Elior Bilow
2025-12-07
ECEN 5730 – PCB Design
CU Boulder
Board 4 Report

Project Overview:

Objective:

The objective of this board was to create a standalone device that would allow a user to measure and record the Thevenin voltage (V_{th}) and resistance (R_{th}) of various sources based on a variable current draw from the source. The device should incorporate a feedback mechanism through on-board smart LEDs and an on-board buzzer. Other features may be added for usability.

Detailed Requirements:

1. Rough Schematic / Block Diagram

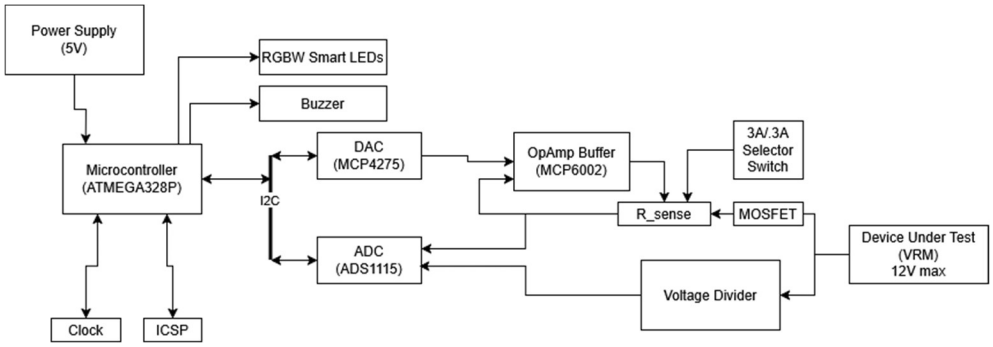

Figure 1: Block Diagram

2. Significant Parts
    a. ATMEGA328P
    b. CH340g
    c. 16MHz and 12MHz Oscillators
    d. MCP4275
    e. MCP6002
    f. ADS1115
    g. Smart LEDs
    h. Buzzer
3. Definition of "Working"
    a. The board can communicate with Arduino IDE and be bootloaded with ICSP
    b. The board can measure Thevenin resistances and voltages for supplies up to 12V
    c. The board indicates its test sequence with the use of the LEDs and buzzer

4. Test plan
    a. SPI and bootloading
        i. Bootload the 328 with the Arduino bootloader by utilizing an ICSP port
    b. Test voltages
        i. Upload sketch for testing Thevenin resistance
        ii. Test the Thevenin resistance on known sources, such as the function generator, to confirm validity

Potential risk sites:

c. Frying the board when connecting it to >5V
d. Incorrect ground plane configuration
e. Incorrect mirroring of parts and their text on the bottom layer
f. Incorrect wiring of mini-USB connector

Schematics, PCB Layout, and PCB Images:

Schematics:

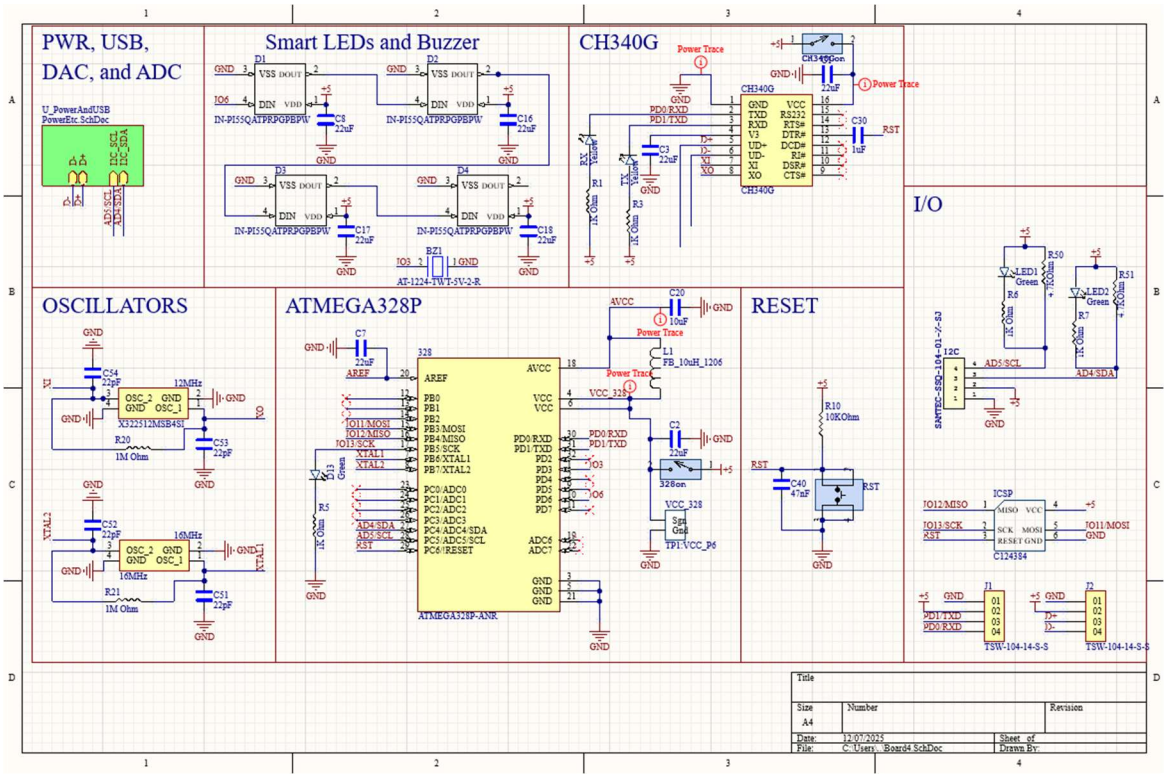

Figure 2: Schematic Sheet 1 (microcontroller)

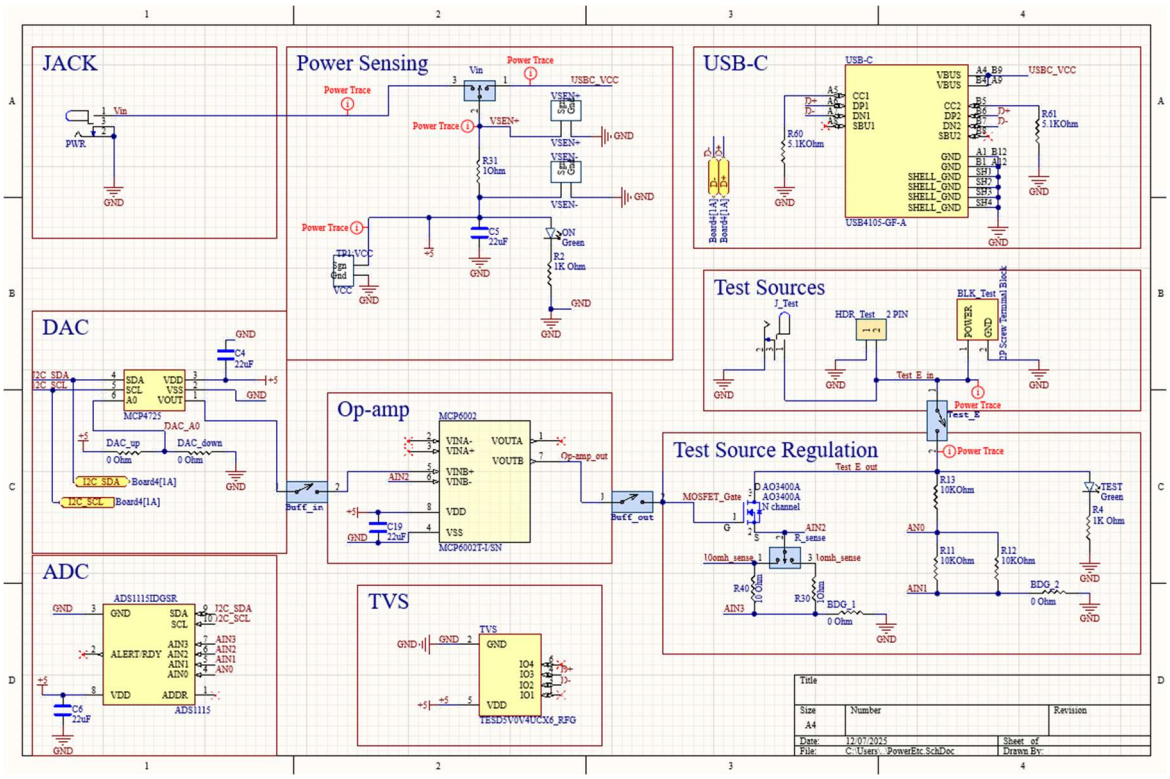

Figure 3: Schematic Sheet 2 (power and testing circuits)

Layout Top:

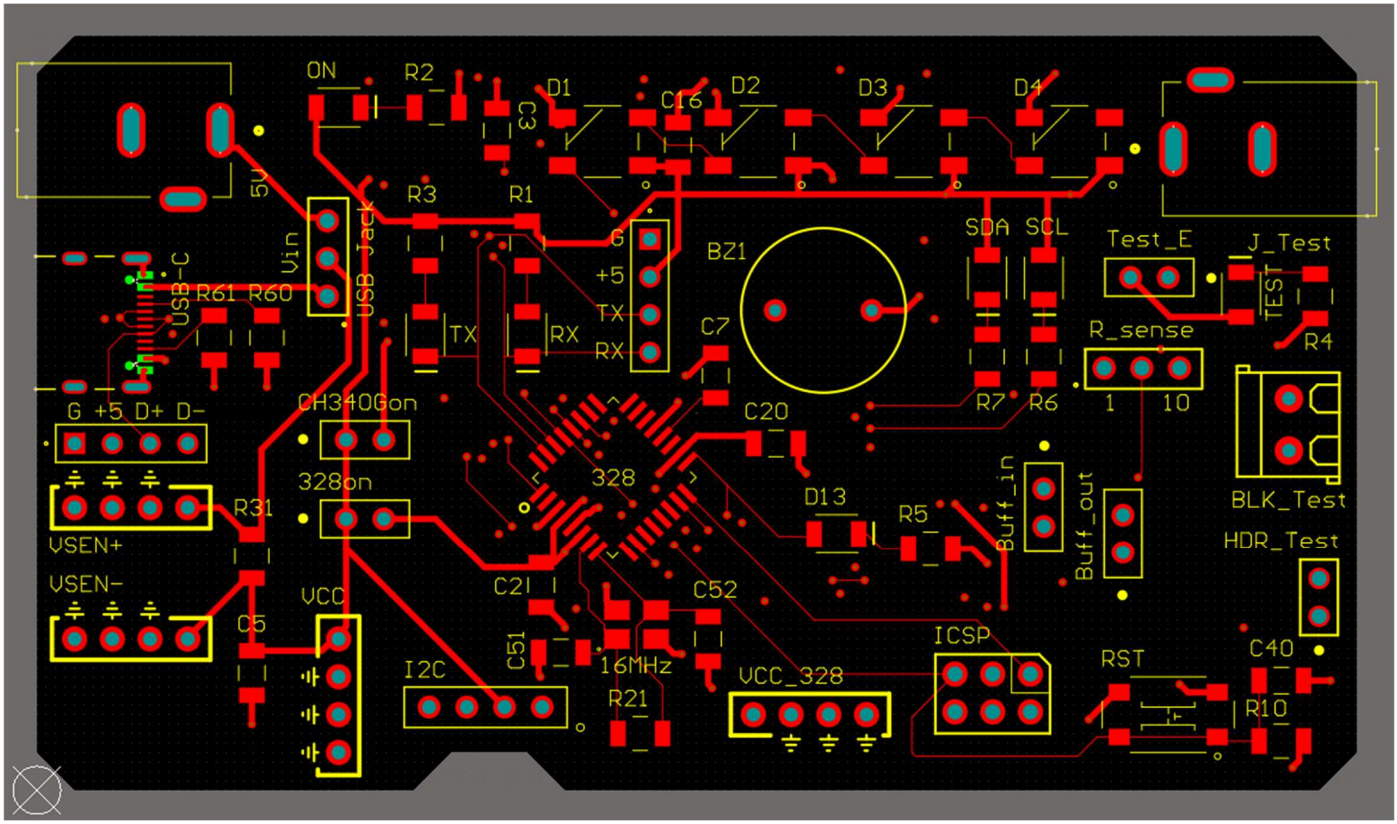

Figure 4: Layout top in Altium

Layout Bottom:

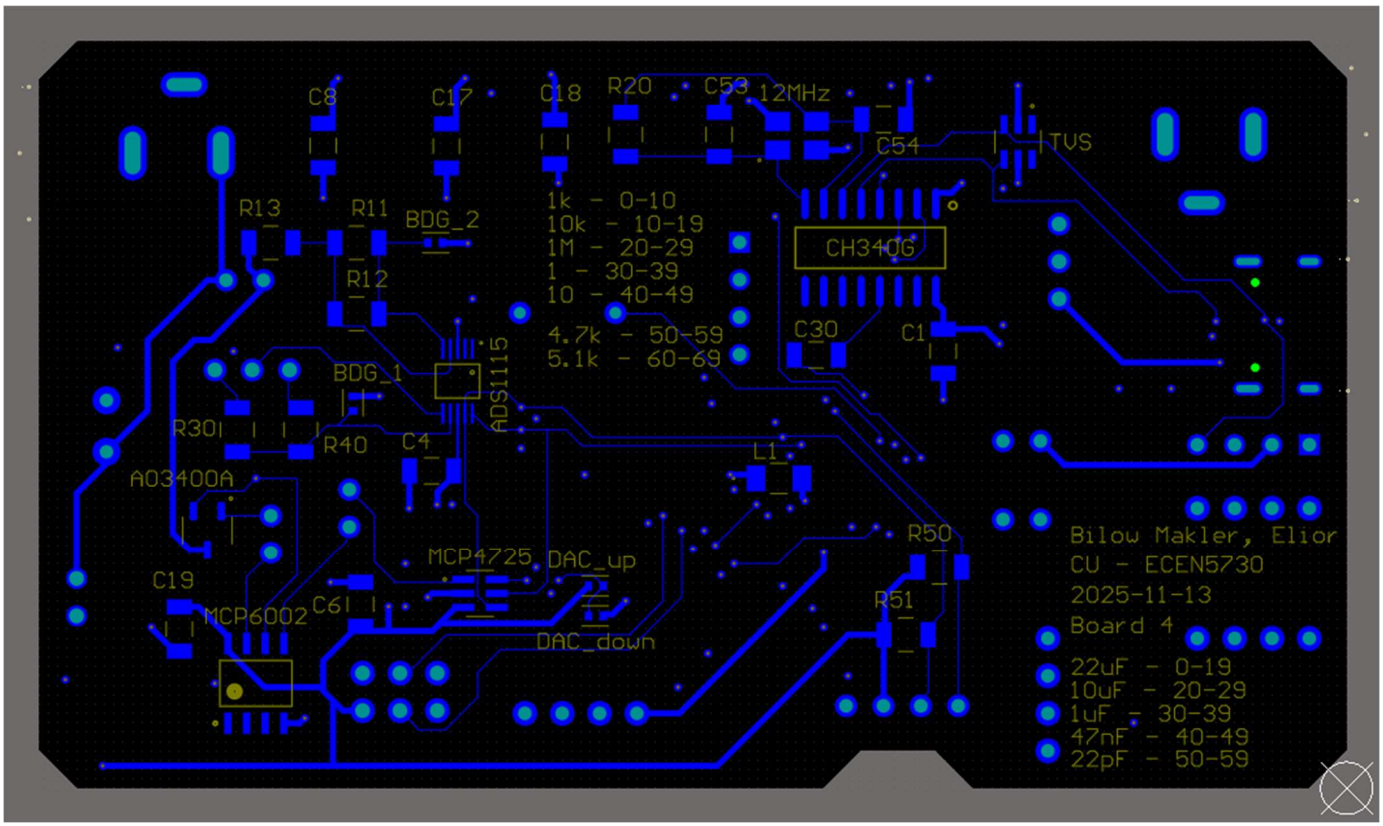

Figure 5: Layout bottom in Altium

PCB:

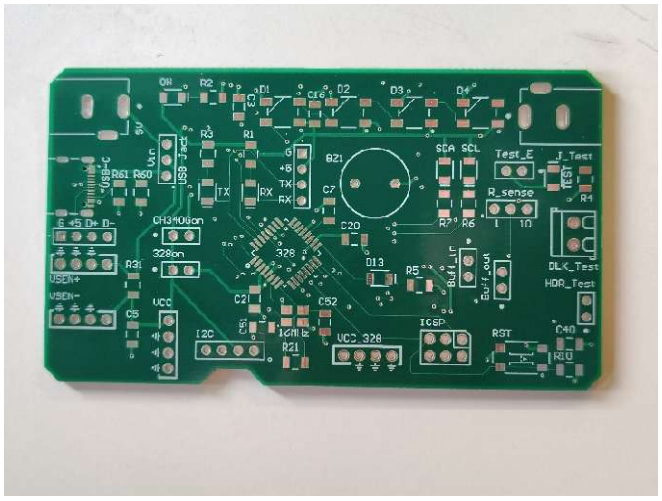

Figure 6: Unassembled PCB top

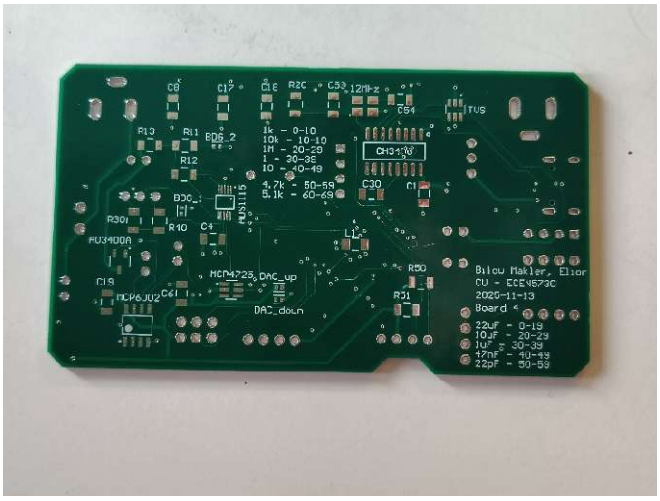

Figure 7: Unassembled PCB bottom

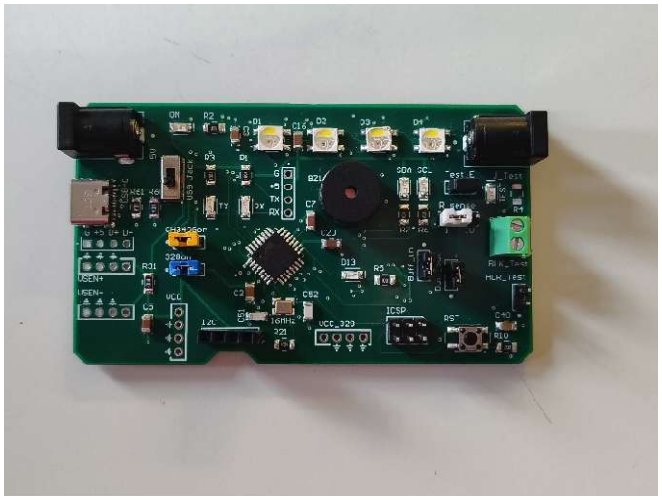

Figure 8: Assembled PCB top

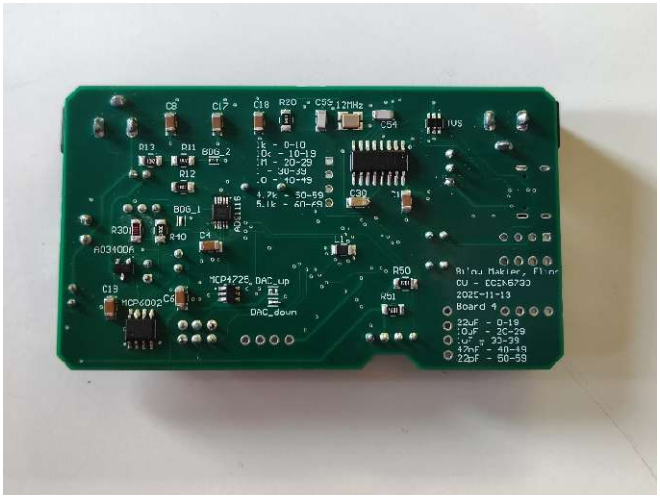

Figure 9: Assembled PCB bottom

Bring-up and Functionality Testing:

The microprocessor was flashed over ICSP flawlessly and performed the blink sketch as expected.

The LEDs were tested with the Adafruit Neopixel example sketch named “strandtest”. Initially, it appeared that the LEDs were working, but upon further inspection, they were behaving irregularly. I created a simple sketch that turned on each color independently in a known sequence and found that some LEDs would function as expected and some would not. This narrowed down the problem to my use of different models of smart LEDs. Once the LED models were consistent, the sketch worked as expected.

The buzzer was tested with a simple sketch provided in the lab manual (chapter 41, lab 24). Initially nothing happened, but I reviewed the schematic and found that the pin assignment was different. Upon updating the pin assignment, the buzzer functioned as expected.

To test the DAC and ADC, the code from Lab 21 (chapter 37) was uploaded to the board. This immediately fried the 1-ohm sense resistor, and it made the 10-ohm sense resistor very hot when I switched to that. This suggested that there was something wrong with the DAC or the buffer circuit. At this point I also noticed that I flipped the silkscreen labels for the 1Ω and 10Ω sense resistor selector I tested the output of the DAC and found that it was outputting a constant voltage of ~2.5V, meaning that it was the problem, but that the problem wasn’t that it was not connected to power. I referenced the SBB circuit and realized that I had pulled the A0 pin high instead of low. Upon switching the address of the DAC, the DAC and buffer functioned as intended.

After fixing the DAC and buffer, I was getting a reasonable but inaccurate V_{th} and R_{th} reading. I resolved this by removing an indicator LED that I had added as a notification LED to let the user know that the test source was providing power and that the polarity was correct. The reason this messed with my readings is because the code was not intended to handle a circuit with an LED.

User Experience:

The aspect that surprised me the most from my design was the tactility of the business card form factor combined with the notch in the PCB. It made the board fun to hold and made me excited to use the tool that I made.

I added an ascending tone on the buzzer for when testing started and a descending tone for when it ended. Additionally, I added tones that ascended in pitch as each test was performed.

The smart LED functionality that I added was a “progress bar” that went all red when not testing, turned green when the test started, and slowly changed from green to blue on each LED as the test progressed.

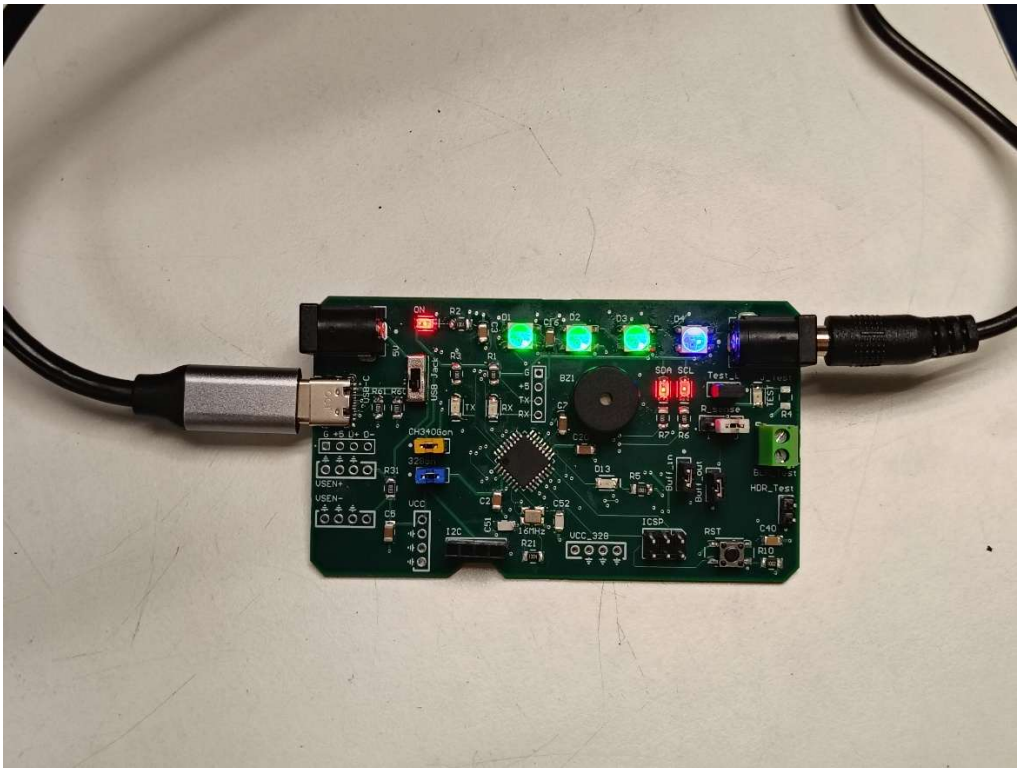

Figure 10: Progress bar using the smart LEDs

A demo of the device that showcases the smart LED and buzzer functionality can be found here: https://youtu.be/il8NrB8SMrA.

I added an I2C port to allow me to add an OLED, though I never incorporated this functionality since I broke the OLED on a different project and didn’t have a replacement on-hand. Adding OLED functionality would make this tool much more useful since it would allow the user to check the R_{th} without a serial monitor.

I am proud that my board has USB-C functionality since it is the modern standard.

I would have liked to change the way that the sense resistor is selected. Instead of needing to change the code, I would have added another input pin to the microprocessor and configured the selector as either a pull-up or pull-down. Then, I would add two MOSFETs that could select to turn on either the 1Ω or the 10Ω resistor either via the pull-up/down functionality of the selection jumper or via one or two output pins from the microprocessor.

I found that labelling the resistor and capacitor values on the board itself made assembly much easier and I wish to do that in the future.

Testing Battery Consistency:

To create a "practical" demo of how my board could be used, I tested the consistency of Kirkland-brand AA batteries. I tested 14 batteries to obtain a large enough sample size that I could see a trend. From this, I found that the Thevenin resistance drops off in a logarithmic fashion.

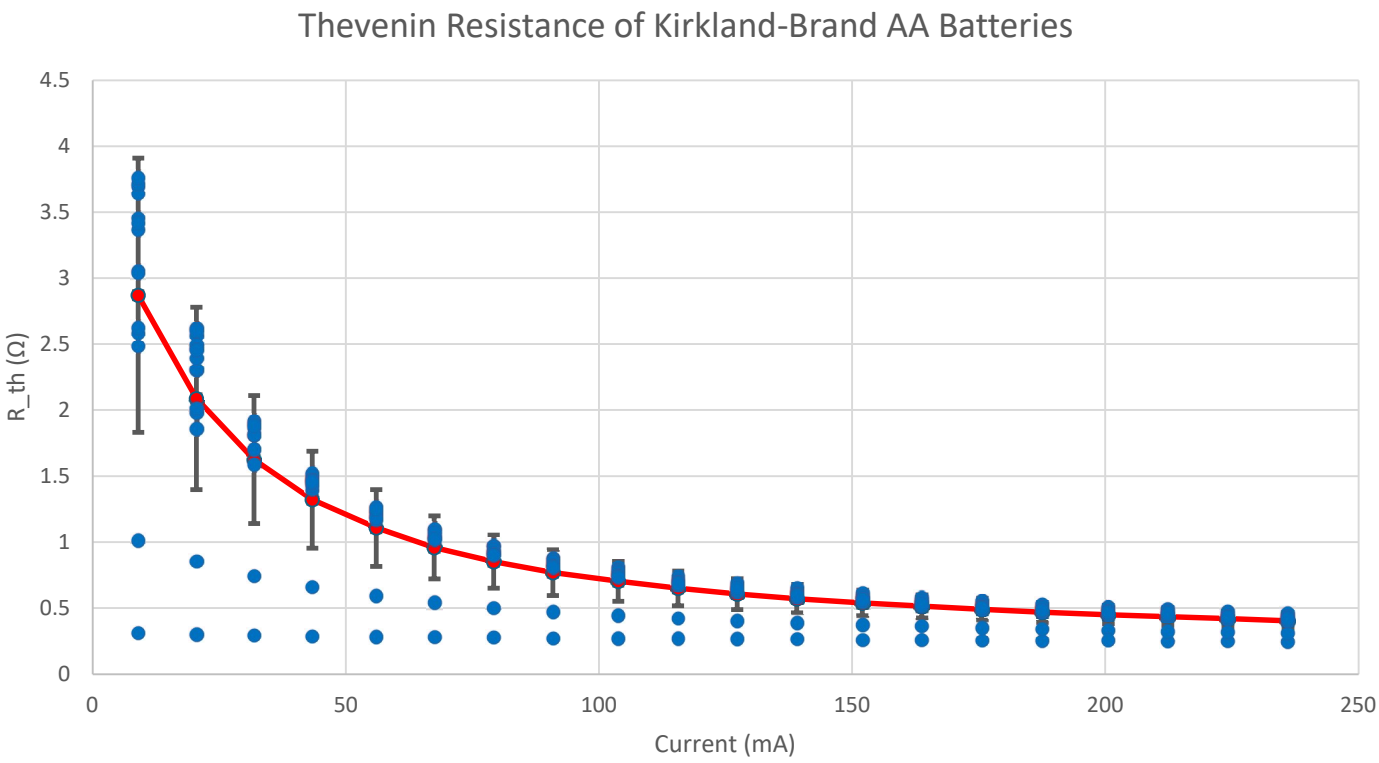

Figure 11: Thevenin resistance of Kirkland-brand AA batteries based on different load currents. The error bars show standard deviation. The red line is the average. 14 batteries were tested.

Key Learnings:

Overall, this project was a success since it meets the requirements, though there are places for improvement and I learned valuable lessons from this project.

Areas of improvement:

- The terminal block could have been labelled for which terminal is positive and which is negative.
- The test notification LED should have been controlled by a MOSFET so that it doesn't affect measurements.
- Adding smart functionality to switch between sense resistor values.
- Adding an OLED for stand-alone functionality.
- Labelling the pins for connectors (namely the I2C connector)

I also learned valuable skills in debugging since I did destroy some components after passing 2.5A through them. Some of these learnings are listed below.

- Check the SDA line and SCL line to ensure that they are both functioning and different to verify that
- Test all leads of a component to verify it is functioning as expected.
- Estimate the maximum current that my circuit can handle before powering it with that high current.

Apart from the learning from that critical error, I also learned to isolate sensitive electronic regions (such as the "test input on" LED from the test source input) and to add capabilities for "smart" functionality (like being able to switch sense resistors without changing the code.

Appendix A: Code

```
// vrm characterizer board
#include <Wire.h>
#include <Adafruit_MCP4725.h>
#include <Adafruit_ADS1X15.h>
#include <Adafruit_NeoPixel.h>

#define   INCLUDE_TONE
#define   TONE_PIN          3
#define   R_SENSE           1
#define   I_MAX             0.25
#define   INCLUDE_SMART_LEDS true
#define   SMART_LED_PIN     6
#define   SMART_LED_COUNT   4

#ifdef INCLUDE_SMART_LEDS
Adafruit_NeoPixel strip(SMART_LED_COUNT, SMART_LED_PIN, NEO_GRBW + NEO_KHZ800);
#endif //INCLUDE_SMART_LEDS

Adafruit_ADS1115 ads;
Adafruit_MCP4725 dac;
float R_sense = R_SENSE;                    //current sensor
long itime_on_msec = 100;                   //on time for taking measurements
long itime_off_msec = itime_on_msec * 10;  // time to cool off
int iCounter_off = 0;                       // counter for number of samples off
int iCounter_on = 0;                        // counter for number of samples on
float v_divider = 5000.0 / 15000.0;         // voltage divider on the VRM
float DAC_ADU_per_v = 4095.0 / 5.0;         //conversion from volts to ADU
int V_DAC_ADU;                              // the value in ADU to output on the DAC
int I_DAC;                                // the current we want to output
float I_A = 0.0;                            //the current we want to output, in amps
long itime_stop_usec;                       // this is the stop time for each loop
float ADC_V_per_ADU = 0.125 * 1e-3;         // the voltage of one bit on the gain of 1 scale
float V_VRM_on_v;                           // the value of the VRM voltage
float V_VRM_off_v;                          // the value of the VRM voltage
float I_sense_on_A;                         // the current through the sense resistor
float I_sense_off_A;                        // the current through the sense resistor
float I_max_A = I_MAX;                       // max current to set for the load resistor
int npts = 20;                              //number of points to measure
float I_step_A = I_max_A / npts;            //step current change

float V_sense_off_v;
float V_sense_on_v;
float I_load_A;  // the measured current load
float V_VRM_thevenin_v;
float V_VRM_loaded_v;
float R_thevenin; int i;

void setup() {
  Serial.begin(115200);
  Serial.println("serial initialized");
  dac.begin(0x60);           // address is either 0x60, 0x61, 0x62,0x63, 0x64 or 0x65
  dac.setVoltage(0, false);  //sets the output current to 0 initially
  Serial.print("DAC started ");
  // ads.setGain(GAIN_TWOTHIRDS);  // 2/3x gain +/- 6.144V  1 bit = 3mV      0.1875mV (default)
  ads.setGain(GAIN_ONE);  // 1x gain   +/- 4.096V  1 bit = 2mV      0.125mV
  // ads.setGain(GAIN_TWO);        // 2x gain   +/- 2.048V  1 bit = 1mV      0.0625mV
  // ads.setGain(GAIN_FOUR);       // 4x gain   +/- 1.024V  1 bit = 0.5mV    0.03125mV
  // ads.setGain(GAIN_EIGHT);      // 8x gain   +/- 0.512V  1 bit = 0.25mV   0.015625mV
  // ads.setGain(GAIN_SIXTEEN);    // 16x gain  +/- 0.256V  1 bit = 0.125mV  0.0078125mV
  ads.begin(0x48);   // note- you can put the address of the ADS111 here if needed
  ads.setDataRate(RATE_ADS1115_860SPS); // sets the ADS1115 for higher speed
  Serial.println(" ADS Started");
#ifdef INCLUDE_TONE
  pinMode(TONE_PIN, OUTPUT); //buzzer output
#endif  //INCLUDE_TONE
```

```
#ifdef INCLUDE_SMART_LEDS
  strip.begin();           // INITIALIZE NeoPixel strip object (REQUIRED)
  strip.show();            // Turn OFF all pixels ASAP
  strip.setBrightness(50); // Set BRIGHTNESS to about 1/5 (max = 255)
#endif //INCLUDE_SMART_LEDS

}

void loop() {
#ifdef INCLUDE_TONE
  // increasing pitch to indicate start
  for (int i = 1; i <= 3; i++) {
    tone(TONE_PIN, 300 * i);
    delay(100);
  }
  noTone(TONE_PIN);
#endif // INCLUDE_TONE

#ifdef INCLUDE_SMART_LEDS
  uint32_t red =    strip.Color(127, 0, 0);
  uint32_t green =  strip.Color(0, 127, 0);
  uint32_t blue =   strip.Color(0, 0, 127);
  strip.setPixelColor(0, blue);
  strip.setPixelColor(1, blue);
  strip.setPixelColor(2, blue);
  strip.setPixelColor(3, blue);
  strip.show();
  delay(10);
#endif //INCLUDE_SMART_LEDS

  Serial.print("i");
  Serial.print(",\t");
  Serial.print("V_sense_off");//, 4);
  Serial.print(",\t");
  Serial.print("V_sense_on");//, 4);
  Serial.print(",\t");
  Serial.print("I_load_A");// * 1e3, 3);
  Serial.print(",\t");
  Serial.print("V_VRM_thev");//, 4);
  Serial.print(",\t");
  Serial.print("V_VRM_loaded");//, 4);
  Serial.print(",\t");
  Serial.println("R_thevenin");//, 4);

// TESTING LOOP
  uint8_t i = 1;
  while (i <= npts) {

#ifdef INCLUDE_TONE
    // increasing pitch to percent completion start
    tone(TONE_PIN, 50 * i);
    delay(100);
    noTone(TONE_PIN);
#endif // INCLUDE_TONE


#ifdef INCLUDE_SMART_LEDS
    uint8_t points_per_cycle = npts/4;
//    Serial.print("points_per_cycle = ");
//    Serial.println(points_per_cycle);
    uint32_t switch_val = (i-1) / (points_per_cycle);
//    Serial.print("switch_val = ");
//    Serial.println(switch_val);
    uint32_t green_val = 127 * ((i - 1) % points_per_cycle) / 5;
    uint32_t blue_val = 127 * (points_per_cycle - ((i - 1) % points_per_cycle)) / 5;
//    Serial.print("green_val = ");
//    Serial.println(green_val);
//    Serial.print("blue_val = ");
//    Serial.println(blue_val);
```

```
    switch (switch_val) {
      // In first fifth of cycle, all pixels are blue
      case 0:
        strip.setPixelColor(0,strip.Color(0, green_val, blue_val,0));
        break;
      // In second fifth of cycle, pixel 0 is green
      case 1:
        strip.setPixelColor(1,strip.Color(0, green_val, blue_val,0));
        strip.setPixelColor(0, green);
        break;
      // In third fifth of cycle, pixel 1 is green
      case 2:
        strip.setPixelColor(2,strip.Color(0, green_val, blue_val,0));
        strip.setPixelColor(1, green);
        break;
      // In fourth fifth of cycle, pixel 2 is green
      case 3:
        strip.setPixelColor(3,strip.Color(0, green_val, blue_val,0));
        strip.setPixelColor(2, green);
        break;
     // In fifth fifth of cycle, pixel 3 is green
      case 4:
        strip.setPixelColor(3, green);
        break;
      default:
        //strip.setPixelColor(3, green);
        break;
    }
    strip.show();
#endif //INCLUDE_SMART_LEDS

    I_A = i * I_step_A;
    dac.setVoltage(0, false);  //sets the output current
    func_meas_off();
    func_meas_on();
    dac.setVoltage(0, false);  //sets the output current

    I_load_A = I_sense_on_A - I_sense_off_A;  //load current
    V_VRM_thevenin_v = V_VRM_off_v;
    V_VRM_loaded_v = V_VRM_on_v;
    float V_sense_off = V_sense_off_v;
    float V_sense_on = V_sense_on_v;
    R_thevenin = (V_VRM_thevenin_v - V_VRM_loaded_v) / I_load_A;

    Serial.print(i);
    Serial.print(",\t");
    Serial.print(V_sense_off, 4);
    Serial.print(",\t\t");
    Serial.print(V_sense_on, 4);
    Serial.print(",\t\t");
    Serial.print(I_load_A, 4);// * 1e3, 3);
    Serial.print(",\t\t");
    Serial.print(V_VRM_thevenin_v, 4);
    Serial.print(",\t\t");
    Serial.print(V_VRM_loaded_v, 4);
    Serial.print(",\t\t");
    Serial.println(R_thevenin, 4);

    if (V_VRM_loaded_v < 0.75 * V_VRM_thevenin_v) i = npts;  //stops the ramping

    i++;
  }

#ifdef INCLUDE_TONE
  // decreasing pitch to indicate end
  for (int i = 3; i >= 1; i--) {
    tone(TONE_PIN, 300 * i);
    delay(100);
```

```
  }
  noTone(TONE_PIN);
#endif // INCLUDE_TONE

#ifdef INCLUDE_SMART_LEDS
  strip.setPixelColor(0, red);
  strip.setPixelColor(1, red);
  strip.setPixelColor(2, red);
  strip.setPixelColor(3, red);
  strip.show();
#endif //INCLUDE_SMART_LEDS

  Serial.println("done");   delay(4000);
}


void func_meas_off() {
  dac.setVoltage(0, false);  //sets the output current
  iCounter_off = 0;                 //starting the current counter
  V_VRM_off_v = 0.0;             //initialize the VRM voltage averager
  I_sense_off_A = 0.0;       // initialize the current averager
  V_sense_off_v = 0.0;
  itime_stop_usec = micros() + itime_off_msec * 1000;  // stop time

  float V_sense_off_v_new = 0;
  while (micros() <= itime_stop_usec) {
    V_VRM_off_v = ads.readADC_Differential_0_1() * ADC_V_per_ADU / v_divider + V_VRM_off_v;
    V_sense_off_v_new = ads.readADC_Differential_2_3() * ADC_V_per_ADU;
    I_sense_off_A = V_sense_off_v_new / R_sense + I_sense_off_A;
    V_sense_off_v += V_sense_off_v_new;
    iCounter_off++;
  }

  V_VRM_off_v = V_VRM_off_v / iCounter_off;
  I_sense_off_A = I_sense_off_A / iCounter_off;
  V_sense_off_v = V_sense_off_v / iCounter_off;

  // Serial.print(iCounter_off);Serial.print(",  ");
  // Serial.print(I_sense_off_A * 1e3, 4); Serial.print(",  ");
  // Serial.println(V_VRM_off_v, 4);

}

void func_meas_on() {
  //now  turn on the current

  V_DAC_ADU = (I_A * R_sense) * DAC_ADU_per_v;
  dac.setVoltage(V_DAC_ADU, false);  //sets the output current

  iCounter_on = 0;
  V_VRM_on_v = 0.0;             //initialize the VRM voltage averager
  I_sense_on_A = 0.00;         // initialize the current averager
  V_sense_on_v = 0.0;
  itime_stop_usec = micros() + itime_on_msec * 1000;  // stop time
  float V_sense_on_v_new = 0;
  while (micros() <= itime_stop_usec) {
    V_VRM_on_v = ads.readADC_Differential_0_1() * ADC_V_per_ADU / v_divider + V_VRM_on_v;
    V_sense_on_v_new = ads.readADC_Differential_2_3() * ADC_V_per_ADU;
    I_sense_on_A = V_sense_on_v_new / R_sense + I_sense_on_A;
    V_sense_on_v += V_sense_on_v_new;
    iCounter_on++;
  }
  dac.setVoltage(0, false);  //sets the output current to zero

  V_VRM_on_v = V_VRM_on_v / iCounter_on;
  I_sense_on_A = I_sense_on_A / iCounter_on;
  V_sense_on_v = V_sense_on_v / iCounter_on;
  // Serial.print(iCounter_on);Serial.print(",  ");
  // Serial.print(I_sense_on_A * 1e3, 4);Serial.print(",  ");
```

```
  // Serial.println(V_VRM_on_v, 4);
}
```