B EE 425 - Ohmmeter Design Project

(Autumn 2023)

by

Leon Nguyen - 2278288

&

Rui qi Huang - 1800920

Lab Due Date: 12/11/2023

Introduction

We created an auto-ranging ohmmeter for this design project using a constant current source. Below, in the theory of operations, we explain how the ohmmeter worked. For the ohmmeter project, we were expected to create an auto-ranging ohmmeter that can measure the resistance of 1Ω to 1MΩ. The precision from 10Ω to 100kΩ was expected to be 0.1% and the precision for 1Ω to 10Ω and 100kΩ to 1MΩ was expected to be 1%. The power supply we were expected to use was either a wall wart or a battery. The input voltage we can use is 5V to 12V. For our project in particular, we used only 5V from our Arduino to power everything.

Theory of Operation:

The auto-ranging ohmmeter uses a constant current source and Ohm's law to determine the resistance. For our ohmmeter, we used the constant current source of LM334Z, which was recommended by our classmate Aleksandra. Because the LM334Z has a temperature coefficient, we used the zero-temperature coefficient circuit given in the LM334Z datasheet to eliminate most of the temperature coefficient. To make the ohmmeter auto-ranging, we used relays as switches so we could change the constant current depending on our load. Due to our spec's expected precision, we also used a 16-bit ADC, the ADS1115, to convert our analog voltage to digital for the Arduino to read.

Procedure and Process

To start the design, we have to decide on how to create our constant current source. We decided to use the LM334Z by National Semiconductor recommended by our classmate Aleksandra for our constant current source. To test the LM334Z, we tested the LTSpice model also provided by Aleksandra to see how the overall circuit is going to work. You can see the LTSpice model in Figure 1 below.

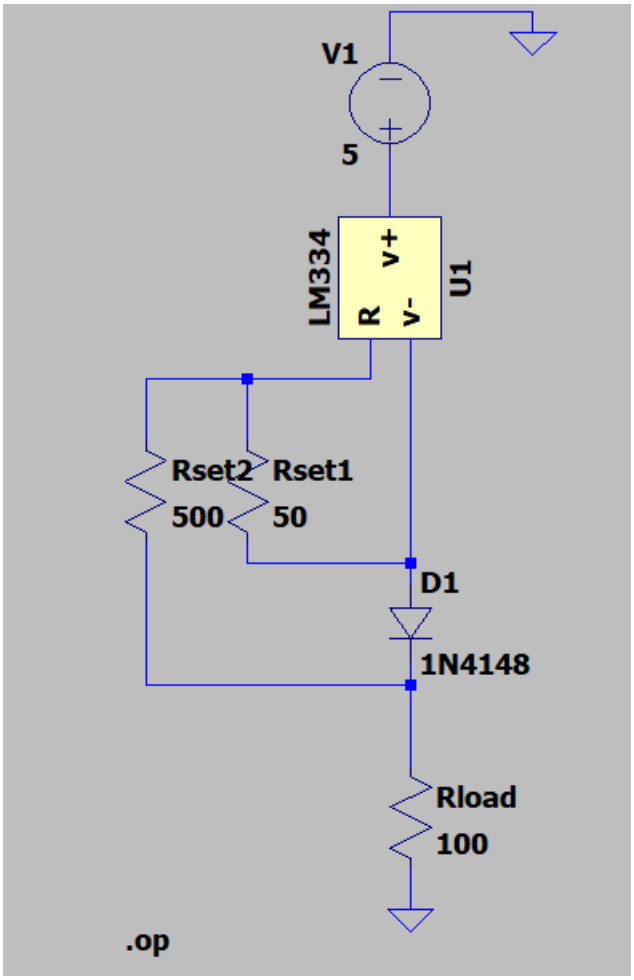

Figure 1 - LTSpice Schematic of Zero-Temperature Coefficient LM334Z circuit

For the overall circuit design, we used the zero-temperature coefficient circuit shown in the datasheet of the LM334Z by National Semiconductor. We also reference the datasheet to decide on the values for Rset1 and Rset2. The datasheet is provided as reference [1] in the reference section of this lab. According to the datasheet:

$$R2 = 10 \times R1$$

After testing the circuit and better understanding how the LM334Z works, we were able to physically test the circuit in the lab. In Figure 2, you can see the circuit we created to replicate the circuit on a breadboard.

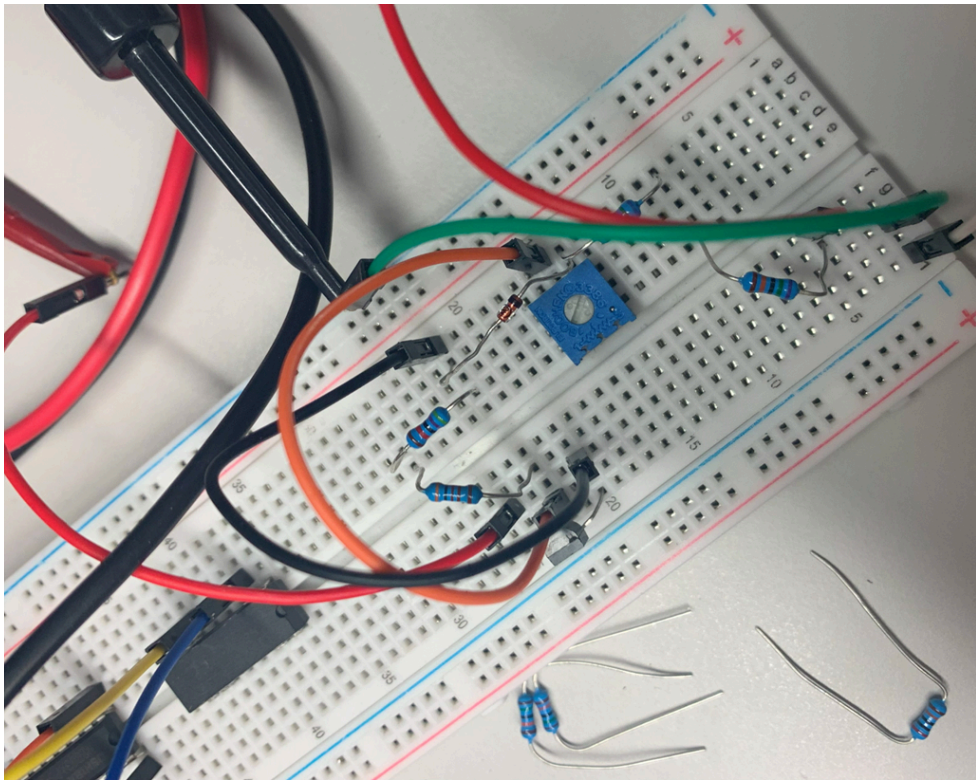

Figure 2 - Zero-Temperature Coefficient Constant Current Source Circuit

During this, we were able to collect data on what set of resistors we needed for different load resistor values. You can see the data collected in Table 1.

Table 1 - Measurements for Zero-Temperature Coefficient Constant Current Source Circuit (Version 1)

R1	R2	Current	Min	Max	Highest Drop
50	500	2.788mA	1	1.2k	3.316V
500	5k	164.9uA	1k	22k	3.629V
5k	50k	15.2uA	22k	220k	3.302V
40k	400k	2uA	220k	1.4M	x

In Table 1, R2 and R1 are the set resistor values that we will be switching for different currents using our relay. The constant current is measured also. The minimum and maximum values of the load resistor (which is labeled Min and Max in the table) are determined based on when the current measurement changes. For example, when we reached 1k in our first set, the constant current source started to decrease and therefore we know we need to change the set resistors for a new constant current. The highest drop in the table represents the highest voltage drop of the maximum load resistance for that set. These voltage drop values will help us determine when to switch the sets when coding the Arduino.

Before testing the whole ohmmeter on the breadboard, we tested all the components and code so we knew everything worked. The first test we did was test the relays and the relay driver. We started with one relay to see how a relay works with the relay driver.

For the relays, we used the reed relay HE721A0510 reed relay recommended by our classmate Aleksandra. The original relay we were gonna use was the Elec-trol relay that our professor recommended but because the original Elec-trol company was bought by Little Fuse, we used the ones similar to the Elec-trol ones that are made by Little Fuse. The datasheet and where we bought the relay is mentioned in reference [9] and [10]. We used reference [19] to better understand how the relay works. For the relay driver, we used the uln2803 which was recommended to us by the professor. You can see the datasheet and where we can buy the uln2803 in references [4] and [5].

To test the relay driver and the relay we set up the breadboard circuit shown in Figure 3. In Figure 4 you can also see the code we used to test the relay. This code is written the help of reference [16] but it is not a copy of the code. The reference code was used to better understand how to write to the digital inputs of the Arduino.

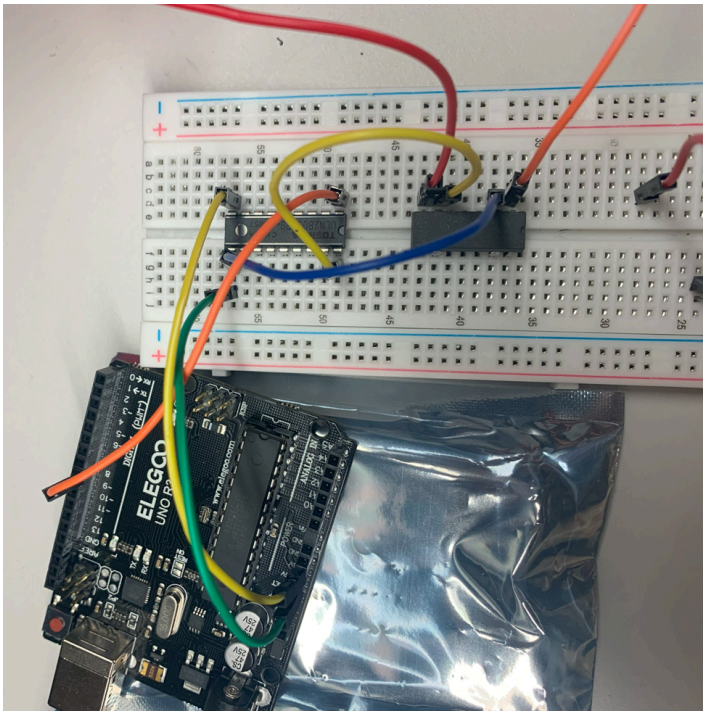

Figure 3 - Relay Driver and Relay Test using Arduino

sketch_nov15a.ino

```
1   void setup() {
2     Serial.begin(9600);
3     pinMode(8, OUTPUT);
4   }
5
6   void loop() {
7     digitalWrite(8, HIGH);
8     delay(1000);
9     digitalWrite(8, LOW);
10    delay(1000);
11    digitalWrite(8, HIGH);
12    delay(1000);
13    digitalWrite(8, LOW);
14    delay(1000);
15  }
16
```

Figure 4 - Switch Test Code

In the code, we are writing to digital pin 8 of the Arduino which is connected to the input of the uln2803. In this code, when the Arduino is writing high to the inputs of the uln2803 which in turn switches the relay to the closed position. When the Arduino is writing low to the inputs of the uln2803, it will cause the relay to be in the open position.

To test whether the relay is in the closed position, we put our multimeter on both pins 1 and 7 of the relay. When the digital signal is written high, the switch should be closed and therefore the multimeter should make a sound indicating that both pins are connected.

The next test we did was to test the LCD and the code for the LCD. For the LCD code, we used the 12C Liquid Crystal library in reference [12]. In Figure 5 you can see the connection between the Arduino and LCD and the display of the LCD after running the Arduino code. In Figure 6, you can see the Arduino code used to test the LCD on its own. The code used for the LCD test was written based on the reference [13].

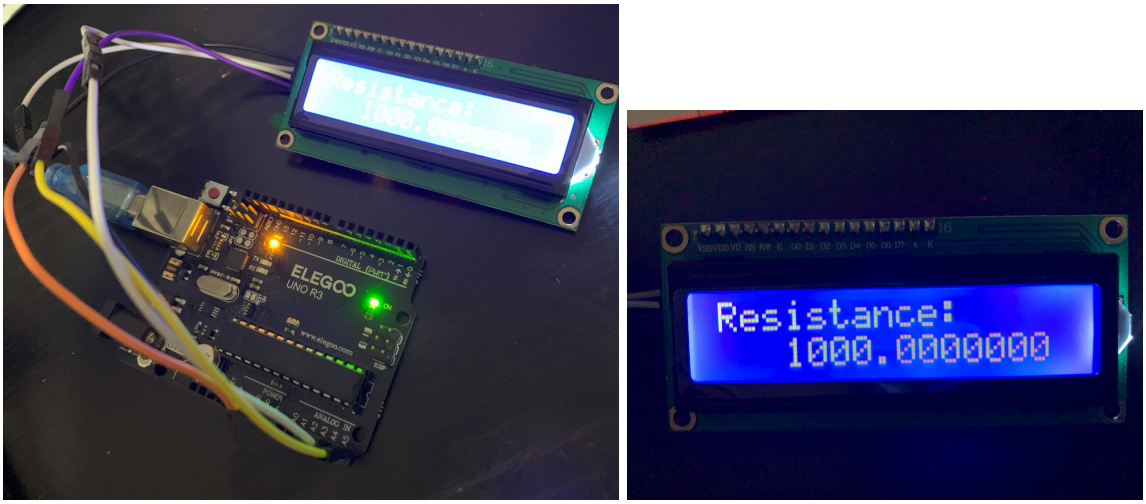

Figure 5 - LCD Test

```
Arduino Uno
sketch_nov17a.ino
4   LiquidCrystal_I2C lcd(0x27, 20, 4);
5
6   void setup() {
7     lcd.init();
8     lcd.backlight();
9     Serial.begin(9600);
10
11  }
12
13  void loop() {
14    float Voltage = 1000;
15    lcd.clear();
16    lcd.setCursor(0,0);
17    lcd.print("Resistance:  ");
18    lcd.setCursor(3,1);
19    lcd.print(Voltage,7);
20    delay(1000);
21  }
22
```

Figure 6 - LCD Test Code

This code simply prints out the resistance of 1000. It is not measuring anything so the printed resistance is just for testing.

After testing the LCD, we decided to test both the LCD and the ADS1115 together. We got the ADS1115 from can be seen in reference [7]. For this test, we used reference [13] for both the connection of the LCD and the ADS1115. We also used reference [13] for the overall testing code so we could learn how the ads1115 and I2C LCD library work. In Figure 7, you can see the connection we had for the ADS1115 and

the LCD. For this test, we used a voltage divider to see if we could measure and print out the voltage on the LCD. Figure 8 is the code we used to test.

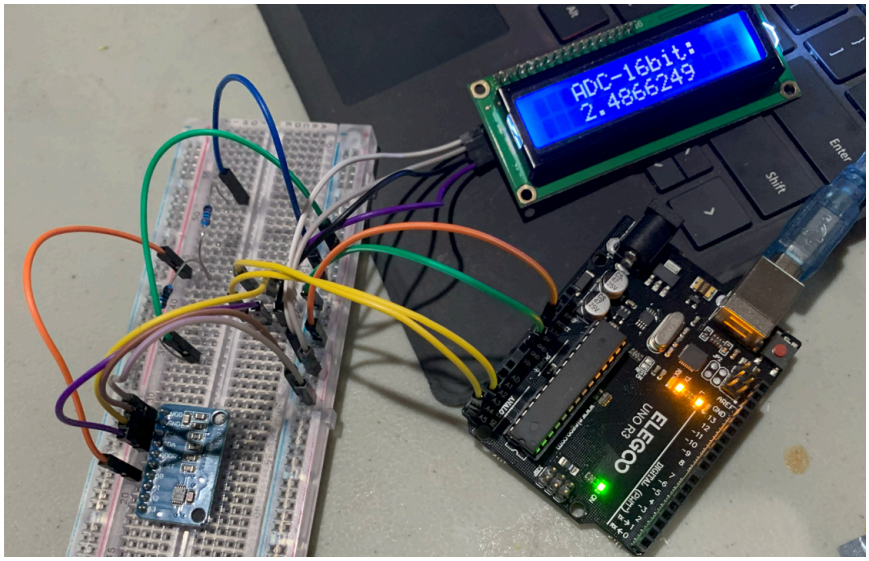

Figure 7 - ADS1115 and I2C LCD Test on Voltage Divider Circuit


```ino
#include <Wire.h>
#include <LiquidCrystal_I2C.h>
// Set the LCD address to 0x27 or 0x3f for a 16 chars and 2 line display
LiquidCrystal_I2C lcd(0x27, 20, 4);
#include <Adafruit_ADS1015.h>
Adafruit_ADS1115 ads(0x48);
float Voltage = 0.0;


void setup(void)
{
  lcd.init();
  lcd.backlight();
  Serial.begin(9600);
  ads.begin();
}

void loop(void)
{
int16_t adc0; // 16 bits ADC read of input A0
adc0 = ads.readADC_SingleEnded(0);
Voltage = (adc0 * 0.1875)/1000;

Serial.print("AIN0: ");
Serial.print(adc0);
Serial.print("\tVoltage: ");
Serial.println(Voltage, 7);
Serial.println();
lcd.clear();
lcd.setCursor(0,0);
lcd.print("   ADC-16bit:   ");
lcd.setCursor(3,1);
```

Figure 8 - ADS1115 and I2C LCD Test on Voltage Divider Circuit Code

In the code, we are reading the voltage taken by the ADS1115 and simply printing it out on the LCD. For this test, we used two 1kΩ resistors in series (voltage divider). We measured the voltage across the second resistor and got a voltage of 2.49V as shown in Figure 7. If we did the calculations using a voltage divider formula as shown below, we should get 2.5V which is similar to the output voltage recorded by the LCD.

$$Vout = \frac{R2}{R1+R2}$$

This test shows us that we can successfully get the voltage using the ADS1115 and display it on the LCD.

After knowing that our display, ADC, relays, relay driver, and code work, we went on creating the schematic of our ohmmeter design. We know that this will be a crucial part of our project so we spent a lot of time in this part. For the placement of the relays, we got a little help from our classmate Aleksandra as she is also using the LM334Z and relays as us. Professor Berger also helped us review and give us feedback on the design, which was helpful. In Figure 9, you can see the full schematic of our project all done in Kicad.

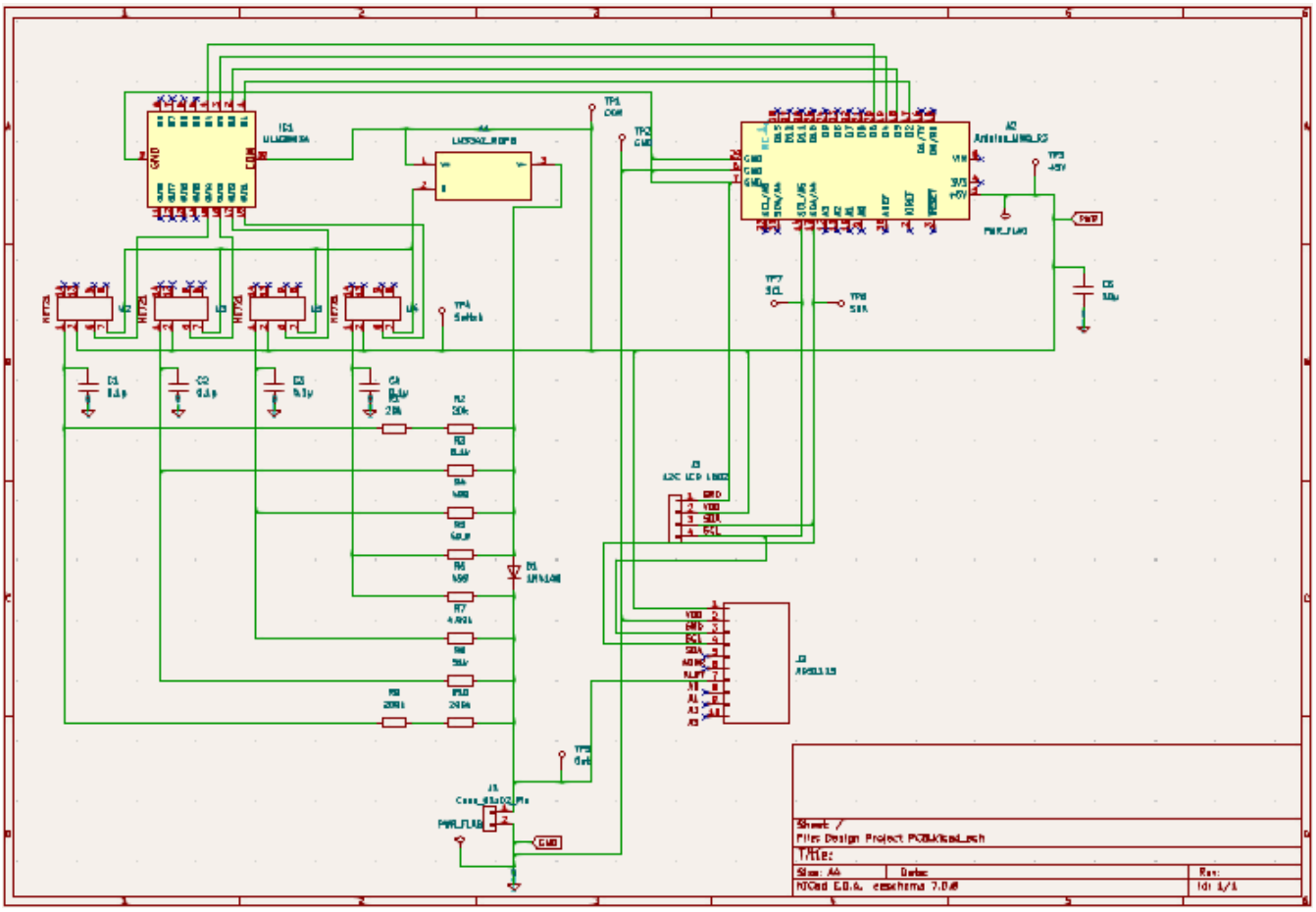

Figure 9 - Schematic for Ohmmeter Created in Kicad

Before moving on to create the PCB layout, we built and tested the whole circuit on a breadboard. In Figure 10, you can see the circuit built on the breadboard and in Figure 11 you can see the final code we used to test the circuit; Note that the code is very long so it takes up a few pages. This code will be our final code for our entire ohmmeter too.

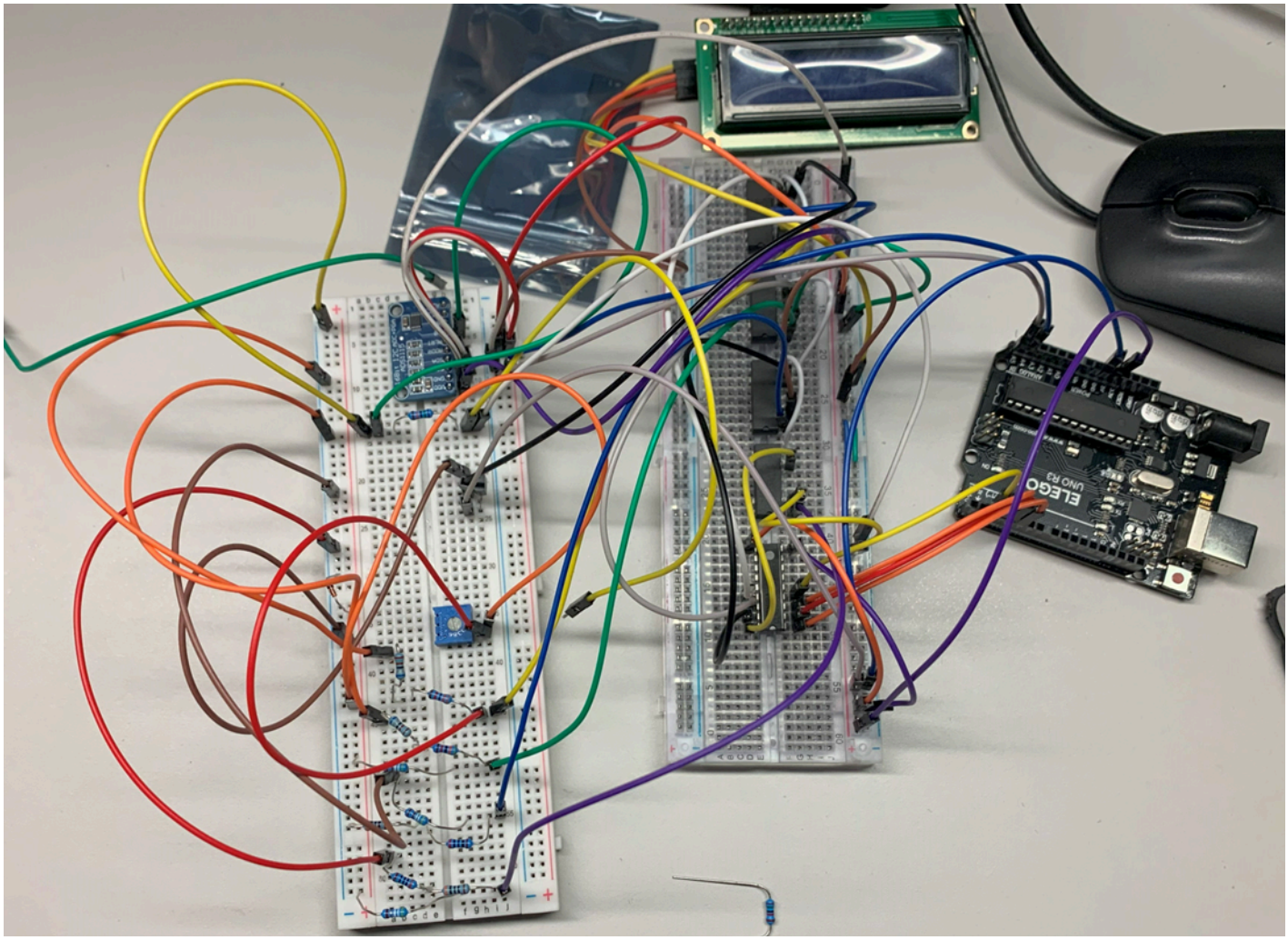

Figure 10 - Auto-ranging Ohmmeter Full Circuit

```
#include <Wire.h>
#include <LiquidCrystal_I2C.h>
// Set the LCD address to 0x27 or 0x3f for a 16 chars and 2 line display
LiquidCrystal_I2C lcd(0x27, 20, 4);
#include <Adafruit_ADS1015.h>
Adafruit_ADS1115 ads(0x48);
float Voltage = 0.0;
float Resistance = 0.0;
float Sum = 0.0;
float Average_Voltage = 0.0;
int Sample_size = 50;

void setup() {
  lcd.init();
  lcd.backlight();
  Serial.begin(9600);
  ads.begin();
  pinMode(2, OUTPUT);
  pinMode(3, OUTPUT);
  pinMode(4, OUTPUT);
  pinMode(5, OUTPUT);
}

void loop() {
  int16_t adc0; // 16 bits ADC read of input A0
  Voltage = 0;
  Sum=0;
  Average_Voltage=0;
  Resistance=0;
```

```
  digitalWrite(2, HIGH);
  digitalWrite(3, LOW);
  digitalWrite(4, LOW);
  digitalWrite(5, LOW);

  for(int i = 0; i < Sample_size; i++){
    adc0 = ads.readADC_SingleEnded(0);
    Voltage = (adc0 * 0.1875)/1000;
    Sum = Sum + Voltage;
  }

  Average_Voltage = Sum / Sample_size;
  Resistance = (Average_Voltage)/(0.00283);
  if(Voltage > 3.316){
    Voltage = 0;
    Sum=0;
    Average_Voltage=0;
    Resistance=0;
    digitalWrite(2, LOW);
    digitalWrite(3, HIGH);
    digitalWrite(4, LOW);
    digitalWrite(5, LOW);
    for(int i = 0; i < Sample_size; i++){
      adc0 = ads.readADC_SingleEnded(0);
      Voltage = (adc0 * 0.1875)/1000;
      Sum = Sum + Voltage;
    }
```

```
60      if(Voltage > 3.629){
61        Voltage = 0;
62        Sum=0;
63        Average_Voltage=0;
64        Resistance=0;
65        digitalWrite(2, LOW);
66        digitalWrite(3, LOW);
67        digitalWrite(4, HIGH);
68        digitalWrite(5, LOW);
69
70        for(int i = 0; i < Sample_size; i++){
71          adc0 = ads.readADC_SingleEnded(0);
72          Voltage = (adc0 * 0.1875)/1000;
73          Sum = Sum + Voltage;
74        }
75
76        Average_Voltage = Sum/Sample_size;
77        Resistance = (Average_Voltage)/(0.000015);
78
79        if(Voltage > 3.200){
80          Voltage = 0;
81          Sum=0;
82          Average_Voltage=0;
83          Resistance=0;
84          digitalWrite(2, LOW);
85          digitalWrite(3, LOW);
```

```
      digitalWrite(4, LOW);
      digitalWrite(5, HIGH);

      for(int i = 0 ; i < Sample_size; i++){
        adc0 = ads.readADC_SingleEnded(0);
        Voltage = (adc0 * 0.1875)/1000;
        Sum = Sum + Voltage;
      }
      Average_Voltage = Sum/Sample_size;
      Resistance = (Average_Voltage)/(0.00000208);
    }
  }
  }
  if(Resistance > 2000000){
    lcd.clear();
    lcd.print("No load");
    delay(1000);
  } else {
    Serial.print("AIN0: ");
    Serial.print(adc0);
    Serial.print("\tVoltage: ");
    Serial.println(Voltage, 7);
    Serial.println();
    lcd.clear();
    lcd.setCursor(0,0);
    lcd.print("Resistance:   ");
    lcd.setCursor(3,1);
    lcd.print(Resistance,4);
    delay(500);
  }
```

Figure 11 - Final Full-length Ohmmeter Code

Using all the test codes as a reference, we were able to create a code for our ohmmeter. Due to the length of the code, I will not be able to describe everything in detail but I will try to summarize the code. The code uses the LiquidCrystal_I2C.h and adafruit_ads1015.h libraries for the 12C LCD and the ADS1115, both taken from reference [13]. To know which set of resistors we need to measure the load resistance, we used nested if statements to see if the output voltages reached the max voltage drop measured in Table 1. If it has then we switch to the next set until we find the set we want. We used the for loops in each if statement to get the sample voltage (average voltage) for each set resistance. If there was no load the LCD would print out “No Load” and the resistance otherwise. We know there is no load when the resistance is calculated to be above 2M, which is way above the threshold we can measure.

After testing our whole ohmmeter on a breadboard, we moved on the creating the PCB design. In Figure 12, you can see the full PCB Layout. It is important to note that we did not end up using the 0.1µF capacitors but it was still implemented in the PCB Design. Below Figure 12, you can see all the references on where and how we got our symbols and footprints for the whole design process.

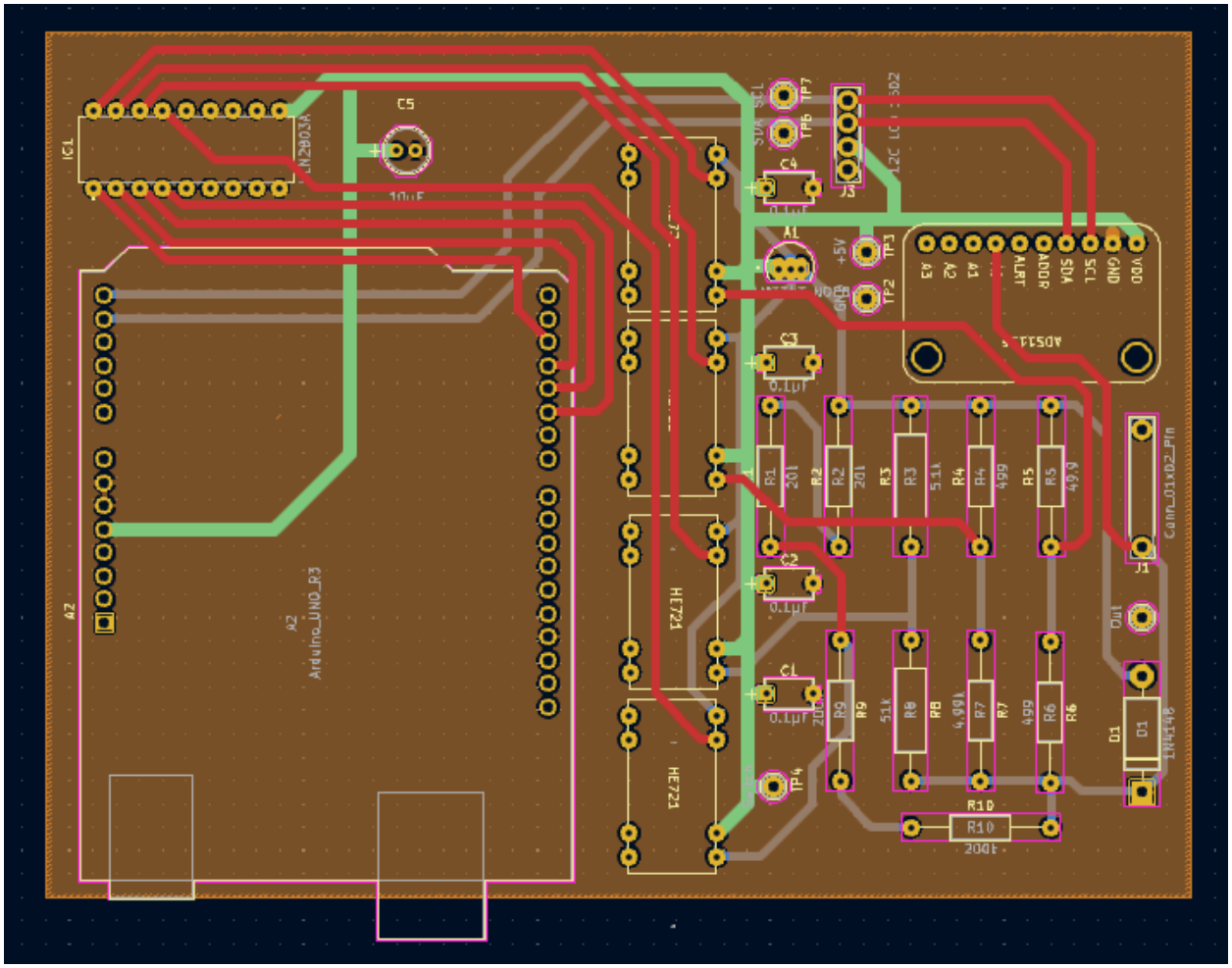

Figure 12 - PCB Layout of Auto-ranging Ohmmeter in Kicad

Note on how we got our symbols and footprints for PCB and schematic design:

For the uln2803 symbol and footprint, we used the one from reference [6]. For the LM334Z symbol and footprint, we used the one from reference [2]. For the Arduino symbol and footprint, we used reference [18]. To help create the ADS1115 symbol and footprint we used reference [8]. For the HE721A0510 reed relays, we custom-made the footprint and the symbol using the dimensions in the datasheet; we used reference [17] to help us determine the hole and pad size. For the LCD, we used a 2.54mm pin header. For the 10µF Aluminum Electrolytic Capacitor Radial footprint we used

reference [21] and for the 0.1µF Ceramic Capacitor footprint we used reference [20]. The rest of the footprints and symbols are provided in Kicad.

The finished PCB is shown in Figure 12. The soldering process of the relay was straightforward. The PCB was also tested right after soldering to make sure everything worked.

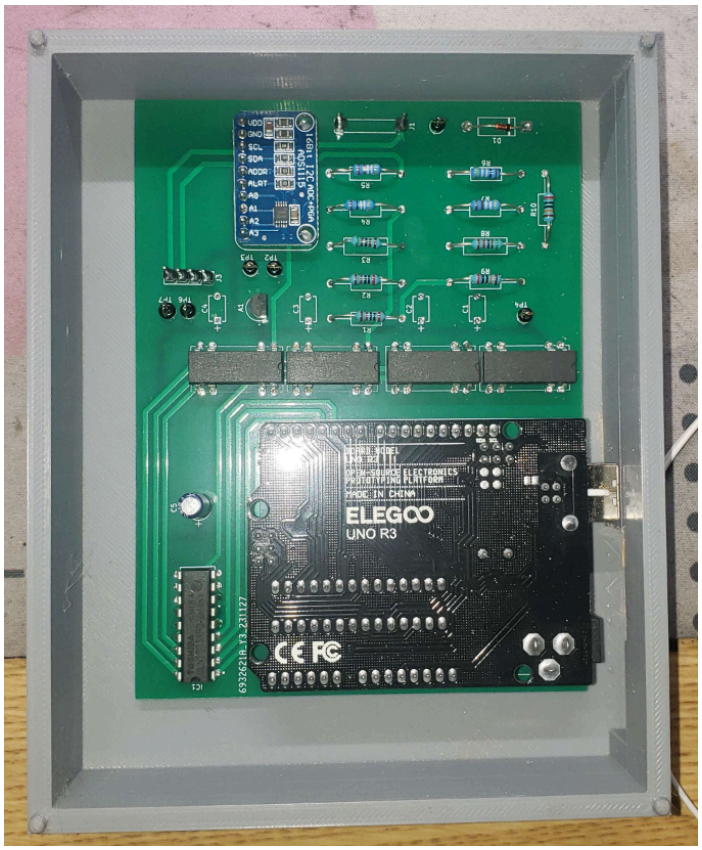

Figure 13: Ohmmeter inside of box design

When making the box, we wanted it to be modular as well as easily accessible in case it needed to be taken out for some reason. The board was measured and given a tolerance of about 10mm for there to be hand room when removing. A hole was designed on the side of the box so that the power/USB cord was easily accessible. The LCD screen was mounted to the acrylic panel with holes to easily match up with the corner pegs on the box itself.

At lower resistance from 1Ω to 200kΩ, we were able to measure the resistance within 0.1% precision. From 300kΩ to 700kΩ, we were able to measure about 1% precision. And anywhere above 700kΩ to 1M, the precision is about 2% or less. The specs say that we need 0.1% precision for 10Ω to 100kΩ and 1% for anything above or

below that until 1MΩ. Most of our ohmmeters should meet the specifications except in higher value resistors such as 700kΩ and up. That is when the resistance measurements become more off and the precision becomes more off as well.

Conclusion

In conclusion, we have successfully designed and implemented a functioning auto-ranging ohmmeter within the given specifications. By providing a known constant current source and using a voltage divider to read the voltage at the unknown resistor, we can apply Ohm's law to calculate the resistance of the unknown resistor. We ended up having 4 ranges that range up to about 1MΩ.

Something that we would do differently for the ohmmeter would be to try to make the ohmmeter more precise by adding more switches for different constant currents. We would also like to work on the range of the ohmmeter if reading more than 1M is a challenge. We can do that by also adding more switches and set resistors.

Overall, in this project, we learned many useful skills including designing and implementing PCBs, planning our design and implementing them, working with Arduino and relays, learning about different components like the uln2803 as a driver, and many more. It is also great that we can showcase the knowledge we acquired from this project to interviews by showcasing our ohmmeter.

References

[1] LM334Z datasheet - https://datasheet.octopart.com/LM334Z-National-Semiconductor-datasheet-7272458.pdf

[2] LM334Z footprint - LM334Z/NOPB footprint & symbol by Texas Instruments | SnapMagic Search (snapeda.com)

[3] The LM334Z LTSpice Model was given by Aleksandra on canvas

[4] ULN2803 - ULN2803: 8 Channel Darlington Driver (Solenoid/Unipolar Stepper) [ULN2803A] : ID 970 : $3.95 : Adafruit Industries, Unique & fun DIY electronics and kits ULN2803,04APG/AFWG (adafruit.com)

[5] ULN2803 datasheet - https://cdn-shop.adafruit.com/datasheets/ULN2803A.pdf

[6] ULN2803 footprint - ULN2803A footprint & symbol by STMicroelectronics | SnapMagic Search (snapeda.com)

[7] ADS1115 - https://www.amazon.com/HiLetgo-Converter-Programmable-Amplifier-Development/dp/B01DLHKMO2/ref=asc_df_B01DLHKMO2/?tag=hyprod-20&linkCode=df0&hvadid=642170445257&hvpos=&hvnetw=g&hvrand=8121495614103402442&hvpone=&hvptwo=&hvqmt=&hvdev=c&hvdvcmdl=&hvlocint=&hvlocphy=9033336&hvtargid=pla-821733260382&psc=1&mcid=c2b36f9d719a3e2a8459234f60016e6c

[8] ADS1115 footprint - https://www.youtube.com/watch?v=zYXPuMA9KzQ

[9] HE721A0510 reed relay- https://www.digikey.com/en/products/detail/littelfuse-inc/HE721A0510/133186
Note: We were referred to these relays by Aleksandra since it is similar to the Elec-trol ones given by the professor.

[10] HE721A0510 reed relay datasheet - media (littelfuse.com)
Note: for the relay footprint, we custom-made it using the datasheet provided above.

[11] I2C LCD - https://www.amazon.com/GeeekPi-Character-Backlight-Raspberry-Electrical/dp/B07S7PJYM6

[12] I2C LCD Module - http://www.handsontec.com/dataspecs/module/I2C_1602_LCD.pdf

[13] ADS1115 and LCD code implementation - https://electronoobs.com/eng_arduino_tut83.php

[14] 1N4148 diode datasheet - https://www.onsemi.com/download/data-sheet/pdf/1n914-d.pdf

[15] Power supply reference - https://www.instructables.com/Powering-Arduino-with-a-Battery/

[16] Switch code reference - https://roboindia.com/tutorials/arduino-stepper-motor/

[17] Custom holes and pad size for custom footprint calculation - https://electronics.stackexchange.com/questions/197747/through-hole-footprints-pad-size-from-technical-drawings

[18] Arduino library installation in Kicad - https://github.com/Alarm-Siren/arduino-kicad-library

[19] About reed relays - https://amperite.com/blog/normally-closed-relays/

[20] 0.1µF Ceramic Capacitor footprint - https://www.snapeda.com/parts/C322C104K5R5TA7301/KEMET/view-part/?ref=dk&t=C322C104K5R5TA7301&con_ref=None

[21] 10µF Aluminum Electrolytic Capacitor Radial footprint - https://www.snapeda.com/parts/UVR2W100MHD1TO/Nichicon/view-part/?ref=search&t=UVR2W100MHD1TO