Design and Implementation of a Multi-Function Digital Multimeter Using an Arduino

Joseph Havens *
Department of Physics and Astronomy, University of Kansas, Lawrence, KS 66045, USA

In this paper, we describe the design and implementation of a multi-functional digital multimeter using an Arduino microcontroller. The device is capable of measuring voltage, current, capacitance, resistance, and continuity. By integrating a variety of measurement circuits with an Arduino, we can accurately capture the readings of these electrical properties and display the results via a serial monitor interface. The device uses transistors for control, voltage dividers for voltage measurement, and a shunt resistor for current measurement. The capacitance measurement utilizes an RC time constant method, and the resistance measurement is based on a voltage divider equation.

I. INTRODUCTION

A digital multimeter (DMM) is an essential tool in electronics and electrical engineering for measuring various electrical properties such as voltage, current, resistance, and capacitance. In this project, we demonstrate the design and implementation of a simple, cost-effective, and portable digital multimeter using an Arduino microcontroller. The device allows for the measurement of several electrical parameters, including voltage, current, capacitance, resistance, and continuity, and outputs the results to a serial monitor interface. This project is intended as an educational tool to provide a hands-on approach to understanding basic electronics.

II. METHODOLOGY

A. System Design

The system consists of an Arduino microcontroller (Arduino Uno), control pins for activating transistors, and measurement pins for various electrical properties. The system has the following key components:

- **Voltage Measurement**: A voltage divider circuit is used to reduce the input voltage to a level that can be read by the Arduino's analog input pin. The transistor connected to the voltage measurement pin acts as a switch, controlling the voltage divider's connection to the input signal.
- **Current Measurement**: A shunt resistor is used in series with the load to measure the voltage drop across it, which is then used to calculate the current according to Ohm's law.
- **Capacitance Measurement**: The time constant method is used, where the time it takes for the capacitor to charge to approximately 63% of the supply voltage is measured. This time is used to calculate the capacitance.
- **Resistance Measurement**: A voltage divider circuit is also used to measure the resistance by comparing the voltage drop across a known resistor with the total voltage.
- **Continuity Test**: A simple digital read of the continuity pin is used to determine whether there is continuity between two points.

B. Measurement Methodologies

1. *Voltage Measurement*

To measure voltage, the system uses a voltage divider consisting of two resistors. The voltage across one of the resistors is read by the Arduino, and the input voltage is calculated using the voltage divider formula. The system is activated by setting the transistor associated with the voltage measurement pin to HIGH.

2. *Current Measurement*

For current measurement, a shunt resistor is placed in series with the load. The voltage drop across the shunt resistor is measured by the Arduino, and Ohm's law is used to calculate the current. A transistor controls the connection to the shunt resistor, allowing for the measurement of current when required.

3. *Capacitance Measurement*

Capacitance is measured using the time constant method. The capacitor is charged through a resistor, and the time it takes for the capacitor to reach 63% of the supply voltage (3.15V for a 5V system) is measured. The capacitance is then calculated using the formula:

$$C = \frac{t}{R}$$

where t is the time to reach 63% of the voltage, and R is the resistance.

* joe.havens79@ku.edu

4. Resistance Measurement

Resistance is measured using a voltage divider circuit, where a known resistor is placed in series with the unknown resistor. The voltage drop across the known resistor is measured, and the resistance of the unknown resistor is calculated using the voltage divider equation.

5. Continuity Test

Continuity is checked by sending a signal through the test circuit and reading the response. If the continuity pin detects a LOW value, it indicates continuity; otherwise, there is no connection.

III. CODE ANALYSIS

This section provides a breakdown and analysis of the code used for the multi-functional digital multimeter built on an Arduino microcontroller. The code is structured to interface with external components, control measurement circuits, and send results to the Serial Monitor for display. Below we will discuss the system initialization, main loop, and measurement functions.

A. System Initialization

The initial part of the code defines the hardware configuration, setting up pins for the measurement modes and transistor controls. For example, the pins for voltage, current, capacitance, resistance, and continuity measurements are defined as follows:

Listing 1. Pin Definitions

```
const int voltagePin = A0;  // Voltage measurement pin
const int currentPin1 = A1; // Current measurement pin 1
const int capPin = A2;      // Capacitance measurement pin
const int resistancePin = A3; // Resistance measurement pin
const int continuityPin = 7;  // Continuity test pin
```

The `setup()` function initializes serial communication, sets the control pins for output, and configures the continuity pin with a pull-up resistor. Instructions are printed to guide the user through the measurement modes.

Listing 2. Setup Function

```
void setup() {
  Serial.begin(9600);
  Serial.println("Setup started");

  pinMode(tCurrent, OUTPUT);
  pinMode(tVoltage, OUTPUT);
  pinMode(tCapacitance, OUTPUT);
  pinMode(tResistance, OUTPUT);
  pinMode(tContinuity, OUTPUT);
  pinMode(continuityPin, INPUT_PULLUP);

  Serial.println("Setup completed");
  Serial.println("Enter measurement mode:");
  Serial.println("0 - Voltage");
  Serial.println("1 - Current");
  Serial.println("2 - Capacitance");
  Serial.println("3 - Resistance");
  Serial.println("4 - Continuity");
}
```

B. Main Loop

The `loop()` function is responsible for continually checking user input and activating the corresponding measurement mode. Based on the mode selected by the user, the relevant transistor is activated, and the appropriate measurement function is called:

Listing 3. Main Loop

```
void loop() {
  if (Serial.available() > 0) {
    mode = Serial.parseInt();  // Read mode selection from Serial Monitor
    Serial.print("Selected mode: ");
    Serial.println(mode);

    // Reset all transistors
    digitalWrite(tCurrent, LOW);
    digitalWrite(tVoltage, LOW);
    digitalWrite(tCapacitance, LOW);
    digitalWrite(tResistance, LOW);
    digitalWrite(tContinuity, LOW);

    // Process selected mode and execute measurement
    switch (mode) {
      case 0:
        digitalWrite(tVoltage, HIGH);
        measureVoltage();
        digitalWrite(tVoltage, LOW);
        break;
      case 1:
        digitalWrite(tCurrent, HIGH);
        delay(500);  // Short delay for transistor settling
        measureCurrent();
        digitalWrite(tCurrent, LOW);
        break;
      case 2:
        digitalWrite(tCapacitance, HIGH);
        measureCapacitance();
        digitalWrite(tCapacitance, LOW);
        break;
      case 3:
        digitalWrite(tResistance, HIGH);
        measureResistance();
        digitalWrite(tResistance, LOW);
        break;
      case 4:
        digitalWrite(tContinuity, HIGH);
        measureContinuity();
        digitalWrite(tContinuity, LOW);
        break;
```

```
      default:
        Serial.println("Invalid mode selected. Please enter a number between 0 and 4.");
        break;
    }

    delay(1000);  // Delay for stability
    while (Serial.available() > 0) {
      Serial.read();  // Clear input buffer
    }
  }
}
```

C. Measurement Functions

Each measurement function is designed to read from the corresponding sensor or circuit and calculate the appropriate value, which is then printed to the Serial Monitor. The functions include voltage, current, capacitance, resistance, and continuity.

For example, the `measureVoltage()` function reads the analog voltage from the voltage divider:

Listing 4. Voltage Measurement Function

```
void measureVoltage() {
  Serial.println("Measuring Voltage...");
  int rawValue = analogRead(voltagePin);
  Serial.print("Raw analog value (Out): ");
  Serial.println(rawValue);

  float V_out = (rawValue * 5.0) / 1023.0;
  Serial.print("Measured V_out: ");
  Serial.println(V_out, 2);

  float V_in = V_out * 5.0;
  Serial.print("Calculated V_in: ");
  Serial.println(V_in, 2);
}
```

D. Current Measurement

The **measureCurrent()** function measures the voltage drop across a shunt resistor to calculate the current using Ohm's law:

Listing 5. Current Measurement Function

```
void measureCurrent() {
  const float shuntResistorValue = 10.0;
  Serial.println("Measuring Current...");

  int rawValue1 = analogRead(currentPin1);
  Serial.print("Raw analog value: ");
  Serial.println(rawValue1);

  float voltageDrop = (rawValue1 * 5.0) / 1023.0;
  Serial.print("Voltage drop across shunt resistor: ");
  Serial.println(voltageDrop, 3);

  float current = voltageDrop / shuntResistorValue;
  Serial.print("Measured current: ");
  Serial.println(current, 4);
}
```

E. Capacitance Measurement

The `measureCapacitance()` function uses the RC time constant method to measure capacitance. The capacitor is charged through a resistor, and the time to reach 63% of the supply voltage is recorded:

Listing 6. Capacitance Measurement Function

```
void measureCapacitance() {
  const float resistorValue = 10000.0;
  Serial.println("Measuring Capacitance...");

  digitalWrite(tCapacitance, LOW);
  delay(1000);

  pinMode(capPin, INPUT);
  digitalWrite(tCapacitance, HIGH);

  unsigned long startTime = micros();
  while (analogRead(capPin) < (0.632 * 1023)) {
    // Wait until the capacitor voltage reaches 63% of Vcc
  }
  unsigned long elapsedTime = micros() - startTime;

  digitalWrite(tCapacitance, LOW);

  if (elapsedTime == 0) {
    Serial.println("Capacitance measurement failed. Check connections and capacitor.");
    return;
  }

  Serial.print("Elapsed Time (microseconds): ");
  Serial.println(elapsedTime);

  float capacitance = (float)elapsedTime / (resistorValue);
  Serial.print("Capacitance: ");
  Serial.print(capacitance, 2);
  Serial.println(" uF");
}
```

F. Resistance Measurement

The `measureResistance()` function measures resistance using a voltage divider setup. The unknown resistance is calculated based on the known resistor and the measured voltage drop:

Listing 7. Resistance Measurement Function

```
void measureResistance() {
  const int R1 = 1000.0;
  const int Vin = 5;
```

```
  Serial.println("Measuring Resistance...");
  digitalWrite(tResistance, HIGH);

  int val = analogRead(resistancePin);
  float voltage = (val * 5.0) / 1023.0;
  float resistance = (R1 * voltage) / (Vin -
      voltage);
  Serial.print("Resistance: ");
  Serial.print(resistance);
  Serial.println(" Ohms");

  digitalWrite(tResistance, LOW);
}
```

G. Continuity Test

The `measureContinuity()` function checks for continuity by reading the state of a digital pin connected to the test circuit. If the pin reads LOW, continuity is detected:

Listing 8. Continuity Test Function

```
void measureContinuity() {
  Serial.println("Measuring Continuity...");
  int val = digitalRead(continuityPin);
  if (val == LOW) {
    Serial.println("Continuity Detected!");
  } else {
    Serial.println("No Continuity.");
  }
}
```

Each measurement function ensures that the Arduino outputs accurate and readable data through the Serial Monitor, providing a versatile and educational tool for learning basic electronics and measurement techniques.

IV. RESULTS

The Arduino-based digital multimeter was successfully constructed and tested for all five measurement modes. The measurements of voltage, current, capacitance, resistance, and continuity were displayed accurately on the serial monitor. The capacitance measurement, in particular, utilized the time constant method, which proved to be effective for measuring small to medium-sized capacitors. The system also provided a stable measurement for resistance and current, with the voltage divider configuration producing reliable voltage readings.

V. ERROR ANALYSIS

In this section, we provide a detailed error analysis for the five measurement methods employed in the Arduino-based digital multimeter. Each method's error sources are identified and quantified to assess their impact on measurement accuracy. We also compare our findings to commercial multimeter specifications and discuss the assumptions made in the design.

A. Voltage Measurement

Voltage measurement relies on a voltage divider circuit and the Arduino's 10-bit ADC. The primary sources of error in voltage measurement are the ADC resolution, resistor tolerances, and transistor saturation effects.

1. *ADC Resolution*

The Arduino's ADC operates at 10-bit resolution, resulting in 1024 discrete levels for a 5V reference voltage. Each level corresponds to a voltage step of approximately 4.88 mV. This quantization error is expressed as:

$$\sigma_{\text{quantization}} = 4.88 \text{ mV}$$

This error is relatively small but can influence precision measurements, especially when dealing with low voltage levels near the ADC's resolution limit.

2. *Resistor Tolerances*

The voltage divider circuit is composed of resistors with a tolerance typically around 1%. This tolerance can contribute a significant error in the calculated voltage. The uncertainty in voltage due to resistor tolerance is:

$$\sigma_{\text{resistors}} = V_{\text{in}} \times 0.01$$

Where V_{in} is the input voltage. The voltage error becomes more prominent with higher voltage inputs, as the percentage error in resistance contributes more significantly.

3. *Total Uncertainty*

When I propagate the errors in V A 1 and V A 2, I find the following, shown in eqn. V A 3.

$$\sigma_V = \sqrt{(4.88 \text{ mV})^2 + (0.01 \times V_{\text{in}})^2}$$

For measurements in the 0-5V range, this results in an error of approximately 1%. In comparison, the Fluke 175/179 Multimeter has an error of about 0.9% for voltage measurements [1], showing that our DIY multimeter is performing with striking precision despite its simplified design.

B. Current Measurement

For current measurement, the main sources of error include the shunt resistor tolerance, the resolution of the ADC, and the voltage drop across the transistor.

1. Shunt Resistor Tolerance

The shunt resistor has a tolerance of about 1%, which can affect the accuracy of current measurements. The uncertainty in the current due to the resistor tolerance is:

$$\sigma_{R_{\text{shunt}}} = R_{\text{shunt}} \times 0.01$$

2. ADC Resolution

Similar to voltage measurement, the ADC introduces quantization error. For low currents, this error is more significant due to the small voltage drop across the shunt resistor.

$$\sigma_{\text{quantization}} = 4.88 \text{ mV}$$

3. Total Uncertainty

Once again, by propagating the errors we found in V B 1 and V B 2, the propagated uncertainty is shown in eqn. V B 3.

$$\sigma_I \approx \sqrt{\left(\frac{4.88 \text{ mV}}{R_{\text{shunt}}}\right)^2 + \left(\frac{V_{\text{drop}}}{R_{\text{shunt}}^2}\sigma_{R_{\text{shunt}}}\right)^2}$$

For the current measurements performed, the relative uncertainty is approximately 5%. A Fluke 175/179 Multimeter reports an error of approximately 1% for current measurements, meaning our current measurement method is less precise but still functional for general use.

C. Capacitance Measurement

The capacitance measurement method uses the RC time constant, where the capacitor charges through a resistor and the time it takes to reach 63% of the supply voltage is recorded. The primary sources of error include timing resolution, resistor tolerance, and capacitor tolerance.

1. Timing Resolution

The Arduino's 'micros()' function has a resolution of 4 microseconds, which limits the precision of timing measurements for small capacitors. The uncertainty in the timing measurement is given by:

$$\sigma_{\text{timing}} = 4\mu s$$

2. Resistor Tolerance

The resistor used in the RC circuit also has a tolerance, typically around 1% [1]. This contributes to the uncertainty in the time constant, as the charging rate of the capacitor depends on the resistance. The uncertainty due to the resistor tolerance is:

$$\sigma_R = R \times 0.01$$

3. Capacitor Tolerance

Capacitors typically have tolerances of 5-20%, depending on the type and quality. This variability can introduce a large uncertainty in the capacitance measurement.

The uncertainty in the capacitance measurement, propagating the error from V C 1 and V C 2, is shown in equation V C 3.

$$\sigma_C = \sqrt{\left(\frac{t}{R} \times \sigma_{\text{timing}}\right)^2 + \left(\frac{t}{R^2} \times \sigma_R\right)^2}$$

Where t is the time to reach 63% of the supply voltage.

During experimentation, we observed a significant discrepancy, with an experimental error of approximately 30% compared to the expected 1%. A Tenma LCR45 capacitance meter has an error of around 2-3% [2], so this large discrepancy suggests issues in the measurement technique. Further analysis would be needed to determine the cause of this, however capacitor tolerance could potentially be a source of this discrepancy, and transistor effects can also be a source.

D. Resistance Measurement

For resistance measurement, the primary sources of error include the resistor tolerance and the ADC resolution.

1. Resistor Tolerance

The tolerance of the known resistor R_{known} affects the accuracy of the resistance measurement. The uncertainty in the known resistor value is:

$$\sigma_{R_{\text{known}}} = R_{\text{known}} \times 0.01$$

2. ADC Resolution

Similar to previous methods, the ADC introduces quantization error, especially when measuring low resistance values where the voltage drop across the resistor is small.

$$\sigma_{\text{quantization}} = 4.88 \text{ mV}$$

3. Total Uncertainty

The combined uncertainty in the resistance measurement is shown in equation V D 3 by propagating V D 1 and V D 2.

$$\sigma_R \approx \sqrt{\left(\frac{4.88\text{ mV} \times R_{\text{known}}}{(V_{\text{in}} - V_{\text{out}})}\right)^2 + \left(\frac{V_{\text{out}} \times \sigma_{R_{\text{known}}}}{(V_{\text{in}} - V_{\text{out}})}\right)^2}$$

For resistance measurements made, the uncertainty is around 1%. A Fluke 175/179 Multimeter has an error of 0.9% [1], indicating that our resistance measurement system performs similarly to a commercial unit.

E. Continuity Test

The continuity test is based on a modified Darlington pair configuration of two transistors. This arrangement allows for a high current gain, but introduces complications due to the base-emitter voltage drops and transistor saturation effects.

1. Darlington Pair Configuration

In our continuity test, T1's collector runs from +5V through an LED and a resistor, then into the collector of T1. The emitter of T1 goes through the continuity test and into the base of T2, while the emitter of T2 connects to ground. This configuration amplifies current but introduces a higher base-emitter voltage drop, typically around 1.4V. This higher voltage drop causes a shift in the threshold voltage for continuity detection.

The false positive rate (FPR) for the continuity test is influenced by the threshold voltage and transistor saturation effects. The FPR can be modeled using the error function in equation V E 1.

$$P_{\text{FPR}} = \frac{1}{2}\left[1 + \text{erf}\left(\frac{V_{\text{data}} - (V_{\text{threshold}} + V_{BE1} + V_{BE2})}{\sigma_n\sqrt{2}}\right)\right]$$

Where:

- V_{data} is the measured voltage.
- $V_{\text{threshold}}$ is the nominal threshold voltage for continuity detection.
- V_{BE1} and V_{BE2} are the base-emitter voltages of T1 and T2, respectively.
- σ_n is the noise standard deviation, which accounts for electrical noise and ADC precision.
- erf is the Gauss error function, defined by $\text{erf}(z) := \frac{2}{\sqrt{\pi}}\int_0^z e^{-t^2}\,dt$

The combined uncertainty for continuity detection must account for transistor interactions, such as the effect of T1's saturation on T2's base voltage. The transistor's behavior, especially the saturation of T1, affects the threshold for detecting continuity. This can cause fluctuations in the readings and influence the false positive and false negative rates (FPR and FNR), making the continuity detection less reliable under certain conditions.

The noise standard deviation, σ_n, is a critical factor in determining the sharpness of the transition between detecting continuity and not detecting continuity. The interaction between the two transistors in the Darlington pair configuration introduces a noise floor that must be considered in the error function. This noise may stem from the Arduino's analog-to-digital conversion, power fluctuations, or the inherent electrical noise from surrounding components.

VI. DISCUSSION

This Arduino-based digital multimeter is an excellent tool for understanding basic electronics and circuit theory. The system is flexible, allowing for the addition of new measurement functions or the modification of existing ones. The use of transistors for switching between modes helps reduce component complexity, while the voltage divider and shunt resistor methods offer a straightforward way to measure electrical properties. The capacitance measurement, based on the RC time constant, provides an accurate method for capacitor testing. However, improvements can be made in terms of the precision of the time measurements and the integration of an automatic range feature.

A. Assumptions

It is important to note that many assumptions were made in the error analysis as well as in determining the optimal functioning ranges for each method. Firstly, the use of Gauss' error function in the FPR for the continuity test assumes that the noise profile is Gaussian in nature. This isn't necessarily the case, as I have no evidence to assert the nature of the noise profile. Additionally, while it was mentioned sporadically in the error analysis, the placement of the transistors off of the data lines helps to reduce or mitigate any contribution that the transistors may have on the error. As such, I assumed they had no effect, except in the continuity test where this was not a valid assumption.

VII. CONCLUSION

By understanding and quantifying the errors in each measurement method, I can improve the accuracy and reliability of the Arduino-based digital multimeter. Each

method's error sources need careful consideration, and appropriate measures should be taken to mitigate these errors where possible.

VIII. ACKNOWLEDGEMENTS

I would like to thank Charles Havens for his invaluable input during the design phase of this project.

Appendix A: Full Code

The full code for the digital multimeter is available in the following GitHub repository: `https://github.com/JHavens2187/DIY_Multimeter`

[1] *Models 175, 177 & 179 True RMS Multimeters: Users Manual*, Fluke Corporation, Everett, WA (2003), revision 1, 10/08.

[2] *Tenma 72-370 LCR Meter User Manual*, Tenma (2022), last edited by Stephen Sanders, Aug 05, 2022.