Project Report
Resistor Sorter

Submitted By,

Vishesh Yadav(2021UEC2600)

Electronics and Communication Engineering

Acknowledgement

I extend my deepest gratitude to Prof. Dhananjay V. Gadre, Associate Professor, ECE Division at Netaji Subhas University of Technology. His constant support and guidance are what led to the completion of this project. CEDT gave me the optimal environment to learn, research, and implement my newly acquired knowledge. My thanks and appreciation also goes to my friends and seniors at the centre who have helped me out willingly throughout the course of this project. I would also like to express gratitude to my family who has always supported me in my endeavours

Contents

1. Introduction

A resistor sorter is a device that sorts the resistor based on in which range the resistor is with respect to reference resistor. It has three modes: 1% mode 0.5% mode and 0.1% mode here, the modes signify how much the unknown resistor is within range of the reference resistor. The output is shown by a single green LED. When it turns on, it signifies that the unknown resistor is in range otherwise, the LED is off.

The Arduino Nano, an ATmega 328-based board, was used in this project, specifically the ADC and the Internal Voltage Reference. Arduino Nano offers a 10-bit resolution analog-to-digital converter, which is not sufficient for the level of precision we require. This has two solutions: either using an external ADC module or Oversampling the current ADC to the desired resolution. For this project, we have used Oversampling.

Oversampling and Decimation are signal processing techniques used to achieve higher resolution. ADC oversampling is a technique employed to enhance the resolution and accuracy of ADC measurement by sampling the input signal at a higher rate than the Nyquist frequency.

In this report, we will delve into the concept of ADC oversampling, its calibration, and some advantages and disadvantages of ADC oversampling.

2. Arduino Nano

2.1 Introduction

Arduino nano is board based on the AVR microcontroller ATMega 328. The ATmega328 is an 8-bit microcontroller from the AVR family, manufactured by Microchip Technology. It is widely used in various electronic projects, prototyping, and commercial applications due to its versatility, reliability, and ease of use.

Key features of the ATmega328:

- **Architecture:** The ATmega328 is based on the Harvard architecture and operates at a clock speed of up to 20 MHz. It features a RISC (Reduced Instruction Set Computing) core with a rich set of instructions, making it efficient for handling a wide range of tasks.
- **Memory:** The microcontroller has 32KB of Flash memory for program storage, which is non-volatile and retains the program even when power is removed. It also has 2KB of SRAM for data storage and 1KB of EEPROM for non-volatile data storage.
- **GPIO Pins:** The ATmega328 has a total of 23 GPIO (General Purpose Input/Output) pins. These pins can be configured as digital inputs or outputs, and some of them also support PWM for controlling analog-like outputs.
- **Analog Inputs:** It includes 6 analog input pins, labeled as A0 to A5, which can be used to read analog sensor values through the built-in ADC (10-bit resolution).
- **Communication Interfaces:** The ATmega328 supports popular communication protocols like UART , SPI , and I2C , making it easy to interface with other devices and components..

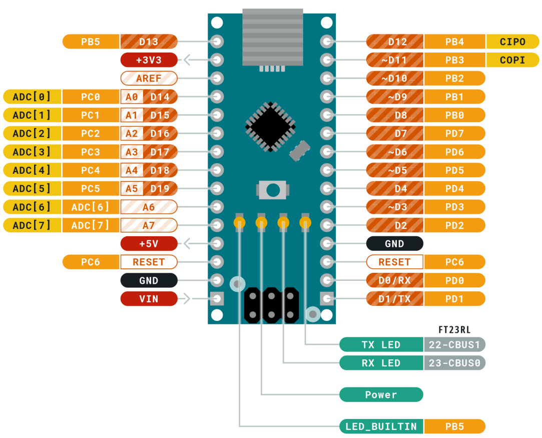

Figure 1: Arduino Nano Pinout

2.2 Voltage Measurement

The ADC of Ardunino nano is used to measure voltage. The ADC offers a resolution of 10-bits which is sufficient for basic use but not enough where the high precision is required.

Based on the voltage reference source, the ADC calculates the value , by default it is set to 5V but can be changed to either internal voltage reference or external voltage reference using the AREF pin.

The voltage reference can be set using the '**analogReference()**' function where '**INTERNAL**' is used to set reference to 1.1volts and '**EXTERNAL**' for voltage applied to the AREF pin (0 to 5V only) to be used as the reference.

Formula used for Voltage measurement :

$$\textbf{Voltage} = (\textbf{ADCValue}/2^{resolution})*\textbf{V}_{ref},$$

ADCValue is the raw ADC value, resolution is the ADC resolution (10 bits in this case), and Vref is the reference voltage.

The given below circuit represents how voltage can be measured using ADC.This is the most basic circuit for measuring voltage including a pull down resistor to avoid floating values.

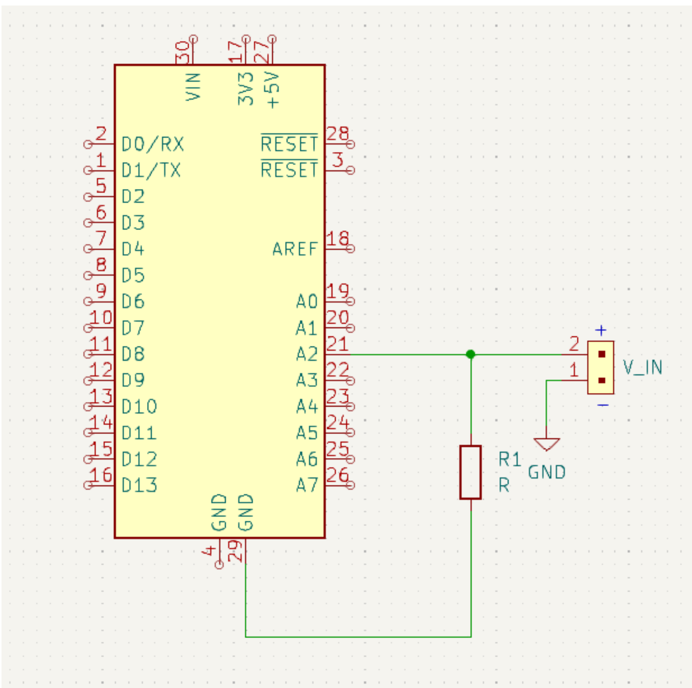

Figure 2: Voltage measurement circuit

2.3 Basic Circuit

The basic circuit of this project consists of a resistor divider circuit dividing the incoming internal voltage through AREF pin(1.1 volts) and feeding it to the ADC pin for measuring.The precision required here is more so we are oversampling the ADC to 16-bit resolution.

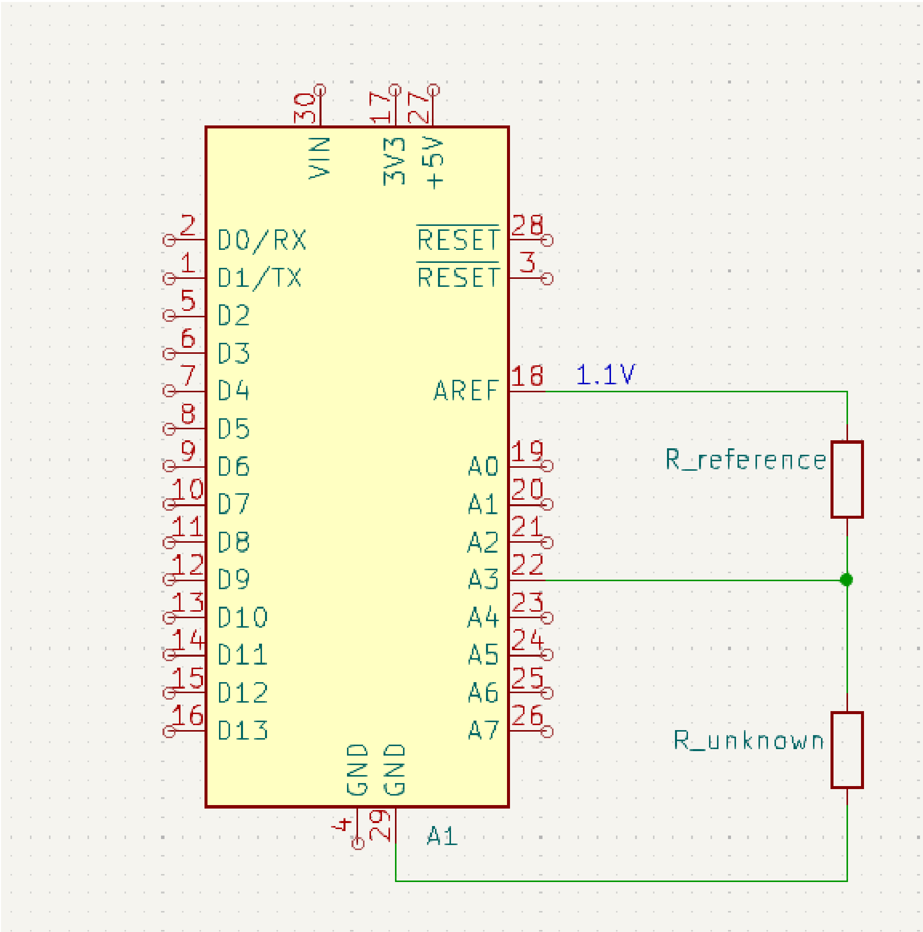

Figure 3: Resistor Divider Circuit

Description

The figure shows two resistors forming a resistor divider, with one reference resistor having a known value and the other having an unknown value. The value of voltage dropped across them is used to classify them into different ranges (1%, 0.5%, or 0.1%).

3. Oversampling and Decimation

This technique requires a higher amount of samples and these can be achieved by oversampling the signal. For each additional bit of resolution, the signal must be oversampled four times. To get a better representation of the signal it is necessary to oversample this much because a larger amount of samples gives a better representation of the input signal when averaged.

On the other hand, decimation is the process of reducing the sampling rate of a signal by discarding some samples while preserving the essential characteristics of the signal. Decimation is often used after oversampling to restore the signal to its original sampling rate or to reduce computational requirements in certain applications.

$\mathrm{F}_{oversampling} = 4^n * F_{nyquist}$

3.1 Basic Code for Oversampling

```
float Oversampling(uint8_t x) {
  byte NOS = 2;\\Number of Samples (NOS)
  long readingSum = 0;
  for (uint8_t i = 0; i < NOS; i++) { //Loop for Averaging
    long inner = 0;
  for (uint16_t j = 0; j < 4^n ; j++) { //Oversampling loop
      inner += analogRead(x);
      delayMicroseconds(150);
    }
    long reading = ((inner) >> n); // Decimation
    readingSum += reading;
  }
  float average = readingSum / NOS;
  return average;
}
```

Description

The oversampling function contains two loops, loop 1 is for averaging which is defined to take an average of two samples and the inner loop is the oversampling loop where 'n' defines the number of desired bits.

4. Calibration of ADC

4.1 Need for Calibration

The total error of the actual ADC comes from more than just quantization error, known as gain error and offset error.

For most applications, the ADC needs no calibration when used in the single-ended conversion. However, when using differential conversion the situation changes, especially with high-gain settings. Minor process variations are scaled. Therefore calibration in ADC is required.

4.2 Offset Error

The offset error, also known as zero-scale error or DC offset error, refers to the deviation from zero output when the input signal is at or near zero. In other words, it represents an error in the ADC's ability to accurately represent zero input as a zero output. The offset error is typically specified in LSB (Least Significant Bit) or in volts.

4.3 Gain Error

The gain error refers to the deviation from the ideal gain or slope of the transfer function of the ADC. It represents an error in the ADC's ability to accurately amplify and scale the input signal. The gain error is usually expressed as a percentage or in LSB or volts.

To compensate for the gain error, you can use calibration techniques or adjust the output of the ADC by applying a correction factor to the measured values.

Both offset error and gain error contribute to the overall accuracy of an ADC. They can affect the linearity, precision, and resolution of the converted digital output. It's important to consider these errors when designing or using ADCs in applications that require high accuracy.

The calibration of the Resistor Sorter was done with respect to $6\frac{1}{2}$ **digit** a DMM (Digital Multi Meter) and a non-linear regression model, curve-fitting was done using machine learning.

4.4 Non-linear Regression Model

Non-Linear regression is a type of polynomial regression. It is a method to model a non-linear relationship between the dependent and independent variables. It is used in place when the data shows a curvy trend and linear regression would not produce very accurate results when compared to non-linear regression. This is because in linear regression it is pre-assumed that the data is linear.

There are many different regressions that exist and can be used to fit whatever the dataset looks like such as quadratic, cubic regression, and so on to infinite degrees according to our requirement.

The oversampling of ADC to 16-bit introduces some non-linearity observed in the before calibration curve

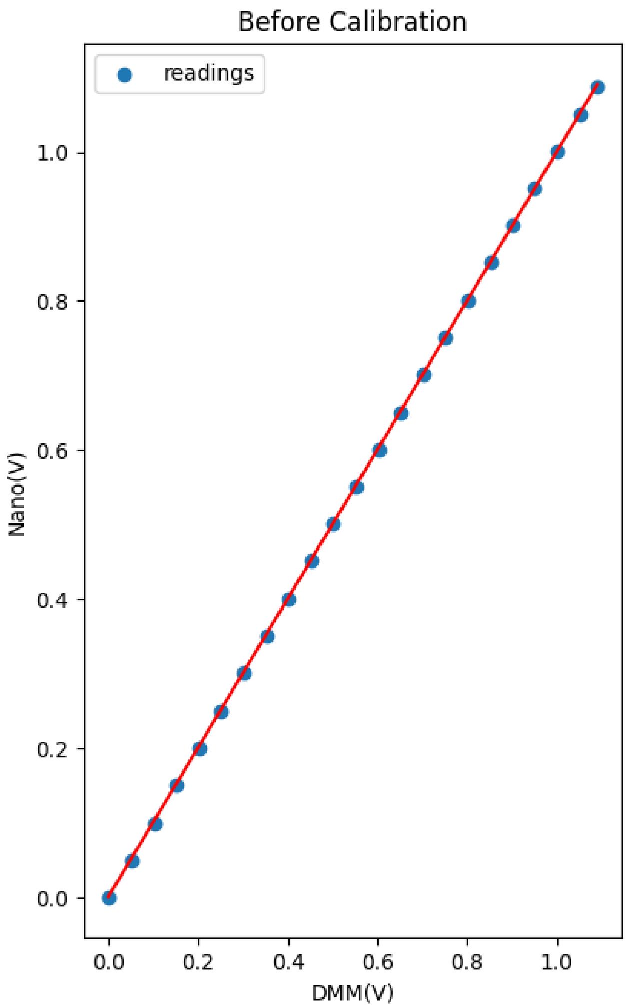

Figure 4: Before Calibration

4.5 Machine Learning Code

```
import matplotlib.pyplot as plt
import numpy as np
from scipy.optimize import curve_fit
import pandas as pd

data1= pd.read_csv('y=x(mv).csv')
x=data1['x'].values
y=data1['y'].values

data3 = pd.read_csv('NanoCalib.csv')
p=data3['dmm'].values
q=data3['nano'].values

data2=pd.read_csv('NanoAfterCal.csv')
r=data2['dmm'].values
t=data2['nano'].values

plt.subplot(1,3,1)
plt.xlabel("DMM(V)")
plt.ylabel("Nano(V)")
plt.title("Before Calibration")
plt.scatter(p,q,label='readings')
plt.plot(x,y,color='red')
plt.legend()

plt.subplot(1,3,2)
plt.xlabel("DMM(mV)")
plt.ylabel("Nano(mV)")
plt.title("After Calibration")
plt.scatter(r,t,label='reading')
plt.plot(x,y,color='red')
plt.legend()

def model_func(x, a, b, c):
    return a * np.exp(b * x) + c

data = pd.read_csv('NanoCalib.csv')
multimeter_readings = data['dmm'].values
adc_voltages = data['nano'].values

x_data = np.array(multimeter_readings)
y_data = np.array(adc_voltages)

popt, pcov = curve_fit(model_func, x_data, y_data,maxfev=5500)
a_opt, b_opt, c_opt = popt
y_fit = model_func(x_data, *popt)

plt.subplot(1,3,3)
plt.scatter(x_data, y_data, label='Data')
plt.plot(x_data,y_fit, 'r-', label='Fitted curve')
plt.legend()
plt.xlabel('DMM(V)')
plt.ylabel('Nano(V)')
plt.title('Non-linear Curve Fitting')
plt.show()
```

The Model function used here is exponential function with parameters **a,b,c** determining the next value.

The After Calibration results can be observed to be much more better curve fitting.

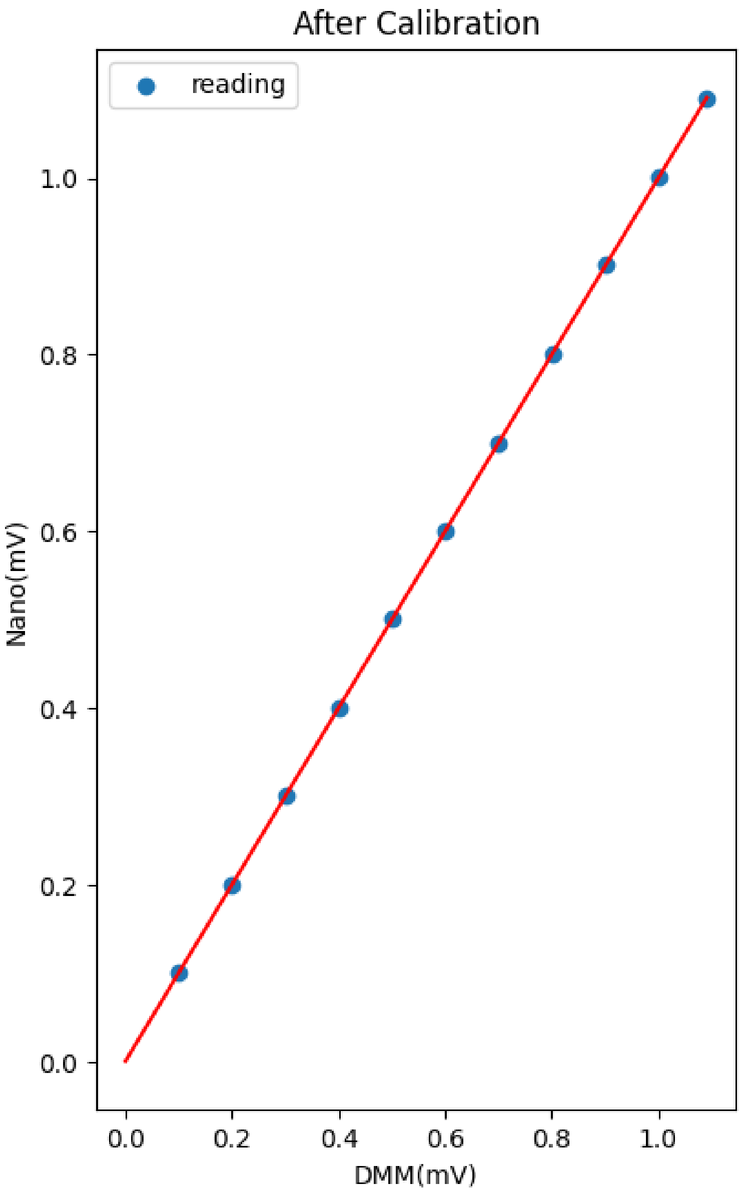

Figure 5: After Calibration

The number of points seen on the before calibration curve are different from the ones seen on after calibration curve to remove the overfitting condition.

Overfitting is a common issue in machine learning and artificial intelligence (AI) that occurs when a model learns to perform exceptionally well on the training data but fails to generalize effectively to new, unseen data.

5. Project Description

5.1 Circuit Diagram

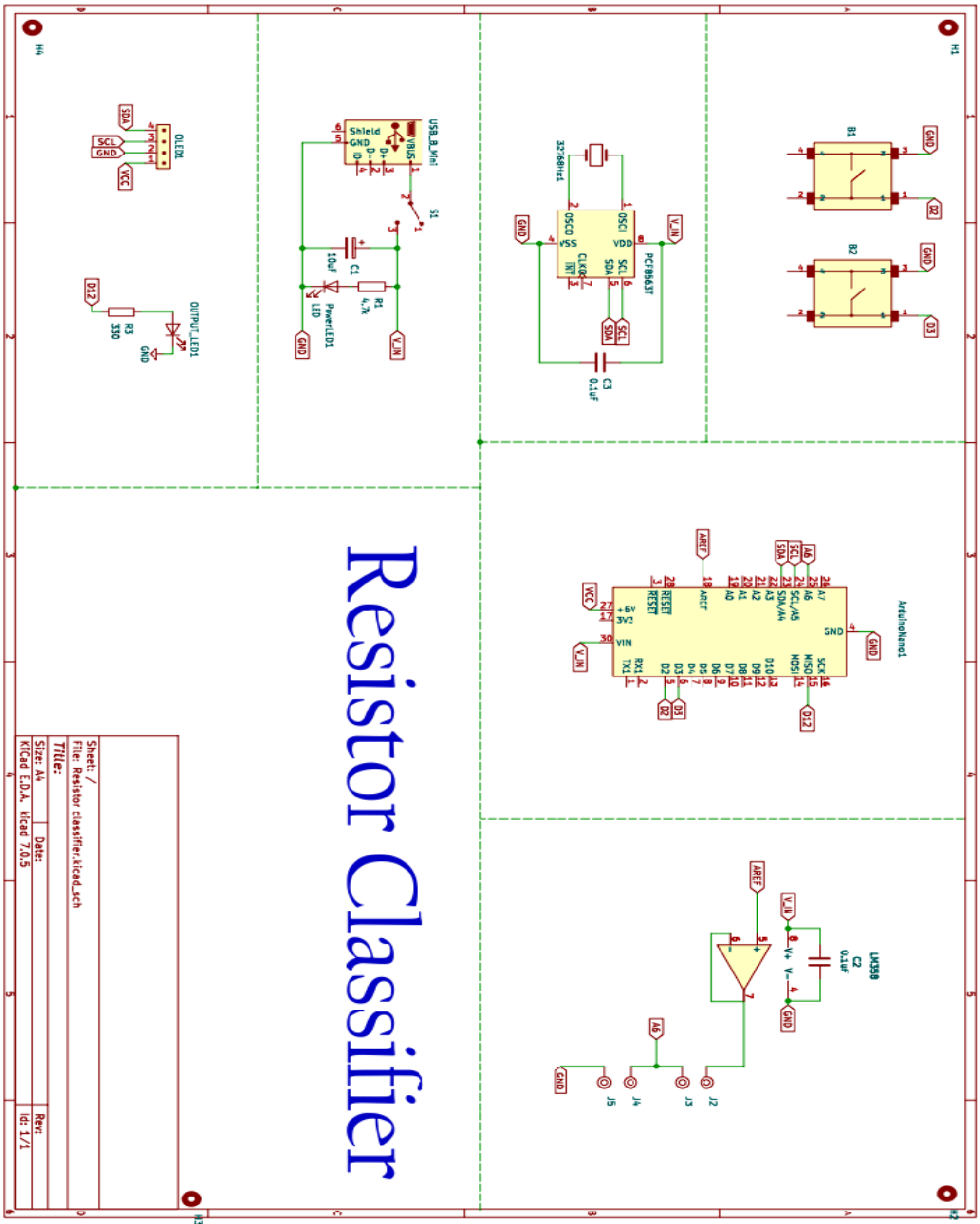

Figure 6: Resistor Sorter Circuit Diagram

5.2 Working

The circuit consists of an Arduino board set to an internal analog voltage reference. The same 1.1 volts are used as a constant voltage source to make a resistor divider with known and unknown resistors. The voltage is passed through an op-amp buffer because it prevents one stage's input impedance from loading the prior stage's output impedance, which causes an undesirable loss of signal transfer. Then the voltage is read by the 16-bit oversampled ADC, and further, the resistor is classified 1%, 0.5% ,0.1% respective ranges of 1%, 0.5%, and 0.1%. PCF8563 (RTC) is also interfaced within the circuit to attach the time also with the count of resistors for this a crystal oscillator (32768HZ) was also used. These values are stored in the EEPROM of the Arduino Nano, so they are retained for further use.

5.3 Sampling Rate

In the averaging loop for oversampling, an average of two samples is taken and the sampling rate is calculated, which comes out to be 2 samples in 992 ms. Considering the resolution it is providing, the sampling rate can be compromised.

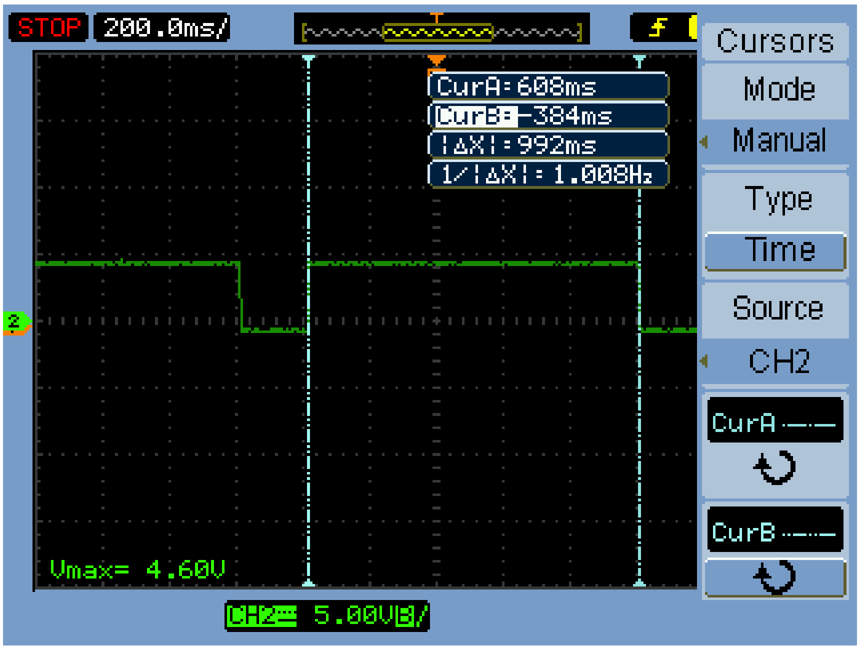

Figure 7: Arduino sampling Rate

5.4 Code

```
#include <EEPROM.h>
#include <Wire.h>
#include <PCF8563.h>
#include <Adafruit_GFX.h>
#include <Adafruit_SSD1306.h>

#define SCREEN_WIDTH 128 // OLED display width, in pixels
#define SCREEN_HEIGHT 32 // OLED display height, in pixels


Adafruit_SSD1306 display(SCREEN_WIDTH, SCREEN_HEIGHT, &Wire, -1);
PCF8563 rtc;

#define NOS 2
int cnt = 1;
int cntR1 = 0 ;
int cntR2 = 0;
int cntR3 = 0 ;
byte cntR1_ = 0;
byte cntR2_ = 0;
byte cntR3_ = 0;
byte cnt1 = 0;
int cntG = 0;

float a = 440.12228838689504, b = 0.0022761981308968937, c = -440.12421524369694; //calibration values


bool isWithin(float inputNumber, float targetNumber, float x) { // Range Justifying Function
  float percentageDifference = abs((inputNumber - targetNumber) / targetNumber) * 100.0;
  return (percentageDifference <= x);
}


double Oversampling16bit() { \\ Oversampling Loop

  long readingSum = 0;
  for (uint8_t i = 0; i < NOS; i++) {
    long innerSum = 0;
    for (uint16_t j = 0; j  < 4096; j++) {
      innerSum += analogRead(A6);
      delayMicroseconds(10);
    }
    long Reading = ((innerSum) >> 6); \\ Decimation
    readingSum += Reading;
  }
  float average = ((readingSum * (1.00)) / NOS);
  float v = average * (0.000016632080078125);

  float q = v;
  v = (log((v - c) * (1.00) / a)) / b;
  return (q > 0) ? v  : 0;
}
```

```

void setup() {
  Serial.begin(9600);
  if (!display.begin(SSD1306_SWITCHCAPVCC, 0x3C)) {
    Serial.println(F("SSD1306 allocation failed"));
    for (;;);
  }
  display.clearDisplay();
  display.setTextSize(1);
  display.setTextColor(WHITE);
  analogReference(INTERNAL);//1.092
  pinMode(12, OUTPUT);
  pinMode(A6, INPUT);
  pinMode(2, INPUT_PULLUP);
  pinMode(3, INPUT_PULLUP);

  rtc.init();
  rtc.stopClock();
  rtc.setYear(23);
  rtc.setMonth(7);
  rtc.setDay(10);
  rtc.setHour(23);
  rtc.setMinut(59);
  rtc.setSecond(0);

  rtc.startClock();
  display.setCursor(25, 6);
  display.println("Resistor");
  display.setCursor(35, 20);
  display.print("Sorter");
  display.display();
  delay(2000);
  display.clearDisplay();
}

float roundFloat(float value, int digits) { // Round off to desired
    digits
  float roundedValue = roundf(value * pow(10, digits)) / pow(10,
    digits);
  return roundedValue;
}

void loop() {

  Time nowTime = rtc.getTime();

  float vol = Oversampling16bit();
  float newn = roundFloat(vol, 5);
  float x;
  bool f = false;

  Serial.print("  Voltage : ");
  Serial.println(newn, 5);

  if (digitalRead(3) == LOW) {
    display.clearDisplay();
    f = true;
```

```
    cnt1 = 0;
    cntG++;
  }

  if (digitalRead(2) == LOW && digitalRead(3) == LOW) {
    display.setCursor(15, 14);
    display.print("Values Stored!");
    display.display();
    delay(750);
    Store();
    cntG = 1;
  }

  if (cntG == 1 && cnt1 == 0) {
    Read();
    display.setCursor(5, 20);
    display.fillRect(0, 20, cntR1_, 5, WHITE); // draw the bar graph
   for value 1
    display.setCursor(cntR1_ + 5, 20);
    display.print(cntR1_);
    display.print("(1%)");
    display.display();
    delay(100);
  }
  else if (cntG == 2 && cnt1 == 0) {
    Read();
    display.setCursor(5, 20);
    display.fillRect(0, 20, cntR2_, 5, WHITE);
    display.setCursor(cntR2_ + 5, 20);
    display.print(cntR2_);
    display.print("(0.5%)");
    display.display();
    delay(100);
  }
  else if (cntG == 3 && cnt1 == 0) {
    Read();
    display.setCursor(5, 20);
    display.fillRect(0, 20, cntR3_, 5, WHITE);
    display.setCursor(cntR3_ + 5, 20);
    display.print(cntR3_);
    display.print("(0.1%)");
    display.display();
    delay(100);
  }
  else if (cntG == 4 && cnt1 == 0) {
    Read();
    display.setCursor(5, 20);
    display.fillRect(0, 20, cntR3_ + cntR2_ + cntR1_, 5, WHITE);
    display.setCursor(cntR3_ + cntR2_ + cntR1_, 20);
    display.print(" ");
    display.print(cntR3_ + cntR2_ + cntR1_);
    display.print(" (Total)");
    display.display();
    delay(100);
  }
  else {
    cntG = 0;
```

```
  }

  if (digitalRead(2) == LOW) {
    display.clearDisplay();
    f = false;
    cnt1 = 1;
    cnt++;

  }

  if (cnt == 1 && cnt1 == 1) {
    display.setCursor(34, 5);
    display.print("Mode 1(1%):");
    display.display();
    x = 1.0;

  }
  else if (cnt == 2 && cnt1 == 1) {
    display.setCursor(30, 5);
    display.print("Mode 2(0.5%):");
    display.display();
    x = 0.5;

  }
  else if (cnt == 3 && cnt1 == 1) {
    display.setCursor(30, 5);
    display.print("Mode 3(0.1%):");
    display.display();
    x = 0.1;
  }
  else {
    cnt = 0;
  }

  if (isWithin(newn, 0.545, x) == true && cnt1 == 1) {
    if (x == 1.0) {
      cntR1++;

    }
    else if (x == 0.5) {
      cntR2++;


    }
    else if (x == 0.1) {
      cntR3++;

    }

    digitalWrite(12, HIGH);
    Serial.print("YES");
    delay(1000);
  }
  else {

    digitalWrite(12, LOW);
    Serial.print("  NO");
```

```

  }
}

void Store() { // Function to store value in EEPROM

  Time nowTime = rtc.getTime();
  EEPROM.write(0, nowTime.day);
  EEPROM.write(1, nowTime.month);
  EEPROM.write(2, nowTime.year);
  EEPROM.write(3, nowTime.hour);
  EEPROM.write(4, nowTime.minute);
  EEPROM.write(5, nowTime.second);
  EEPROM.write(6, cntR1);
  EEPROM.write(7, cntR2);
  EEPROM.write(8, cntR3);
  cntR1_ = 0;
  cntR2_ = 0;
  cntR3_ = 0;
}

void Read() { //Function to read value in EEPROM

  byte day1 = EEPROM.read(0);
  byte month1 = EEPROM.read(1);
  byte year1 = EEPROM.read(2);
  byte hour1 = EEPROM.read(3);
  byte minute1 = EEPROM.read(4);
  byte second1 = EEPROM.read(5);
  cntR1_ = EEPROM.read(6);
  cntR2_ = EEPROM.read(7);
  cntR3_ = EEPROM.read(8);
  display.clearDisplay();
  display.setCursor(5, 5);
  display.print(hour1);
  display.print(":");
  display.print(minute1);
  display.print(":");
  display.print(second1);
  display.print("");
  display.print("(");
  display.print(day1);
  display.print("/");
  display.print(month1);
  display.print("/");
  display.print(year1);
  display.println(")");
}
```

5.5 Component list

Resistor	Capacitor	Semiconductor	Miscellaneous
R1 - 4.7kΩ R2- 330Ω	C1 - 10uF C2 - 0.1uf C3 - 0.1uf	LED1 LED2 LM358 PCF8563 ATMega 328	OLED 40xx - 2x USB Mini Banana Jack - 4x

5.6 PCB Layout

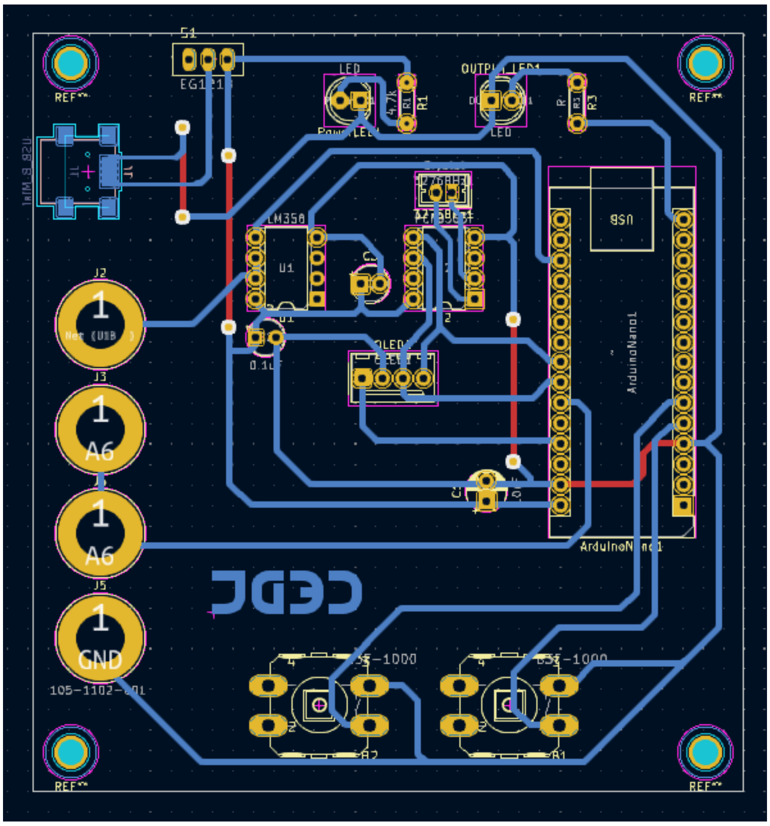

Figure 8: Resistor Sorter PCB Layout

6. References

- Arduino Nano Documentation
  https://docs.arduino.cc/hardware/nano
- Enhancing ADC resolution by oversampling
  https://ww1.microchip.com/downloads/en/appnotes/doc8003.pdf
- Characterization and Calibration of the ADC on an AVR
  https://ww1.microchip.com/downloads/en/Appnotes/Atmel-2559-Characterization-and-Calibration-of-the-ADC-on-an-AVR_ApplicationNote_AVR120.pdf
- Non Linear Regression Model
  https://www.geeksforgeeks.org/non-linear-regression-examples-ml/