Auto Ranging Ohmmeter

Documentation

Design Project

Max Kelton

Danny Kha

BEE 425

Fall 2022

Introduction **3**

Hardware Design **3**

Enclosure **7**

Software Design **9**

Discussion of Parts Selected (budget) **15**

Methodology of Testing **17**

Results **19**

Reflections **20**

Acknowledgements **21**

Introduction

The objective of this design project is to create a digitally controlled auto ranging ohmmeter. With this design project we were able to utilize the knowledge we learned from previous and current courses to build a functional ohmmeter.

Hardware Design

Schematic

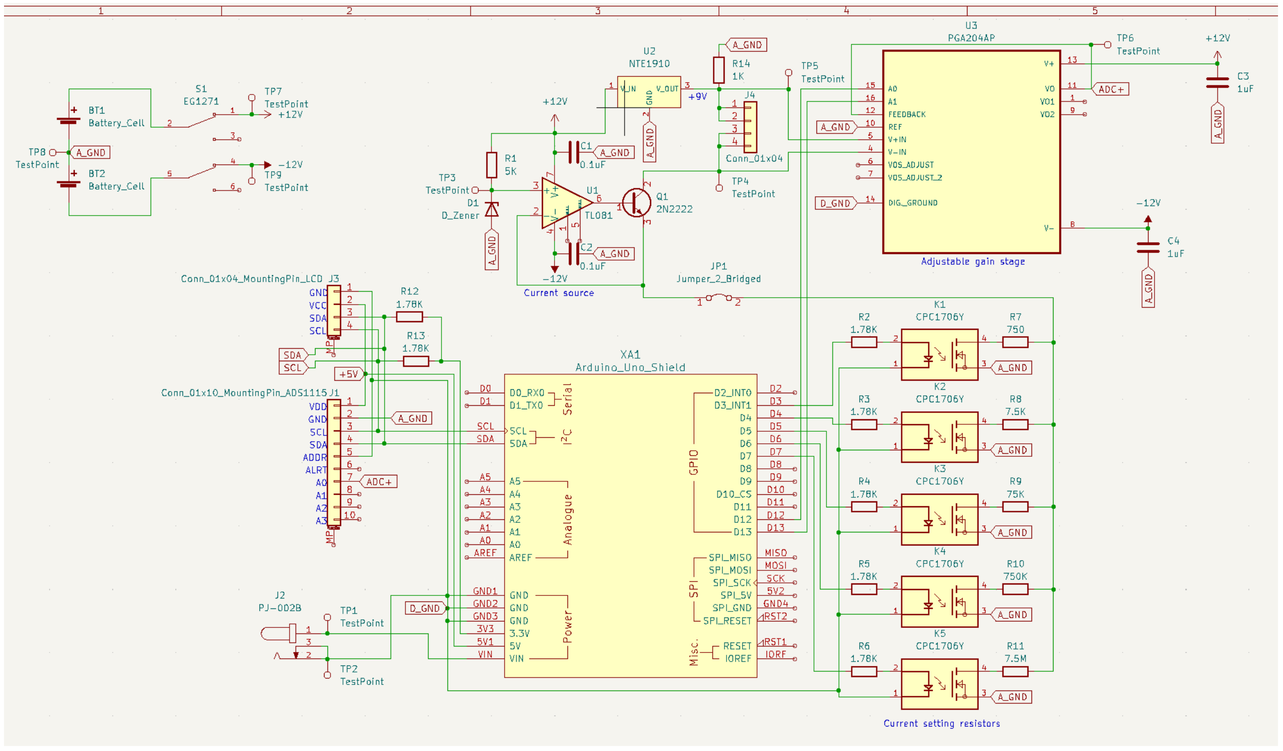

Figure 1 - Ohmmeter Schematic Layout

Major Components in the Ohmmeter Design

1) **Arduino (UNO REV3)**: An Arduino Uno rev3 board is utilized as the main control unit for the entire ohmmeter. The Arduino is tasked with displaying information to the LCD, reading information from the ADC, calculating values, and controlling the selection of relay sets. We chose an Arduino due to the fact that we both have prior experience with them.
2) **Current Source (TL081, 2N2222, CPC1706Y SSR)**: Using a constant current source design consisting of an TL081 Op-Amp and a 2N2222 transistor. Utilizing a set of relays as switches, the outputting current of the circuit is able to change depending on the selected resistor with a known voltage.
   We chose to do a constant current source Ohmmeter design due to the fact that we found it difficult to find a precision voltage source.

3) **ADC (ADS1116)**: A 16-bit analog to digital converter is used to read the voltages in single ended mode, by connecting the ADC to ground and voltage output of the PGA. The ADC communicates with the arduino through I2C.
   We chose 16-bit because we calculated that it would be enough resolution for our measurements, and the Arduino's built-in ADC wouldn't. We chose the ADS1116 specifically because they have Arduino libraries already written and widely used, and it was fairly cheap to buy multiple of them.
4) **PGA (PGA204)**: A programmable gain amplifier is used to read the differential voltage value across the unknown resistor. Also used in increasing the voltage gain reading in certain scenarios to increase the accuracy of the differential measurement.
   We wanted to use a PGA to control the gain in the lower ranges, and needed to use a differential amplifier to get the measurement voltage in reference to ground, bringing it within our ADC's voltage range.
5) **Voltage Regulator (NTE1966)**: Used to regulate the voltage from 12v to 9v for the PGA to accurately measure the difference between test points across the unknown resistor.
   During breadboard testing we realized that the input voltage for the PGA was too close to the positive supply voltage of the PGA at 12v, causing it to not properly read the inputs. We decided to add in a voltage regulator to reduce the voltage down from 12v to 9v so the PGA input would be around 9v while keeping the supply input voltages at +/- 12v.
6) **Display (1906 LCD)**: A standard LCD display connected to the arduino through I2C. Used to display information from the arduino.
   We chose this display because it has Arduino libraries and is simple to use.

The main idea of this Ohmmeter is to utilize Ohm's law,

$$\mathbf{V = I*R}$$

but in this case specifically we use

$$\mathbf{R = V/I}$$

Our design centers around a known current from a constant current source that allows us to then utilize an ADC (which converts an analog voltage signal to digital values) to then calculate resistance. Which means that in this case these are the variables that are the calculations to solve for resistance with the Ohmmeter:

Resistance = Unknown
Voltage = Measuring
Current = Known

To determine different resistor values outside of a single range, we designed and built a set of current setting resistors that are activated by DC Solid-State Relays. Through our code we are able to auto-range to different values of current to match up with the resistor being tested. After the current is set, the PGA and ADC will make the measurements and the Arduino calculates the resistance value with Ohm's law. Which is then displayed on the LCD.

PCB

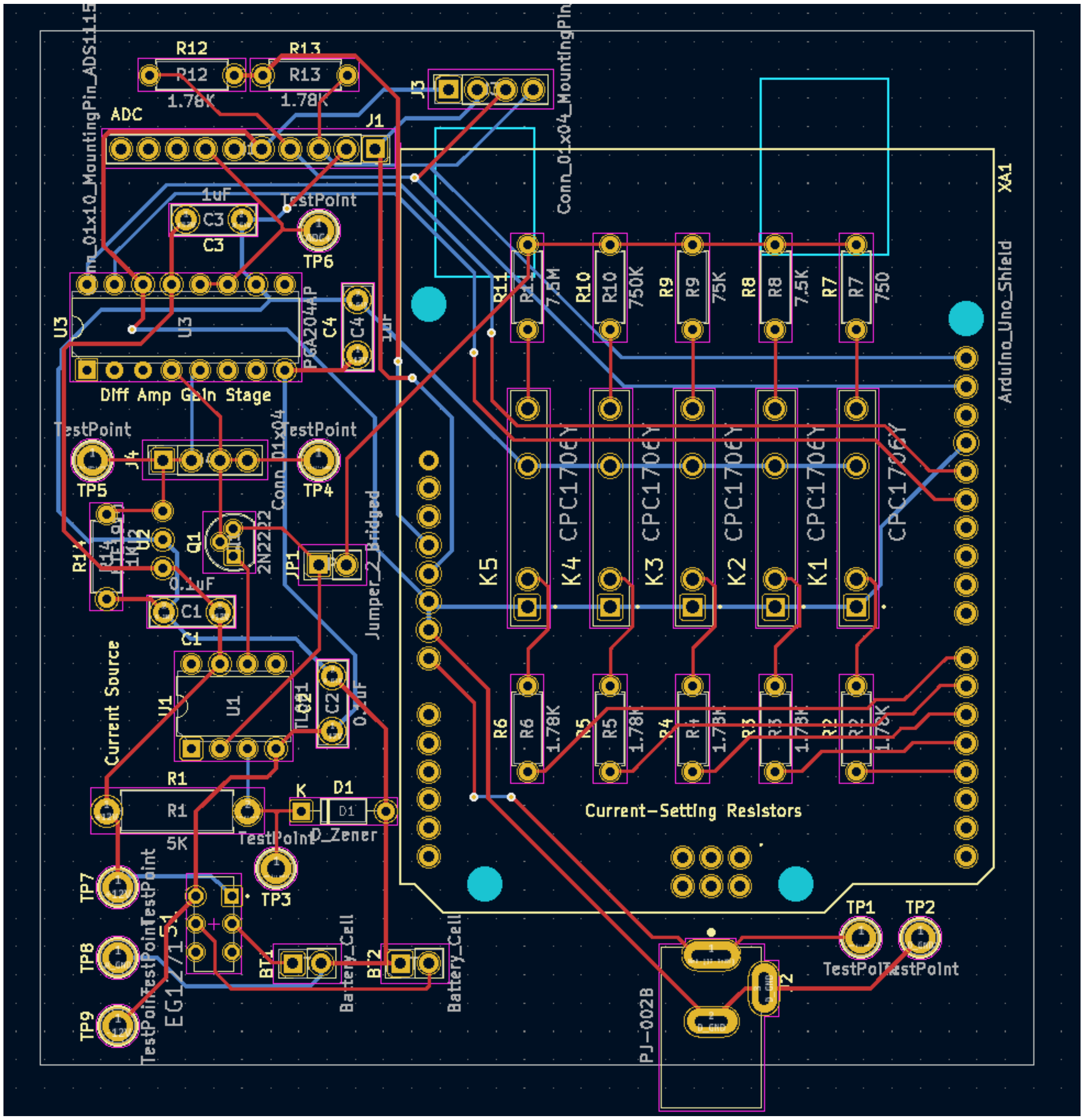

Figure 2 - Ohmmeter PCB Layout

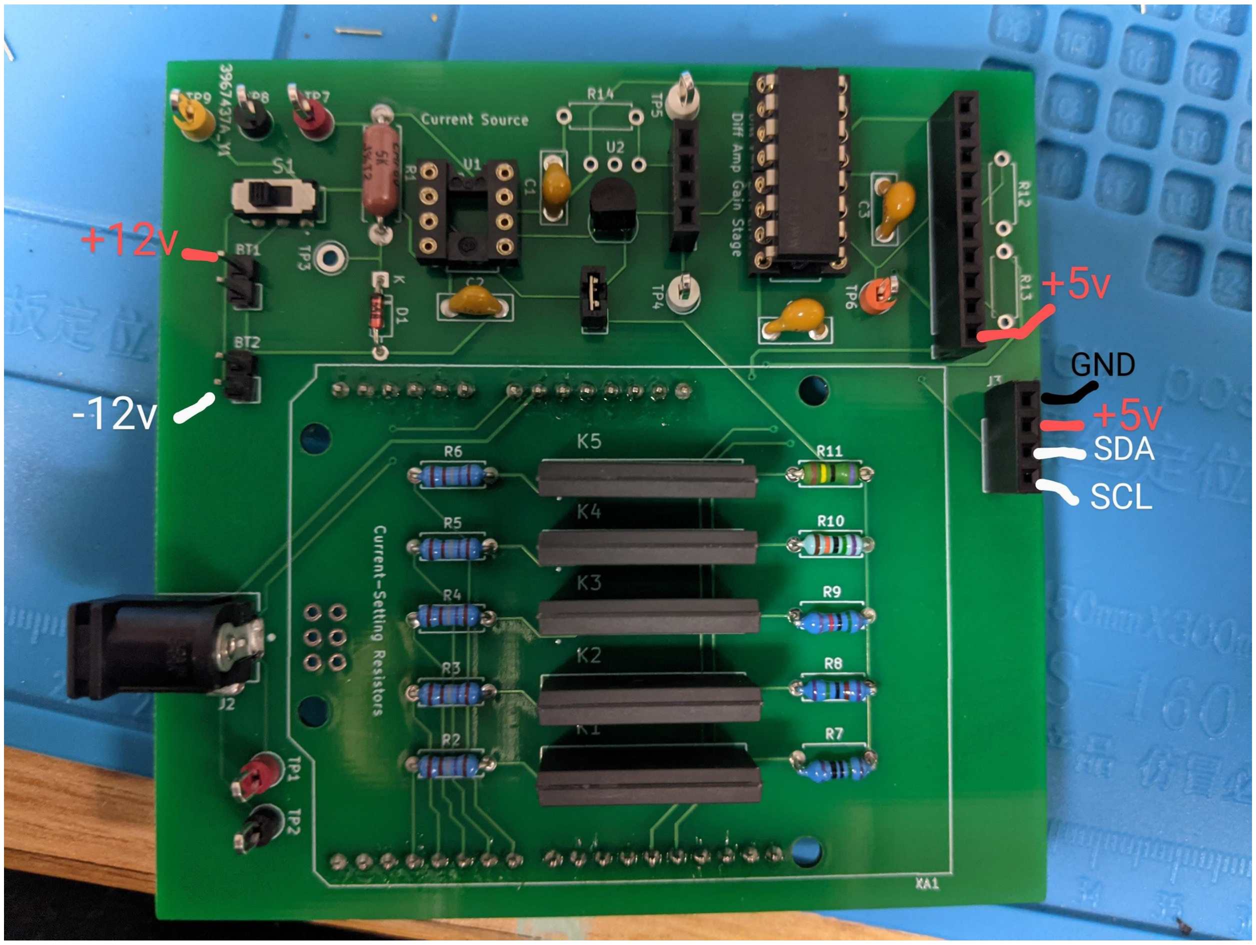

Figure 3 - Fully Assembled Ohmmeter PCB

In the PCB we added in these test points with the following:

Yellow = -12v

Black = GND

Red = +12v

White = Intermediate Values

Orange = PGA Output Voltage

Black Jumper = Current Out of Current Source

Enclosure

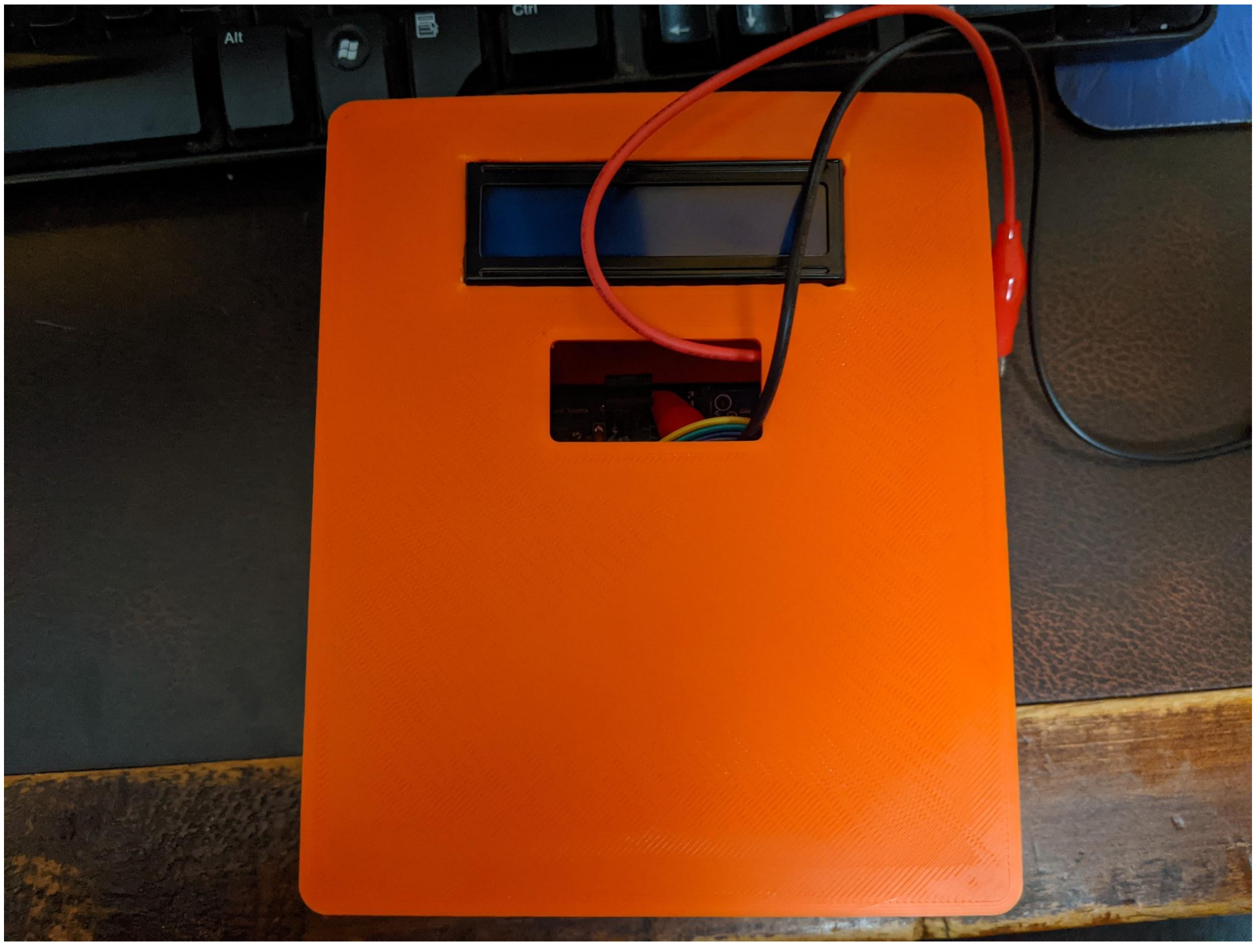

Figure 4 - Enclosure Front

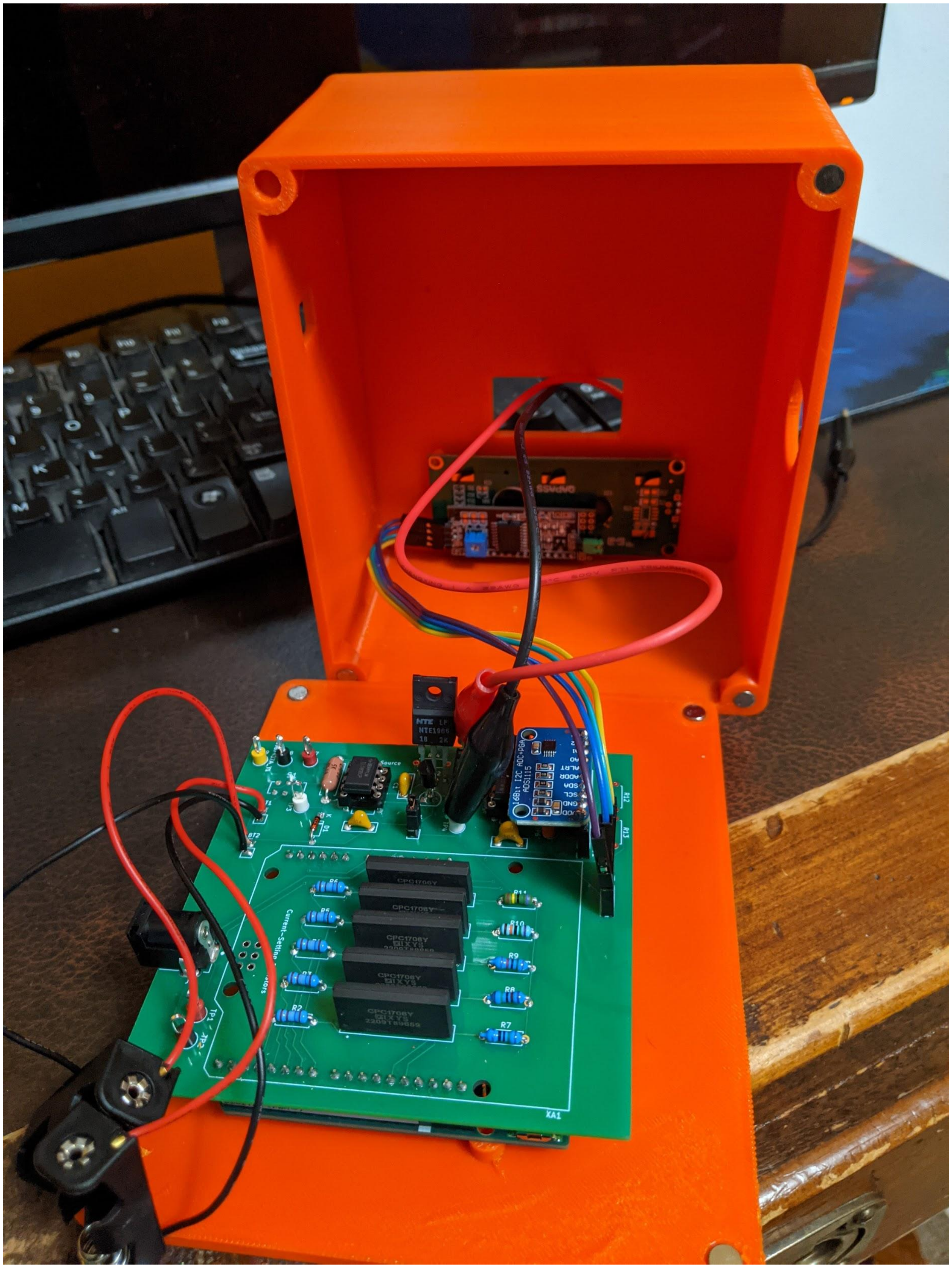
Figure 5 - Enclosure Insides

Software Design

```
/*************************************
EE 425
Ohmmeter Project Code
Professor Arnie Berger and Professor Joe Decuir
Code by Danny Kha and Max Kelton
***************/

#include <Wire.h>
#include <LiquidCrystal_I2C.h>
LiquidCrystal_I2C lcd(0x27, 20, 4);
#include <Adafruit_ADS1X15.h>
Adafruit_ADS1115 ads;

volatile double resistorRes; // unknown resistor
volatile double currCurrent; // current current that the current circuit
is set to

const double openCircuit = 1300000;
const int noBat = -1;
const int sampleRate = 100; // Setting sample rate
const int displayRate = 100; // Setting display rate
volatile double rangeLow;
volatile double rangeHigh;
volatile int lastSample; // value of the last sample
volatile int lastDisplay; // value of the last time dispalyed
volatile int currRange;

int res750 = 3; // current 1 (0.01 A) range 1
int res7p5k = 4; // current 2 (0.001 A) range 2
int res75k = 5; // current 3 (0.0001 A) range 3
int res750k = 6; // current 4 (0.00001 A) range 4
int res7p5M = 7; // current 5 (0.000001 A) range 5
int PGA0 = 12; // for setting gain of PGA
int PGA1 = 13; // for setting gain of PGA

// setup method
void setup(void)
{
  lcd.init();
  lcd.backlight();
  Serial.begin(9600);
  ads.begin(0x48);

  if (!ads.begin()) {
    lcd.setCursor(0, 0);
    lcd.print("NO ADC");
  }

  // Setting out the pins
  pinMode(res750, OUTPUT);
  pinMode(res7p5k, OUTPUT);
  pinMode(res75k, OUTPUT);
  pinMode(res750k, OUTPUT);
  pinMode(res7p5M, OUTPUT);
  pinMode(PGA0, OUTPUT);
  pinMode(PGA1, OUTPUT);

  // Gain setters
  digitalWrite(PGA0, LOW);
  digitalWrite(PGA1, LOW);
  ads.setGain(GAIN_TWO);

  currRange = 5; // initially set to the lowest current setting
  currentSwitch(currRange);
  lastSample = 0;
  lastDisplay = 0;
}

// main loop
void loop(void)
{
  int currTime = millis();
    lastSample = currTime;
    measureAndCalc();
    if (resistorRes < rangeLow) {  // if the resistor is found to be lower
then go down
      if (currRange > 1)
      {
        currRange--;
        currentSwitch(currRange);
      }
    }
    else if (resistorRes > rangeHigh) { // if the resistor is found to be
higher then go up
      if (currRange < 5) {
        currRange++;
        currentSwitch(currRange);
      }
    }
  if ((currTime - lastDisplay) >= displayRate) // display timing
  {
    lastDisplay = currTime;
    displayMeasurement();
  }
}

// display the measurements
void displayMeasurement() {
  lcd.clear();
  lcd.setCursor(0,0);
  if (resistorRes > openCircuit)
  {
    lcd.setCursor(7,0);
    lcd.print("OL");
  }
  else if (resistorRes < noBat)
  {
    lcd.setCursor(3,0);
    lcd.print("NO BATTERY");
  }
  else
   {
    lcd.print("   Ohms:   ");
    lcd.setCursor(3,1);
    lcd.print(resistorRes,7);
  }
}
```

```
// measuring the voltage and then calculating resistor result value
void measureAndCalc()
{
  int16_t readADC;
  double voltageReading;
  readADC = ads.readADC_SingleEnded(0);
  voltageReading = ads.computeVolts(readADC); // find the voltage!!
  resistorRes = voltageReading / currCurrent; // calculate the resistance
  Serial.println(voltageReading);
}

// switching the current
void currentSwitch(int rangeVal)
{
  switch(rangeVal) {
    case 1:
      current1();
      break;
    case 2:
      current2();
      break;
    case 3:
      current3();
      break;
    case 4:
      current4();
      break;
    case 5:
      current5();
      break;
  }
}

// current 1, range 1, at 750 ohm, 0.01 A
void current1()
{
  digitalWrite(res750, HIGH);
  digitalWrite(res7p5k, LOW);
  digitalWrite(res75k, LOW);
  digitalWrite(res750k, LOW);
  digitalWrite(res7p5M, LOW);
  rangeLow = 1;
  rangeHigh = 130;
  currCurrent = 0.01;
}

// current 2, range 2, at 7.5k ohm, 0.001 A
void current2()
{
  digitalWrite(res750, LOW);
  digitalWrite(res7p5k, HIGH);
  digitalWrite(res75k, LOW);
  digitalWrite(res750k, LOW);
  digitalWrite(res7p5M, LOW);
  rangeLow = 130;
  rangeHigh = 1300;
  currCurrent = 0.001;
}

// current 3, range 3, at 75k ohm, 0.0001 A
void current3()
{
  digitalWrite(res750, LOW);
  digitalWrite(res7p5k, LOW);
  digitalWrite(res75k, HIGH);
  digitalWrite(res750k, LOW);
  digitalWrite(res7p5M, LOW);
  rangeLow = 1300;
  rangeHigh = 13000;
  currCurrent = 0.0001;
}

// current 4, range 4, at 750k ohm, 0.00001 A
void current4()
{
  digitalWrite(res750, LOW);
  digitalWrite(res7p5k, LOW);
  digitalWrite(res75k, LOW);
  digitalWrite(res750k, HIGH);
  digitalWrite(res7p5M, LOW);
  rangeLow = 13000;
  rangeHigh = 130000;
  currCurrent = 0.00001;
}

// current 5, range 5, at 7.5M ohm, 0.000001 A
void current5()
{
  digitalWrite(res750, LOW);
  digitalWrite(res7p5k, LOW);
  digitalWrite(res75k, LOW);
  digitalWrite(res750k, LOW);
  digitalWrite(res7p5M, HIGH);
  rangeLow = 130000;
  rangeHigh = 1300000;
  currCurrent = 0.000001;
}
```

Figure 6 - Ohmmeter Code

Above is the code that is the heart of our auto ranging ohmmeter. The code features measuring voltage, calculating resistance according to the current setting, and displaying the correct values. It also features sampling and display rate variables that can be changed within the code.

The main loop of the code is where the function calls occur and is where we also do the autoranging. It starts with finding the current time through millis() and using this value to determine the last sample and last display times to configure a rate of sample and display. When a sample is called for, it starts with calling the measureAndCalc() function that is a function that measures the voltage by performing a single ended ADC reading. Then the function uses the ADCs built in function to convert the pure reading to a voltage reading. Lastly the function uses this voltage reading and ohm's law to calculate the resistor resulting value.

The loop then does a check if the resistor result is either less or greater than some predetermined range lows and highs that will determine if the resistor needs to switch to a different current due to the result. After this is done then the loop also does a check to determine when the last time display occurred and calls the displayMeasurement() function. The displayMeasurement() function works by using the Arduino's built in LCD library and uses print lines to print the resistor result to the LCD display.

Discussion of Parts Selected (budget)

Part	Price ($)	Description of Usage
Arduino Uno REV3	28.5	Controls all of the components.
ADS1115 16 Bit ADC	16.99	Used to read the analog voltage signal across the PGA.
CPC1706Y DC SSR	19.2	Controlling the current going through the circuit by switching resistors.
TL081BCP OP-Amp	1.12	Creating the constant current source circuit.
Zener Diode	0.22	Used to create constant voltage.
LCD Display	10.99	Displaying the outputting value measured.
1.78k Ohm Resistor	0.1	Controlling the current to activate the SSR.
5k Ohm Resistor	0.7	Used in current source circuit
750 Ohm Resistor	0.5	First setting of current at 10mA.
7.5k Ohm Resistor	0.5	Second setting of current at 1mA.
75k Ohm Resistor	0.7	Third setting of current at 100uA.
750k Ohm Resistor	0.7	Fourth setting of current at 10uA.
7.5M Ohm Resistor	0.9	Third setting of current at 1uA.

Part	**Price ($)**	**Description of Usage**
PGA204	19.75	Used to read the voltage difference across the unknown resistor and as a voltage gain stage.
2N2222A Transistor	0.24	Creating the constant current source circuit.
NTE1966 Voltage Regulator	2.14	Regulates the voltage being sent to the PGA.

Table 1 - Parts with Price and Descriptions

Methodology of Testing

We started our testing process after creating our schematic by breadboarding our entire circuit. We wrote a piece of skeleton code to test different modules of our circuit. During the breadboarding process we started by splitting up the circuit into separate sections such as the constant current source, relay array, PGA, ADC and LCD. Once we ensured that each section was working as intended, we started to combine our breadboards together and began a debugging process to determine if some sections were not interacting properly as expected. Below is a figure of our final breadboard with every section integrated together and tested.

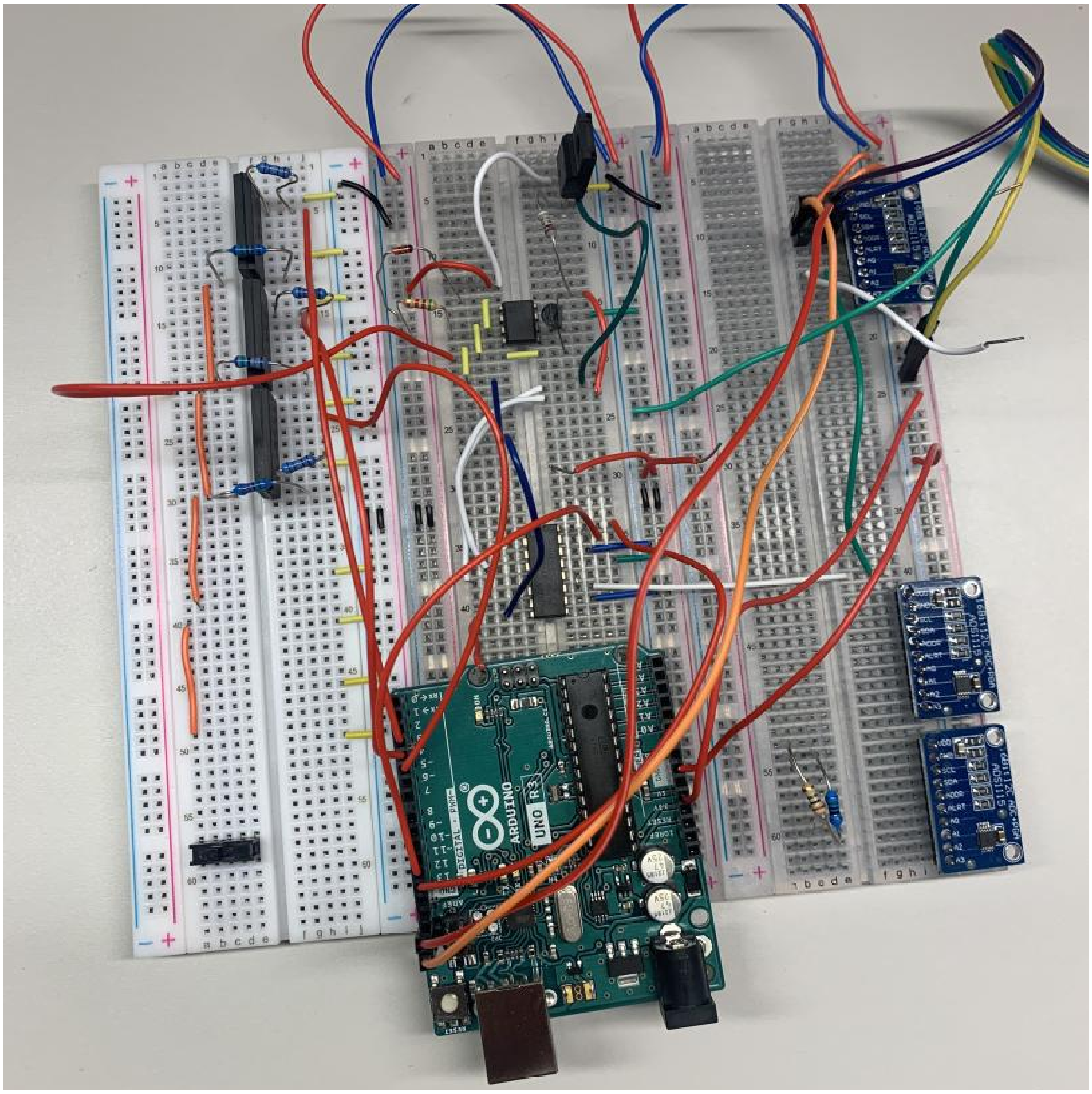

Figure 7 - Breadboard Testing of Ohmmeter

When it came to the PCB itself we decided to use pin headers and socket connections for as many components as we can in the system. This allows us to install components and replace them if necessary during testing operations. A step by step process of how we are going to test our PCB is as follows:

1) Attach the Arduino shield PCB directly to the Arduino Uno. Connect the 12v wall wart to the Arduino Uno itself and ensure that the Arduino is still powering on.
2) Connect the ADC and LCD display using pin headers on the designated location on the shield PCB. Then test that the LCD is working properly (The LCD draws power directly from the Arduino rather than the PCB power supply).
3) Install the voltage supply with both the positive and negative 12v battery system to the power pins of the PCB. Measure that a positive and negative 12v is at the locations where the OP-Amp and the PGA are utilizing it as the voltage supply.
4) After ensuring all of the prior is working, install the TL081 and PGA204 ICs into their corresponding sockets.
5) To test the current we used a skeleton code to just activate one current range at a time, and measured the current by removing jumper JP1 and attaching test leads to either end of the break.
6) Install the PGA and test that it is reading the differential voltage correctly across a known resistor, and then test that the gain is working properly, using a known resistor and current combination such that the voltage is 0.1V or less before the gain stage (This is to ensure we don’t accidentally burn out the ADC with too large of an input voltage).
7) After running through the assembly we start to adjust the software and run a series of tests with a large collection of known resistors and measuring with a separate ohmmeter and comparing that result with our ohmmeter.

Results

Resistor value (ohm)	Lab Multimeter Result (ohm)	Ohmmeter Result (ohm)	Error (%)
1 ohms	1.103 ohms	1.12 ohms	1.54%
10 ohms	10.123 ohms	10.05 ohms	0.72%
100 ohms	98.7 ohms	97.8 ohms	0.91%
1K ohms	0.972 Kohms	0.964 Kohms	0.823%
10k ohms	9.87Kohm	9.78Kohm	0.91%
100k ohms	98.11 Kohms	98.18 Kohms	0.071%
1M ohms	1.005 Mohms	1.002 Mohms	0.29%

Table 2 - Testing Results of Ohmmeter with Different Resistors

Lab Multimeter Result (ohm)	Ohmmeter Result (ohm)	Error (%)
5.12 ohms	5.19 ohms	1.37%
12.15K ohms	11.94K ohms	1.73%
0.978 M Ohms	0.988 M ohms	1.02%

Table 3 - Testing Results of Ohmmeter With Profesor Decuir

After speaking with Professor Decuir we realized that in our code we had a flaw where if we did a rolling average of the values, we would have more accurate results. But other than that we noticed that our testing results were overall fairly accurate. The testing was done on lab bench number 12 in the lab at UWB Beardslee.

Reflections

Prior to this project both of us had little to no experience in terms of designing a fully functioning PCB. Though both of us have had past experience with CAD, but not specifically PCB CAD. In the beginning of the quarter we both tackled and completed professor Berger's first homework assignment in taking a circuit that was given and creating a PCB with, which we found was challenging but doable after a few YouTube tutorials.

From this integration period of us going back and forth on design, picking parts, redesign, testing, and coding we can both agree on this project providing us with the hands-on experience that we both desired prior.

Given the opportunity to redo the project, there are a few things we would do differently:

- Use ground pours. This would have made the circuit more resistant to noise, made routing easier, and helped with heat dissipation. While heat isn't likely to be a problem, the NTE1966 linear regulator does potentially generate noticeable heat, which is made more likely by the enclosure.
- The linear regulator was a late addition to the design, and it limits our available voltage range across the resistor to a maximum of ~1.4V before it stops increasing due to the design of the current source. This means we have to rely more on our gain stages to be able to use as much of our ADC's input range as possible. If we used +/-15V for our analog supply instead of +/-12V, we would have a larger reliable range before we had to switch current values.
- Using both a battery supply and a wall wart supply is not ideal. We wanted to use a battery supply to avoid switching noise for our analog parts, but realized fairly late into our project that the Arduino would draw enough power to drain the batteries in well under an hour. A better solution would be to use a lithium-ion battery and charging circuit. That way, we could pick a battery pack that would have enough capacity to power both the Arduino and analog components.
- In terms of software we can improve on how the LCD displays the values and convert it to scientific units. For example, if our Ohmmeter LCD is currently showing 100000 Ohm reading we would convert that through software to 100.000 Kohms.
- When reading resistance values, we noticed that the outputting values on the LCD jump around due to not utilizing rolling averages for the resistor result. This would be an implementation that we would perform in the software by taking an initial reading, subtracting the new reading with the initial, and then dividing by 2. This would provide us with a rolling average and a result on the LCD that does not jump around.

Acknowledgements

Our team chose to do a schematic design review with Younes Lahrichi and Marc Arzaga. During this process we exchanged schematics and our descriptions of the functionality and what each part is performing in our design.
Through this process we learned from Younes and Marc that we should relook at the schematics of different parts that we are using such as the PGA and Op-Amp and add in filtering capacitors that the datasheet recommends since our original design schematic did not have it. Another recommendation that Younes and Marc offered was to utilize a programmable gain amplifier in the lower ranges of resistor readings and to also use it as a differential amplifier.

We were also fortunate enough to contact another student, Brian Turner, to do a PCB design review. During which we exchanged PCB and schematic CAD designs and our descriptions of the functionality and what each part is performing in our design. The following is Brian's response to our PCB.

1. Traces are very small, some at 0.25mm. I recommend thicker traces for noise immunity, durability, and current carrying capability.
2. 2. R1 has a different footprint from other resistors. This could be for a larger form factor for increased power dissipation, but double check it is not a mistake.
3. 3. No copper pours are used for ground planes or anything else. This can reduce complexity of the interconnects, offer noise reduction, and help with heat dissipation. I recommend using at least a ground plane, or possibly putting your digital ground and analog ground on different sides of the board.
4. 4. The solid state resistors are about 10.5mm tall, ensuring there is enough space for them underneath the Arduino if it is actually going to mount over them.
5. 5. IC footprints do not account for extra space required for IC sockets. Using sockets for the ICs will allow easier construction, replacement, and testing. In KiCad, I recommend using the footprints tagged "Socket" or "Hand Solder" to give you more room.
6. 6. Verify digital and analog ground are connected at some point to give a common reference point for the GPIO.
7. 7. KiCad warning exists saying J1 and J3 mounting pins aren't accounted for in the footprint. Verify the connector you intend to use is going to work without those.
8. 8. I think some custom components or libraries were left out of the materials I received, making it difficult to review pin assignments for those components. Ensure custom component symbols match datasheets.
9. 9. Many pins are left unconnected without being marked, reducing the effectiveness of KiCad's electrical rules checker. Recommend putting a no-connect symbol and any pins left unconnected and ensuring any lasting ERC warnings are taken care of.

Thanks to Brian we were able to solve all of these problems or explain to Brian why our PCB had these design decisions and thus left it alone.

Our response to Brian Turner's PCB for design review was that in certain sections of his design there were collections of components that seemed to be incredibly close to each other. Making for a tough soldering assembly, which Brian agreed and made adjustments to ensure that his PCB assembly would be flawless.

We would also like to acknowledge all of the help and support that professor Berger and Decuir provided us throughout the quarter.

Code was written with the assistance of Patrick Banez and Brian Turner as well as this website: https://simple-circuit.com/arduino-auto-ranging-ohmmeter-lcd/

The linear voltage regulators were purchased from VetCo in Bellevue.
The Arduino, LCD, and ADC were purchased from Amazon.
The rest of the parts were sourced from Digikey.
The PCB was made with love by JLCPCB.