MULTIMETER

PROJECT TECHNICAL REPORT

Making a digital multimeter using Arduino UNO Microcontroller

جامعة الإسكندرية
ALEXANDRIA UNIVERSITY

Assigned to: Dr. Thanaa

Contents

1. Introduction

Measurement lies at the heart of engineering, serving as the cornerstone for design, analysis, and validation across various disciplines. In the realm of electrical engineering, the accurate quantification of voltage, current, and resistance is paramount for ensuring the functionality and safety of electronic systems. While analog measurement techniques have historically been prevalent, the advent of digitalized measurement offers several distinct advantages in today's technological landscape. In particular, digital multimeters provide enhanced precision, versatility, and ease of integration with modern electronic systems. Therefore, the decision to pursue the development of a digital multimeter using the Arduino Uno microcontroller platform is a natural progression, aligning with the growing demand for advanced measurement instrumentation in engineering applications.

The report meticulously explicates the core concepts underlying digital multimeters, encompassing the theoretical foundations of voltmeter, ammeter, and ohmmeter functionalities. Furthermore, it delves into the intricacies of the algorithms and software testing procedures employed to translate these concepts into operational capabilities on the Arduino Uno platform. To enhance comprehension, operational principles are elucidated through Tinkercad simulations and examination of the constructed hardware model alongside its corresponding outputs.

2. Digital Multimeter Using Arduino Uno Microcontroller

- **2.1. Main Concept**

Used as a multimeter, the Arduino Uno functions as the central processing unit responsible for acquiring, processing, and displaying measurement data. The following points briefly illustrate how Arduino Uno functions:

1. Input Signal Acquisition: The Arduino Uno acquires the input signal from the device being measured, which could be voltage, current, resistance, or capacitance in the case of this multimeter model. The signal is then conditioned and scaled using external components, resistors in our model, to ensure compatibility with the Arduino Uno's analog-to-digital converter (ADC).

2. Analog-to-Digital Conversion (ADC): The conditioned input signal is then fed into one of the analog input pins of the Arduino Uno. The built-in ADC converts the analog voltage or current into a digital value with a certain resolution, typically ranging from 8 to 10 bits.

3. Measurement Processing: The digital value obtained from the ADC represents the magnitude of the input signal. Depending on the selected measurement mode (e.g., voltage, current, or resistance), the Arduino Uno executes specific measurement algorithms to process this digital value and calculate the corresponding measurement value.

4. Display Output: The calculated measurement value is then displayed on a digital screen connected to the Arduino Uno. The display may also show additional information, such as the measurement units.

- **2.2. Components**

In this project, the following components are used:

1. Arduino UNO microcontroller
2. I2C LCD
3. 10 Resistors (two 10kΩ, five 1kΩ, two 220Ω and one 22Ω)
4. Switches
5. Breadboard
6. Wires

- **2.3. Theory of Operation Explained**

2.3.1. Voltmeter

A voltage divider circuit connected with an Arduino Uno to work as a voltmeter typically consists of two resistors connected in series between the input voltage source and ground, with one end of the circuit connected to the Arduino Uno's analog input pin. The voltage across one of the resistors is then measured by the Arduino Uno's analog-to-digital converter (ADC) to determine the input voltage level. By selecting appropriate resistor values, the voltage divider circuit scales down the input voltage to a level suitable for accurate measurement by the ADC.

2.3.2. Ammeter

A current shunt resistor circuit connected with an Arduino Uno works as an ammeter by measuring the voltage drop across a precision resistor placed in series with the load carrying the current to be measured. This voltage drop is converted into a digital value by the Arduino Uno's ADC, processed to calculate the actual current level, and displayed on a digital screen as an ammeter reading.

2.3.3. Ohmmeter

This model follows a simple, yet effective principle based on Ohm's Law and voltage-divider techniques. The Arduino applies a known voltage across the resistor or component to be measured. This voltage is typically generated using one of the Arduino's digital output pins. By applying a known voltage across the resistor, we can establish a reference for measuring the resulting current. Once the voltage is applied, the resulting current flowing through the resistor or component is measured. This is done by using a current sensing technique. One common method is to place a current shunt resistor with the resistor being measured. The voltage drop across this known resistor is then measured using the Arduino's analog-to-digital converter (ADC). By knowing the value of the current shunt resistor and measuring the voltage drop across it, the current flowing through the circuit is calculated using Ohm's Law. With the current, now known, flowing through the resistor known, the resistance of the resistor being measured can be calculated using Ohm's Law, as well. In summary, by applying a known voltage across a resistor or component, measuring the resulting current, and using Ohm's Law to calculate the resistance, the Arduino can effectively function as an ohmmeter, providing accurate resistance measurements.

2.3.4. Capacitance meter

The working of capacitance meter involves charging a capacitor through a known resistor and measuring the time it takes for the capacitor to reach a certain voltage threshold. The following is a step-by-step explanation of how it measures capacitance:

1. Charging the Capacitor: The Arduino applies a known voltage to the capacitor under test through a series resistor. This resistor limits the current flowing into the capacitor to prevent damage to the Arduino and the capacitor.
2. Measurement of Voltage: As the capacitor charges, the voltage across it gradually increases. The Arduino continuously measures the voltage across the capacitor using one of its analog input pins.
3. Detecting Voltage Threshold: The Arduino monitors the voltage across the capacitor and waits until it reaches a predetermined threshold voltage. This threshold voltage is typically set to a percentage of the maximum voltage.
4. Measuring Time: Once the voltage across the capacitor reaches the threshold voltage, the Arduino starts a timer. It measures the time it takes for the voltage to reach the threshold, starting from the moment the voltage begins to rise.

5. Calculating Capacitance: Using the time taken for the voltage to reach the threshold and the known resistance value, the Arduino calculates the capacitance of the capacitor using the formula for the time constant of an RC circuit: $\tau = R \times C$, where τ is the time constant, R is the resistance, and C is the capacitance. Rearranging the formula gives $C = \tau / R$. Since τ is the time taken for the voltage to reach a certain percentage of the maximum voltage, and R is the known resistance, the Arduino can calculate C, the capacitance of the capacitor. In summary, by charging a capacitor through a known resistor and measuring the time it takes for the capacitor to reach a certain voltage threshold, an Arduino can effectively function as a capacitance meter, providing accurate capacitance measurements for various electronic components and circuits.

3. Implemented Algorithm

The constructed code illustrates an implementation of a multimeter that can measure voltage, current, resistance, and capacitance using Arduino UNO, providing a versatile tool for measuring different electrical parameters. It demonstrates the functionality of each measurement mode and how the Arduino interacts with external components to perform accurate measurements. In this section of the report, the code and its implementation illustration are provided.

- **3.1. The code**

```
#include <Wire.h>
#include <LiquidCrystal_I2C.h>
// Initializing the LCD library with the I2C address

LiquidCrystal_I2C lcd(0x27, 16, 2); // Setting the LCD address to 0x27 for a 16 chars and 2 line display

// Defining the pins for different measurements
const int voltagePin = A2;   // Analog pin for voltage measurement
const int currentPin = A0;   // Analog pin for current measurement
const int resistorPin = A1;  // Analog pin for resistance measurement
const int capacitancePin = A3; // Analog pin for capacitance measurement

// Digital pins for charging and discharging the capacitor
const int ChargePin = 13;
const int DischargePin = 11;

// Defining buttons
const int pushbutton1 = 2;   // Digital pin for current
const int pushbutton2 = 3;   // Digital pin for resistance
const int pushbutton3 = 4;   // Digital pin for voltage
const int pushbutton4 = 5;   // Digital pin for capacitance

// Variables to store button states
int button1state = 0;
int button2state = 0;
int button3state = 0;
int button4state = 0;
```

```
void setup() {
  lcd.init();   // Initializing the LCD
  lcd.backlight(); // Turning on the backlight
   // Setting pushbutton pins as inputs
  pinMode(pushbutton1, INPUT);
  pinMode(pushbutton2, INPUT);
  pinMode(pushbutton3, INPUT);
  pinMode(pushbutton4, INPUT);
  pinMode(ChargePin, OUTPUT);
  digitalWrite(ChargePin, LOW);

  // Printing a message to the LCD
  lcd.setCursor(0, 0);
  lcd.print("Multimeter");
  lcd.setCursor(0, 1);
  lcd.print("Cur Ohm Volt Cap");
}


void loop() {

  // Measuring voltage
  float Vin = 0;
  button3state = digitalRead(pushbutton3); // Reading state of the voltage
button
  if (button3state == HIGH)
  {
    // Reading analog value from voltage pin and convert to voltage
    float voltage_value3 = analogRead(voltagePin);
    float temp_V = voltage_value3 * (5.0 / 1023.0);
    Vin = 11 * temp_V;
    // Displaying measurements on LCD
    lcd.setCursor(0, 1);
    lcd.print("V:");
    lcd.print(Vin, 2); // Print with 2 decimal places
    lcd.print("V         ");
  }

 // Measuring current
  float current_value = 0;
  button1state = digitalRead(pushbutton1); // Reading state of the current
button
  if (button1state == HIGH)
```

```
 {
  // Reading analog values from current pins and calculate current
  float voltage_value0 = analogRead(currentPin);
  current_value = voltage_value0*(5/1023.0)*1000/0.91;
  // Displaying measurements on LCD
  lcd.setCursor(0, 1);
  lcd.print("I");
  lcd.print(current_value, 2); // Print with 2 decimal places
  lcd.print("mA      ");
 }

 // Measurintg resistance
 float Rx = 0;
 button2state = digitalRead(pushbutton2); // Reading state of the
resistance button
 if (button2state == HIGH)
 {
   // Reading analog value from resistor pin and calculate resistance
  float voltage_value2 = analogRead(resistorPin);
  Rx = (1100.0 / (voltage_value2 * (5.0 / 1023.0))) - 220.0;
  // Displaying measurements on LCD
  lcd.setCursor(0, 1);
  lcd.print("R:");
  lcd.print(Rx, 2); // Print with 2 decimal places
  lcd.print(" Ohm        ");
 }

  // Measuring capacitance
button4state = digitalRead(pushbutton4); // Reading state of the
capacitance button
if (button4state == HIGH)
{
 // Initializing variables for capacitance measurement
 unsigned long start_time;
 unsigned long elapsed_time;
 float microFarads;
 float nanoFarads;
 float r_ref = 10000.00;  // Setting the reference resistor value in ohms
 // Charging the capacitor
 digitalWrite(ChargePin, HIGH);

 // Recording the start time
 start_time = millis();
```

```
  // Waiting for the voltage to reach 2/3 of the supply voltage
  // This is equivalent to 648 on a 10-bit ADC (0-1023 range)
  while (analogRead(capacitancePin) < 648) {}
  // Calculating the elapsed time
  elapsed_time = millis() - start_time;

  // Calculating the capacitance in microfarads or nanofarads
  microFarads = ((float)elapsed_time / r_ref) * 1000;
  if (microFarads > 1)
  {
    // Displaying the capacitance in microfarads
    lcd.setCursor(0, 1);
    lcd.print("c:");
    lcd.print((long)microFarads);
    lcd.print("uF           ");
  }
  else
  {
    // Converting microfarads to nanofarads
    nanoFarads = microFarads * 1000.0;
    // Displaying the capacitance in nanofarads
    lcd.setCursor(0, 1);
    lcd.print("c:");
    lcd.print((long)nanoFarads);
    lcd.print("nF           ");
  }

  // Discharging the capacitor
  digitalWrite(ChargePin, LOW);
  pinMode(DischargePin, OUTPUT);
  digitalWrite(DischargePin, LOW);
  while (analogRead(capacitancePin) > 0) {}
    pinMode(DischargePin, INPUT);
}
delay(1000); // Delay for one second before taking another measurement
}
```

- **3.2. Algorithm Illustration**

  In the following points, the key components of the code and its implemented algorithm are illustrated:

1. **Library Inclusion and LCD Initialization**:
   - The code includes the necessary libraries, including the Wire library for I2C communication and the LiquidCrystal_I2C library for interfacing with the LCD.
   - The LCD is initialized with its I2C address and dimensions.
2. **Pin Definitions**:
   - Analog and digital pins are defined for voltage, current, resistance, and capacitance measurements, as well as for charging and discharging the capacitor and push buttons for selecting different measurements.
3. **Setup Function**:
   - Initializes the LCD, sets the backlight on, sets pushbutton pins as inputs, sets the charge pin as an output, and prints a message to the LCD to indicate that the multimeter is ready.
4. **Loop Function**:
   - Continuously loops through the measurement process for voltage, current, resistance, and capacitance.
5. **Voltage Measurement**:
   - Measures voltage using the voltage pin and converts the analog reading to a voltage value.
   - Displays the voltage value on the LCD when the corresponding button is pressed.
6. **Current Measurement**:
   - Measures current using the current pin and converts the analog reading to a current value.
   - Displays the current value on the LCD when the corresponding button is pressed.
7. **Resistance Measurement**:
   - Measures resistance using the resistor pin and converts the analog reading to a resistance value.
   - Displays the resistance value on the LCD when the corresponding button is pressed.

8. **Capacitance Measurement**:
    - Measures capacitance by charging and discharging a capacitor through a known resistor.
    - Measures the time taken for the capacitor to reach a certain voltage threshold.
    - Calculates the capacitance using the time constant formula and displays the capacitance value on the LCD when the corresponding button is pressed.
9. **LCD Display**:
    - Prints the measured values of voltage, current, resistance, or capacitance on the LCD screen with appropriate units (e.g., volts, milliamps, ohms, microfarads or nanofarads)

- **3.3. Software Testing**

The software testing phase of the multimeter project aimed to ensure the reliability, accuracy, and functionality of the Arduino-based multimeter's software implementation. Rigorous testing methodologies were employed to verify that the code performed as intended, producing accurate measurements across various input conditions. This section provides an overview of the software testing methods used during the development process, including unit testing, integration testing, functional testing, boundary testing, error handling testing, stress testing, and usability testing. Through systematic testing, potential software bugs, errors, and performance issues were identified and addressed, ultimately enhancing the overall quality and usability of the multimeter software.

1. Unit Testing: In this test, individual blocks of code were isolated to verify that they produced the expected outputs for given inputs. Each measurement function voltage, current, resistance, and capacitance were tested independently to ensure they provide accurate results. The values obtained were put in comparison with the values of the same quantities obtained from a different measuring apparatus.

2. Integration Testing: In this test, the interaction between different components of the code to ensure they work together as expected was tested. The constructed code of multimeter has been capable of giving accurate measurements and successfully display it on the LCD. However, since the LCD displayed the readings with less clarity when all the values were displayed together, we believed our system would be working optimally when different measuring quantities (e.g.: volt, and current) were individually displayed.

3. Functional Testing: This testing method verifies that the software meets its functional requirements. The entire system has been tested to ensure it performs the intended tasks correctly. For example, we tested the voltage measurement mode by applying different voltages to the input and verifying that the displayed values are accurate.

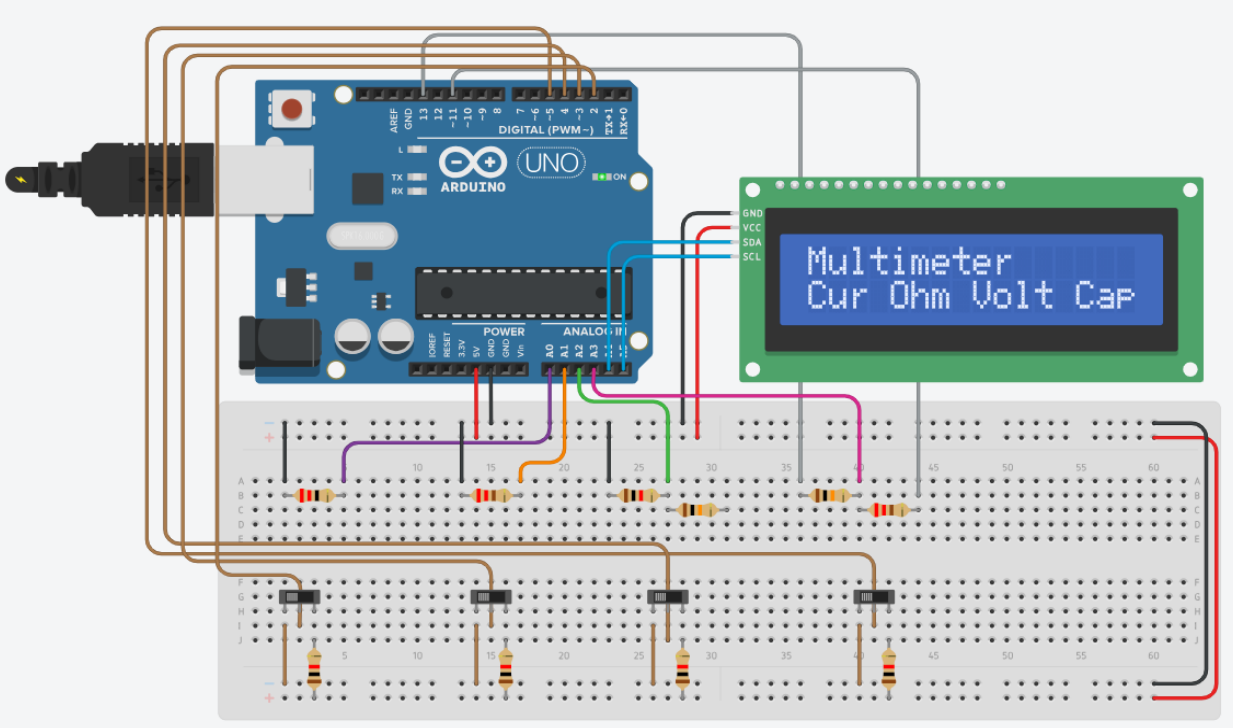
4. Boundary Testing: This testing method has only been tried on the hardware simulation only, as malfunction of the Arduino Uno microcontroller was avoided. That method tests the code at the boundaries of its input ranges to ensure it handles edge cases correctly. All the measuring quantities have been tested, and the following values are the edge cases of our multimeter apparatus:
   - Voltmeter: Can read up to 55 Volts.
   - Ammeter: Can read up to 227.27 mA
   - Ohmmeter: Can read up to 56k Ohm.
   - Capacitance meter: Can read up to 5000 uF, however, the reading can take one minute to be displayed.

Usability Testing: This was our last testing method, where both the code and its results on hardware were examined, and our apparatus has successfully passed the test and has proven to be easy to use, reliable and accurate.

4. Simulation& Hardware Operation

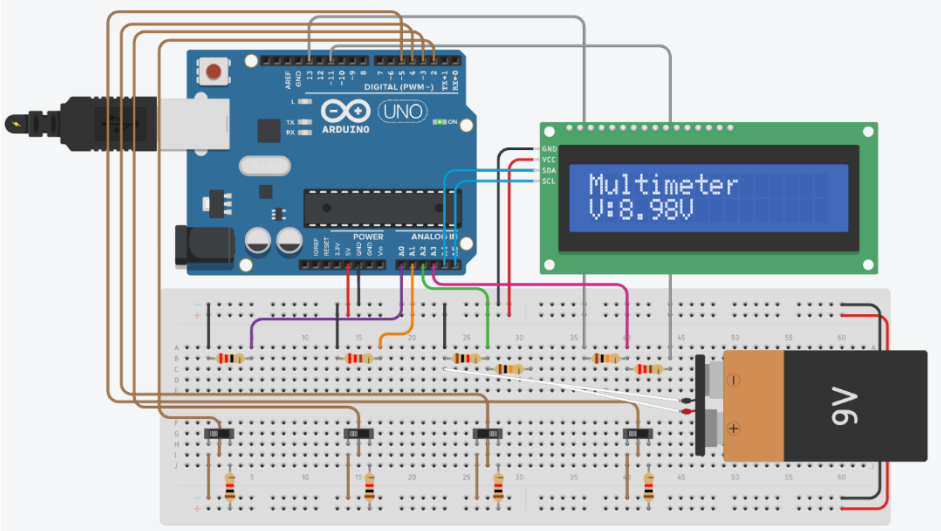
This section will be providing the report with visual representation of both the simulation and hardware.

- **4.1. Tinkercad Simulation and Outputs**



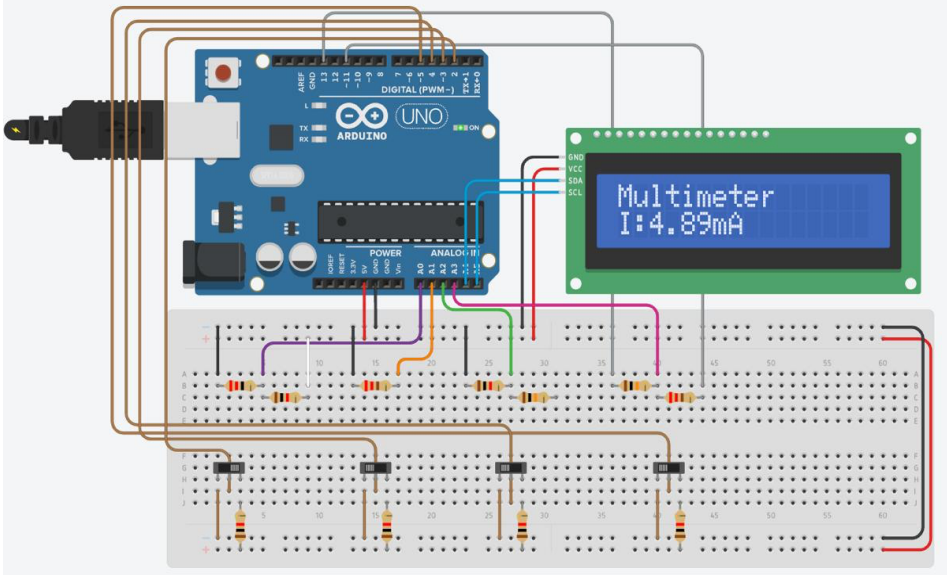
The given image represents the hardware connections of the constructed multimeter on Tinkercad. In this section, different outputs will be displayed, then compared with the hardware outputs to the same measured quantities as the ones used in that part in the following section.

1. Voltmeter



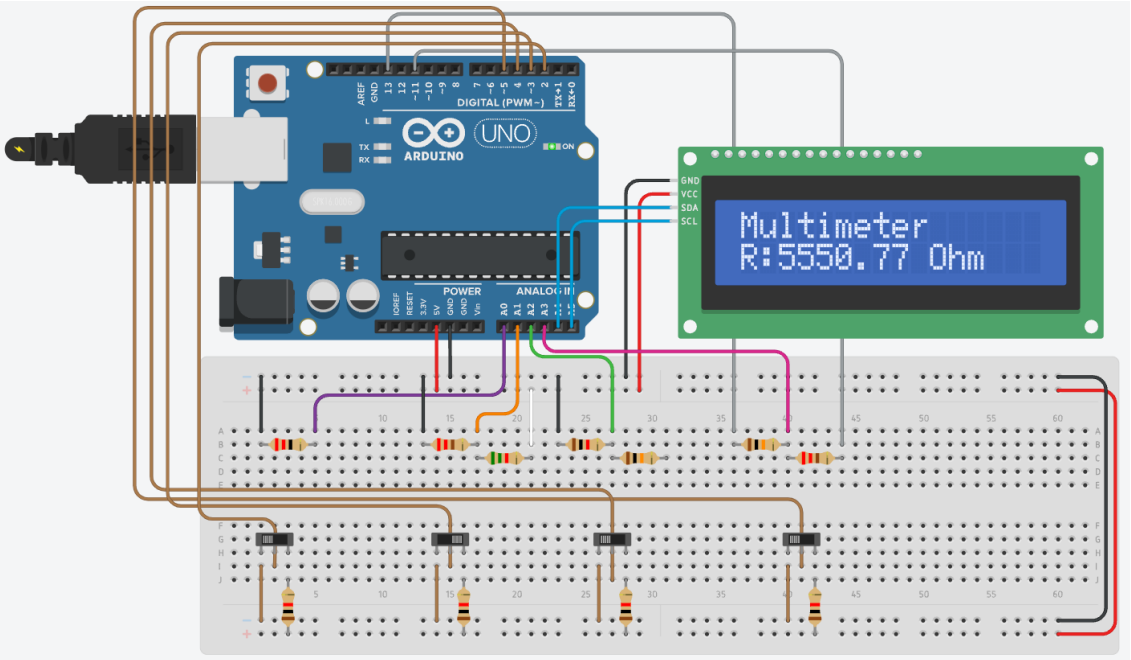
The true value of the measured voltage ≈ 9 Volts.

2. Ammeter



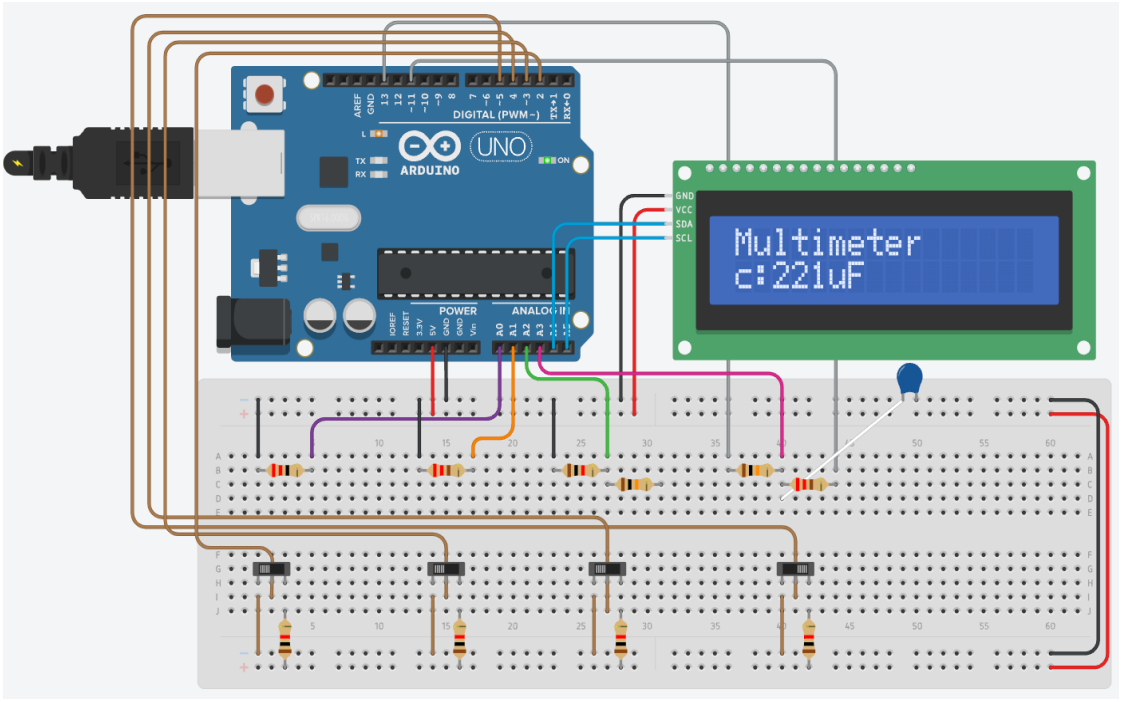
The true value of the measured current is ≈ 5 mA

3. Ohmmeter



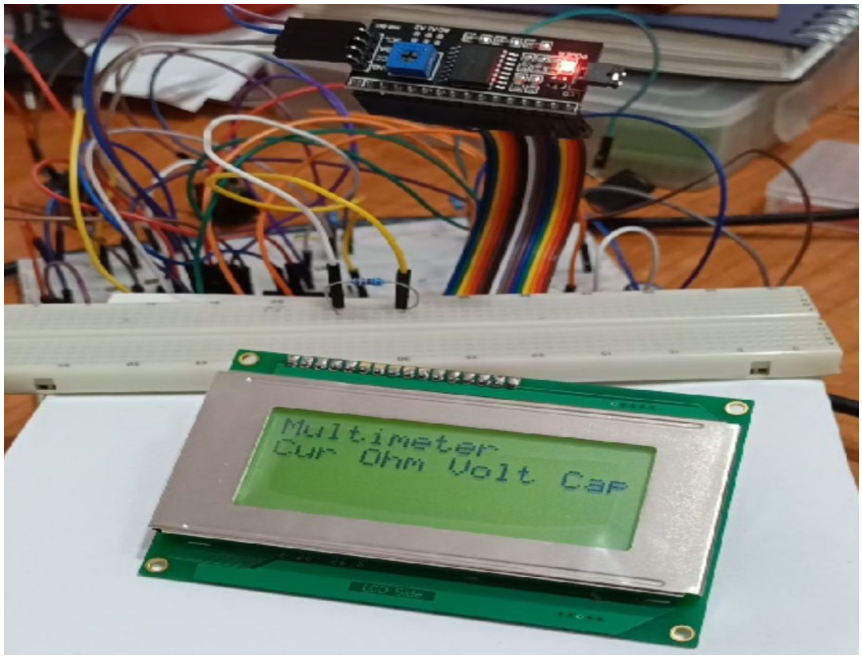
The real value of the tested resistor ≈ 5.5k Ohm.

4. Capacitance meter



The real value of the tested capacitor ≈ 220 uF.

- **4.2. Hardware Outputs**



The image above is the multimeter project hardware connection.
In the following points a comparison between the simulated and the actually-built system will take place, highlighting experimental error percentage in readings.

1. **Voltmeter**

The error in Voltmeter readings ≈ 5.7%

2. Ammeter

The Error in Ammeter readings ≈ -10%

3. Ohmmeter

The Error in Ohmmeter readings ≈ 0%

4. **Capacitance meter**

The Error in Capacitance meter readings ≈ 2.79%

In conclusion, our constructed digital multimeter apparatus gives high accuracy, as its error percentages are quite small.

5. Errors and Calibration

- **5.1. Overview**

This section of the report delves into the identification, analysis, and mitigation of errors encountered during the development of the Arduino-based multimeter, as well as the calibration procedures employed to ensure accurate measurements. Precision and accuracy are paramount in the realm of electronic measurement, and this section aims to address any discrepancies or inaccuracies observed during the testing phase. By examining the sources of error and implementing appropriate calibration techniques, the goal is to enhance the reliability and performance of the multimeter, ensuring its suitability for a wide range of electronic measurement applications.

- **5.2. Errors**

1. The first error faced after software testing was not getting any output displayed on the LCD in the hardware model. Through separate testing for each component in the circuit, it turned out the LCD needed to be replaced, as the malfunction was caused by the LCD screen.

2. The resistors values were chosen through try-and-error method, as the first used resistors reflected poor accuracy.

3. It has been noted that when no quantity-to-be-measured is connected to the apparatus, it gives random and constant values; however, it does not affect the quantity brought to measurement. For instance, the voltmeter may read 0.22Volts when on, but when a battery is measured the output does not need to be -0.22 Volts; the apparatus measures the right value. The precise outputs are the result of our efforts to eliminate that open circuit error by changing the values of the used resistors without affecting the measurement process accuracy.

- **5.3. Calibration**

The multimeter was calibrated to ensure the accuracy and reliability of its measurements. The calibration process involved comparing the measured values against known reference standards and making adjustments to eliminate any discrepancies. Validation tests were then performed to verify the effectiveness of the calibration adjustments, ensuring that the multimeter provides accurate and consistent measurements. The resistors values choice in the hardware circuit, as mentioned above, is the result of proper calibration and testing to ensure proper measurement values.

Team Members

Name	ID
Fairouz Abdelmonem Khamiees Abdelmonem	21010963
Gehad Mohamed Attia Mohamed	22010693
Kareema Ayman Hassan Barouma	22011845
Hania Mohamed Abdelmoniem	22011333
Nada Wael Shabaan Hassan	22011297
Mennatullah Attia Abdelfattah Attia	22011269
Heba Adel Mahrous	22011337
Asmaa Galal Ahmed	19015429