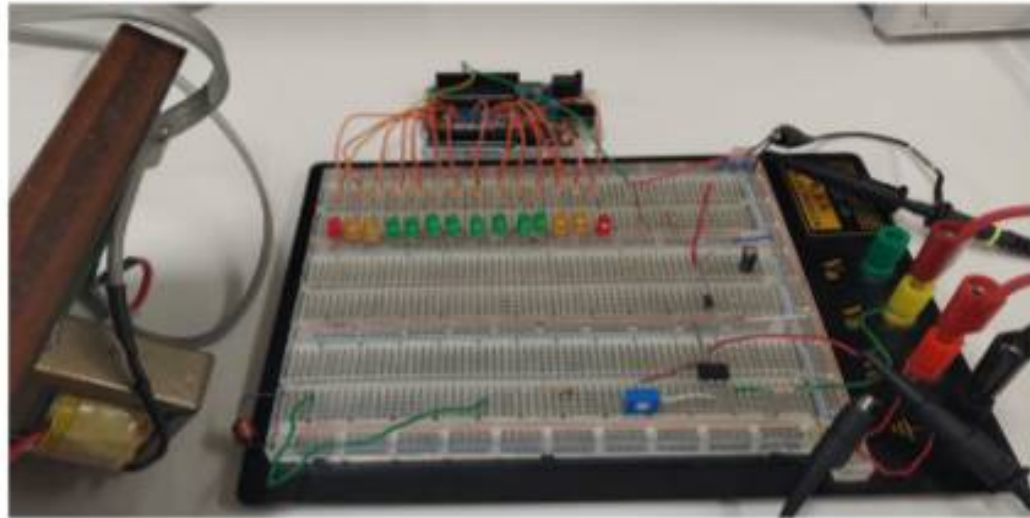


Figure 1.1

The graph below is the illustration showing the signal that is too strong. The yellow line illustrates the input to the Arduino is above the 4V threshold limit that was determined experimentally with Professor De Boer's hearing aid.

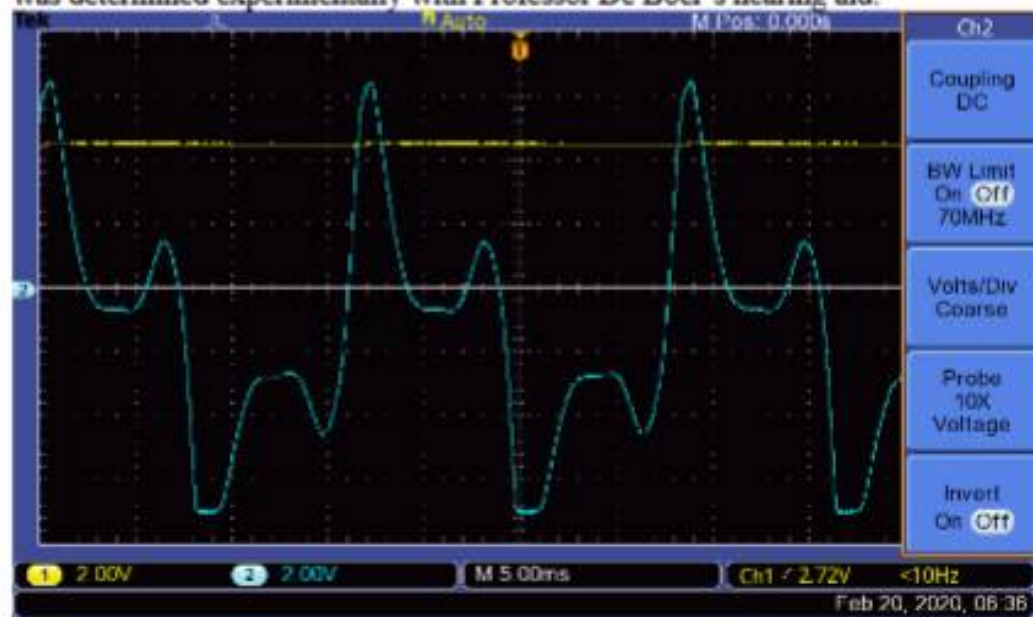


Figure 1.2 Caption should explain which node each channel is displaying. Since this info is missing, the figure is a mystery to your readers.

Below is an example of the device reading an acceptable level of the magnetic field generated by a 60 Hz sine wave. Notice how the green LED is active, demonstrating the acceptable level of the magnetic field.

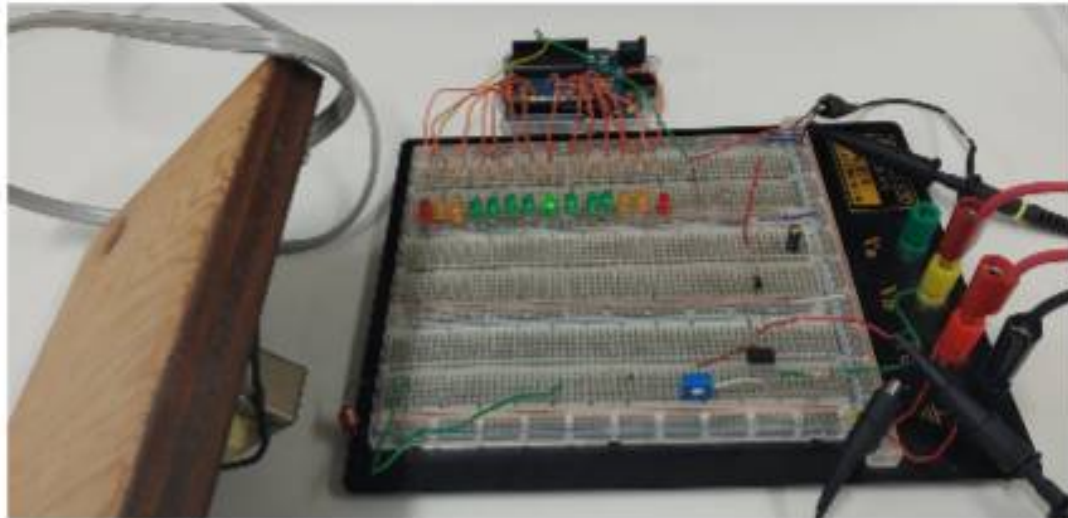


Figure 1.3

Illustrated below is the resulting oscilloscope screen from the acceptable measurement. The blue trace is the signal after amplification, and the yellow trace is the rectified average that the Arduino receives.

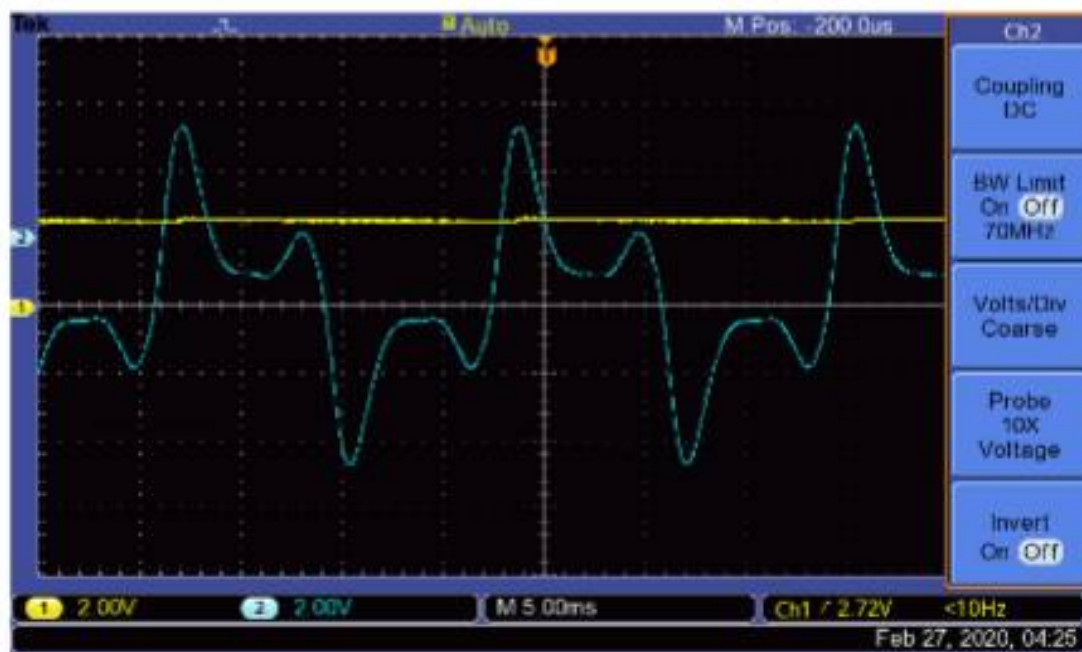


Figure 1.4

Next is an acceptable signal. The transformer's distance from the inductor has been increased, therefore the magnetic field is weaker. Notice how the yellow LED is active, demonstrating a level near the limit of the acceptable range.

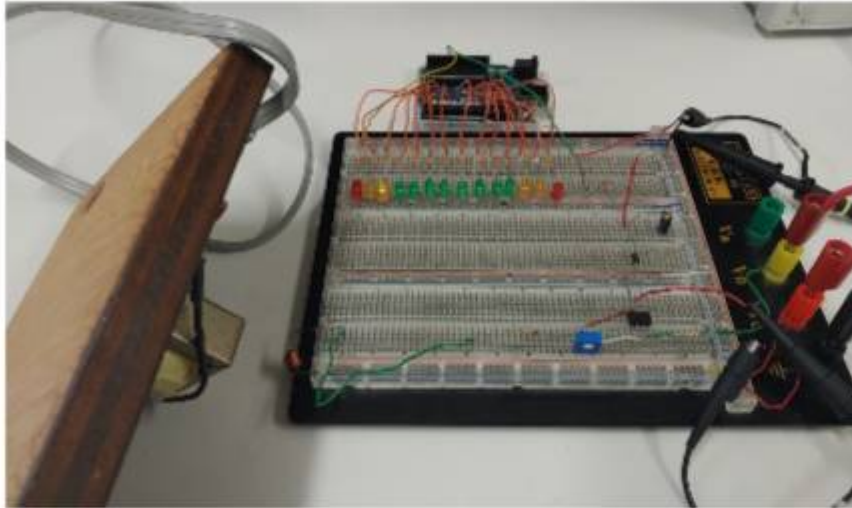


Figure 1.5 "A weak field illuminates a yellow LED."

Captions must include description.

The resulting oscilloscope output for this observed measurement can be found below. Again, the blue trace is the amplified signal; the yellow is the rectified average.

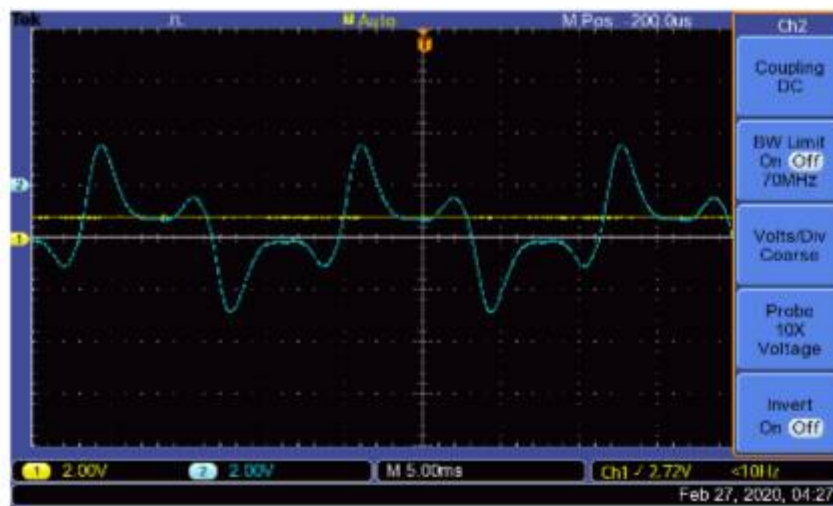


Figure 1.6

"Shown in Figure 1.5. . ."

Absolute references to figures, tables, equations, are almost always better than relative references.



The final set of figures demonstrates an unacceptable signal. The apparatus was programmed such that no LEDs would activate if no field is detected. The circuit can detect the weak signal at this distance, but the signal magnitude is below the acceptable threshold. Notice how the red LED is active.

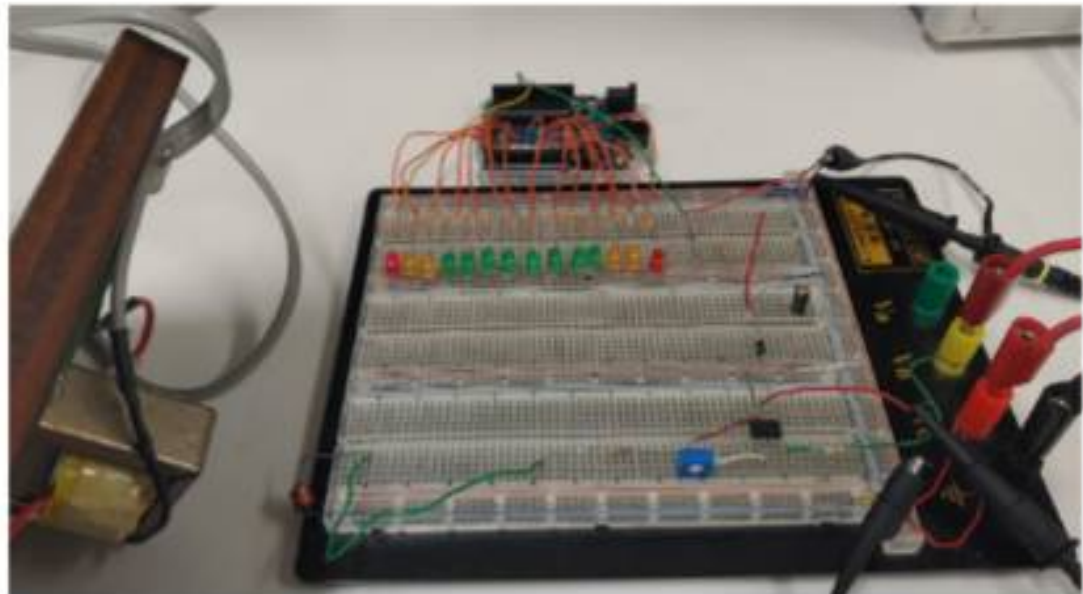


Figure 1.7

Below is the illustration of the unacceptable reading on the oscilloscope.

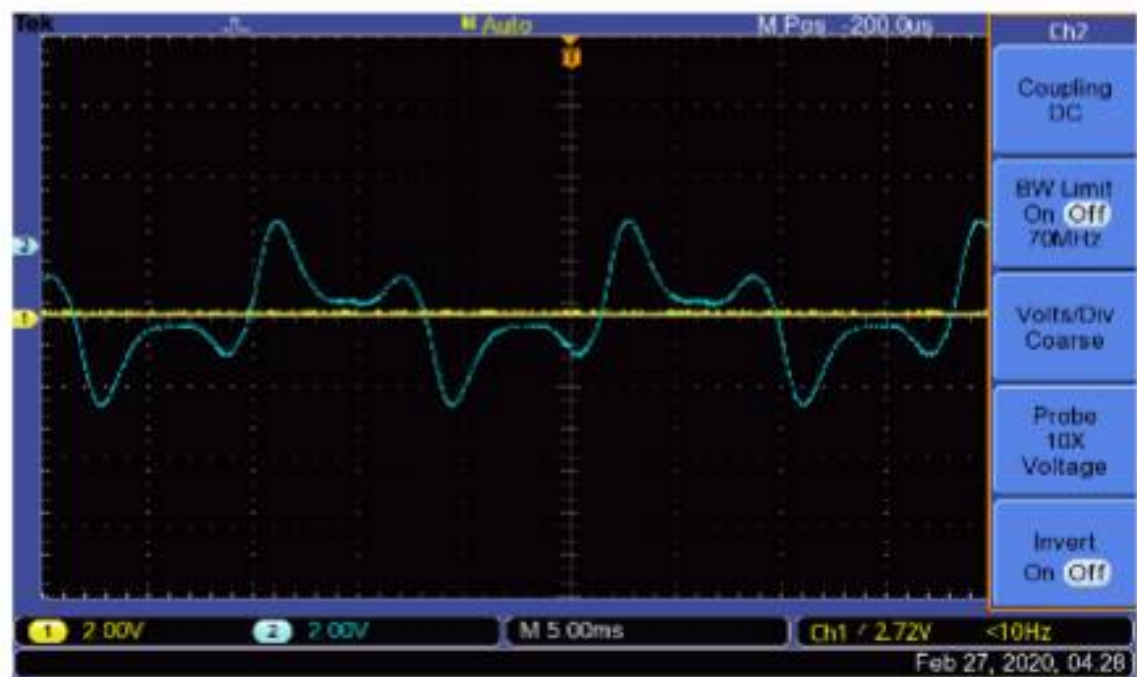
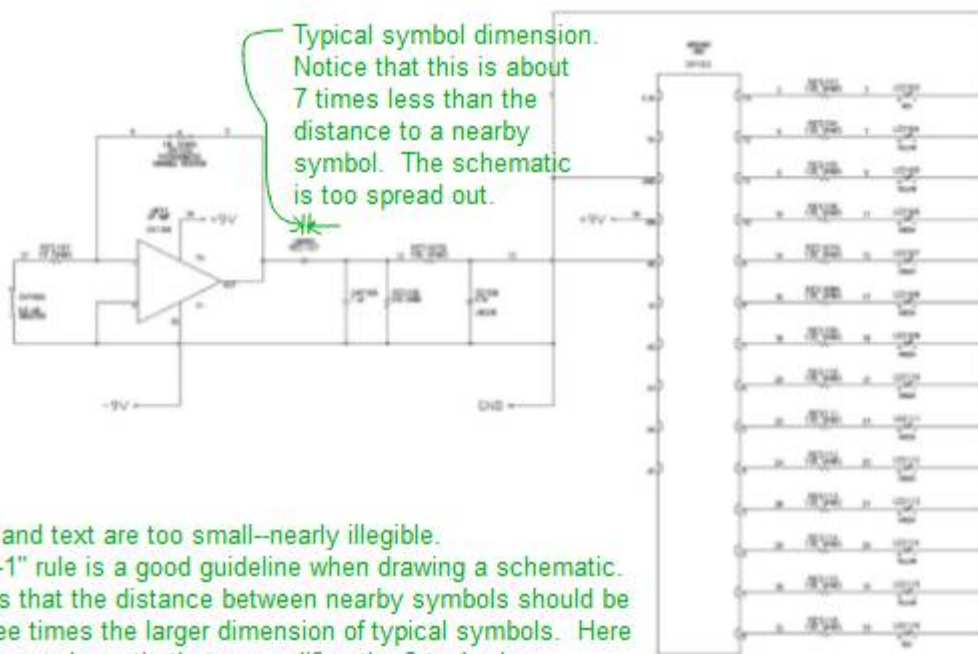


Figure 1.8

In addition, a full schematic of the circuit is provided below.



Symbols and text are too small—nearly illegible.  
The "3-to-1" rule is a good guideline when drawing a schematic.  
The rule is that the distance between nearby symbols should be about three times the larger dimension of typical symbols. Here is a link to a schematic that exemplifies the 3-to-1 rule.

[http://s3.media.squarespace.com/production/1275805/14967857/blog\\_images/w7zoi\\_eagle.png](http://s3.media.squarespace.com/production/1275805/14967857/blog_images/w7zoi_eagle.png)

#### IV. CONCLUSION

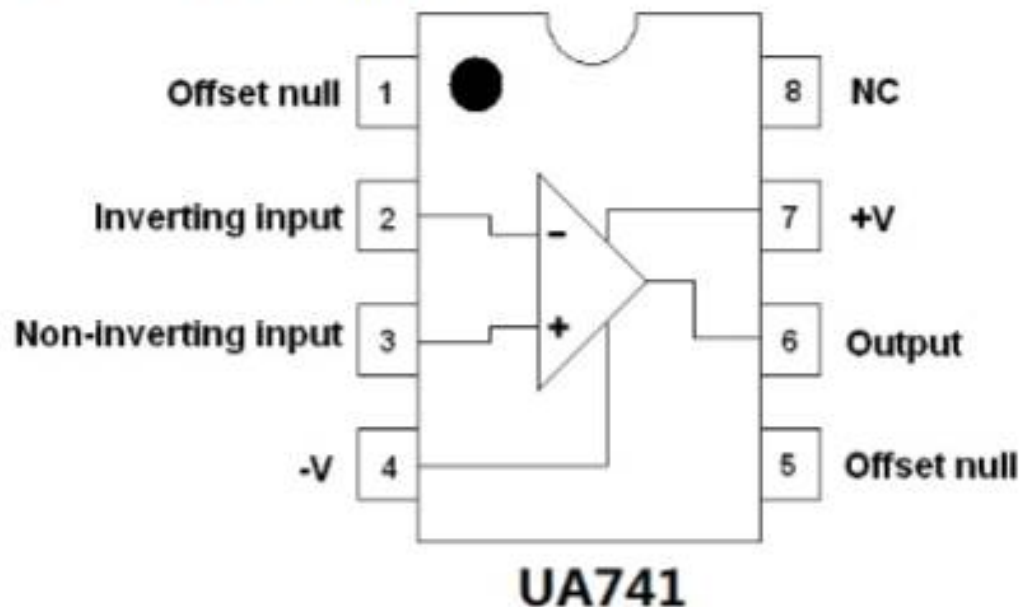
In conclusion, The Hearing Loss Association of America is doing great things for the hearing-impaired persons across America. Having a scale like this would quickly allow someone without a hearing aid to determine the quality of the hearing loop system. Too weak or too strong of signals make it difficult to understand the signal that is transmitted. These signals are displayed with red LEDs to show unacceptable transmissions. Signals that are acceptable but not ideal are displayed with yellow LEDs. Finally, signals securely within the IEC standards are displayed with green LEDs. Using a device like this out in the field allows the sound team of an auditorium to interact and set up a hearing loop system without the need for a "guinea pig" hearing-impaired person to relay their reception quality of the signal.

... "Shown in Figure 1.5. . ."

Absolute references to figures, tables, equations, are almost always better than relative references.

## V. APPENDICES

## a. Appendix I: 741 Op Amp Pinouts



Grading:

A- Style (Quite a number of variances from IEEE style)

A- Completeness (Important information missing or obscured in schematics, and in the figure captions.

A Accuracy (I decided to overlook the statement about "average value" of the signal. It is a bit of a detailed matter.

The lab work I observed was first-class. However this report, while being very good, does not do full justice to the lab work done. In the future, improving your already-good writing skill will pay dividends, especially in a grad-school or on-the-job environment.

Overall report grade: A



## b. Appendix II: Arduino Code

```
1.  int speech_input = 0;
2.  int leds_on = 0;
3.
4.  void setup() {
5.      pinMode(A0, INPUT);
6.      pinMode(0, OUTPUT);
7.      pinMode(1, OUTPUT);
8.      pinMode(2, OUTPUT);
9.      pinMode(3, OUTPUT);
10.     pinMode(4, OUTPUT);
11.     pinMode(5, OUTPUT);
12.     pinMode(6, OUTPUT);
13.     pinMode(7, OUTPUT);
14.     pinMode(8, OUTPUT);
15.     pinMode(9, OUTPUT);
16.     pinMode(10, OUTPUT);
17.     pinMode(11, OUTPUT);
18.     pinMode(12, OUTPUT);
19.     pinMode(13, OUTPUT);
20. }
21.
22. void loop() {
23.     //speech_input is the raw input from the inductor scaled
24.     //between 0-4V (0-818 analog).
25.     speech_input = analogRead(A0);
26.
27.     //leds_on quantizes speech_input into what specific light
28.     //will be on for what voltage input received. LED 13 is a
29.     //strong loud signal while -1 means no signal received.
30.     leds_on = map(speech_input, 0, 818, -1, 13);
31.
32.     if(speech_input >= 818) {
33.         leds_on = 13;
34.     }
35.
36.     for (int i = 0; i <= 13; i++) {
37.         if(i == leds_on){
38.             digitalWrite(i, HIGH);
39.         } else {
40.             digitalWrite(i, LOW);
41.         }
42.     }
43.     delay(10);
44. }
```

## VI. REFERENCES

- Blackwell, D L, J W Lucas, and T C Clark. "Quick Statistics About Hearing."  
National Institute of Deafness and Other Communication Disorders. U.S.  
Department of Health and Human Services, October 5, 2018.  
<https://www.nidcd.nih.gov/health/statistics/quick-statistics-hearing>
- Clarke, John. "Hearing Loop Receiver." *JCAR Electronics*, September 2010.
- International Standard 60118-4*. 2nd ed. Geneva, Switzerland: International  
Electrotechnical Commission, 2006.