Department of Electrical & Electronic Engineering

Rajshahi University of Engineering & Technology

LAB Report

Course Code : **EEE 3100**

Course Title : **Electronics Shop Practice**

Experiment No. : 08

Experiment Name : Design and Implementation of Digital Arduino voltmeter

Submitted By:

Name : Alsadi Md. Promi (Milton)

Roll : 2101098

Section : B

Submitted To:

Asif Zaman Rizve

Lecturer

Department of Electrical & Electronic Engineering

Rajshahi University of Engineering & Technology

Date of Experiment : 27 May, 2025

Date of Submission : 17 June, 2025

Experiment No.: 08

Experiment Name: Design and Implementation of Digital Arduino voltmeter

Objective:

1.To develop a digital voltmeter using Arduino Uno capable of measuring DC voltages up to 50 volts with precision.
2.To use a voltage divider to safely reduce high input voltages to a level that the Arduino's analog pins can handle.
3.To acquire voltage readings through the Arduino's ADC and calculate the actual voltage in software for display.

Theory:
The Arduino Uno is based on the ATmega328P microcontroller and offers 6 analog input pins (A0–A5) which can read voltage levels in the range of 0–5V with a 10-bit resolution (1024 levels). This allows analog voltages to be converted into a digital value between 0 and 1023. However, to measure voltages beyond the 0–5V range, a **voltage divider circuit** is used to safely scale down higher voltages into the readable range of the Arduino. A voltage divider is a simple linear circuit that reduces the input voltage to a lower output voltage based on the ratio of two resistors. It follows the equation:

$$V_{out} = V_{in} \frac{R_2}{R_1 + R_2}$$

By selecting appropriate resistor values (R1 = 100kΩ, R2 = 10kΩ), input voltages up to 50V can be scaled down to a maximum of 5V, making them safe for the analog pin. The measured voltage is then scaled back in the software using the inverse ratio to get the actual input voltage. To display the measured voltage, a 16×2 LCD module is used in conjunction with an I2C (Inter-Integrated Circuit) interface module. The I2C interface simplifies communication with the LCD by reducing the number of required connections to just two lines: SDA (data) and SCL (clock). The LCD provides a real-time visual output of the voltage, formatted to two decimal places.

Figure 08.01: LCD display

Figure 08.02: 12C module

EEPROM (Electrically Erasable Programmable Read-Only Memory) is a non-volatile memory space available on the Arduino (1 KB on the Uno) that retains data even when power is removed. In this experiment, the EEPROM is used to store a series of voltage readings at specific time intervals. Each voltage value is stored at a unique memory address, and an index is maintained to track the number of stored values. Later, these values can be read and displayed by accessing the saved indices. To enable wireless communication, an HC-05 Bluetooth module is used. This module allows the Arduino to communicate with a smartphone using serial communication

(TX/RX pins). When paired with a compatible **Bluetooth terminal or control app,** the Arduino can transmit live voltage readings to the phone. This feature enables remote monitoring of voltage levels without needing a physical display or wired connection. The module operates at 3.3V logic level but is tolerant to 5V inputs on RX if a resistor divider is used. Data transmission uses ATmega328P's hardware serial port (usually via `Serial.begin()`), sending voltage values as strings over Bluetooth which are then displayed in real-time on the mobile application.This experiment combines analog signal acquisition, data scaling and conversion, digital display, non-volatile memory usage, and wireless data transmission into a single embedded system project. It not only demonstrates the functionality of individual Arduino peripherals (ADC, EEPROM, I2C, UART) but also provides practical exposure to interfacing external hardware such as sensors, displays, and communication modules. By saving voltage readings and wirelessly transmitting them, this system simulates real-world data acquisition and monitoring systems used in industrial and consumer electronics for automation, diagnostics, and logging.

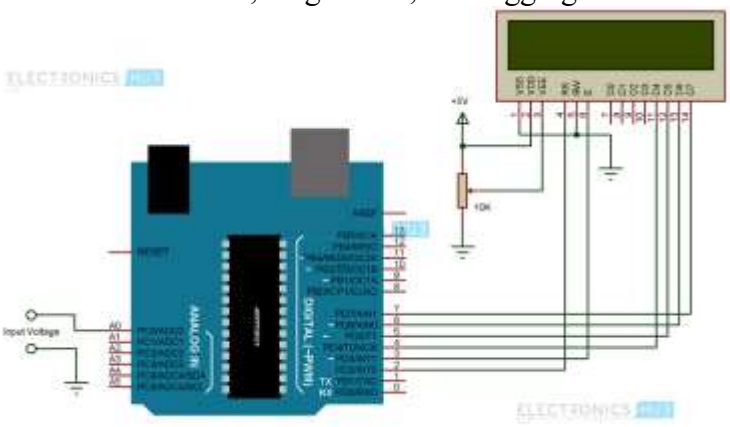

Figure 08.03: Simulation set up for Digital Arduino voltmeter

Required Apparatus:

Table 08.01: Table For Required Apparatus

Sl. No.	Components	Specification	Quantity
1	Arduino Uno	• ATmega328P • Operating Voltage: 5 V • Input Voltage: 7–12 V • Input Voltage: 6–20 V • Digital I/O Pins: 14 • PWM Digital I/O Pins: 6 (Pins 3, 5, 6, 9, 10, 11) • Analog Input Pins: 6 (A0 to A5) • DC Current per I/O Pin: 20 mA • DC Current for 3.3 V Pin: 50 mA • Flash Memory: 32 KB (ATmega328P) • SRAM: 2 KB (ATmega328P) • EEPROM: 1 KB (ATmega328P) • Clock Speed: 16 MHZ	1

Sl. No.	Components	Specification	Quantity
2	16x2 LCD Display	• Display Type: Alphanumeric LCD • Characters: 16 columns × 2 rows • Controller/Driver: HitachivHD44780 or compatible • Operating Voltage: 5 V DC • Character Size : ~2.95 mm × 5.55 mm • Module Size: 80 mm × 36 mm × 10 mm • Operating Temperature: -20°C to +70°C • Current Consumption: ~1.5 mA without backlight, ~20 mA with	1
3	I2C Module	• Interface Protocol: I2C (2-wire: SDA, SCL) • I2C Address (Default): 0x27 • Operating Voltage: 5 V DC • Pins Provided: VCC, GND, SDA, SCL • I2C Clock Frequency: 100 kHz or 400 kHz • Power Consumption: ~<5 mA	1
4	Resistance	100kΩ	1
5	Jumper wires	Male to Male wire Male to Female wire	20
6	IDE Software	Arduino IDE	---
7	Breadboard	---	1

Circuit Diagram:

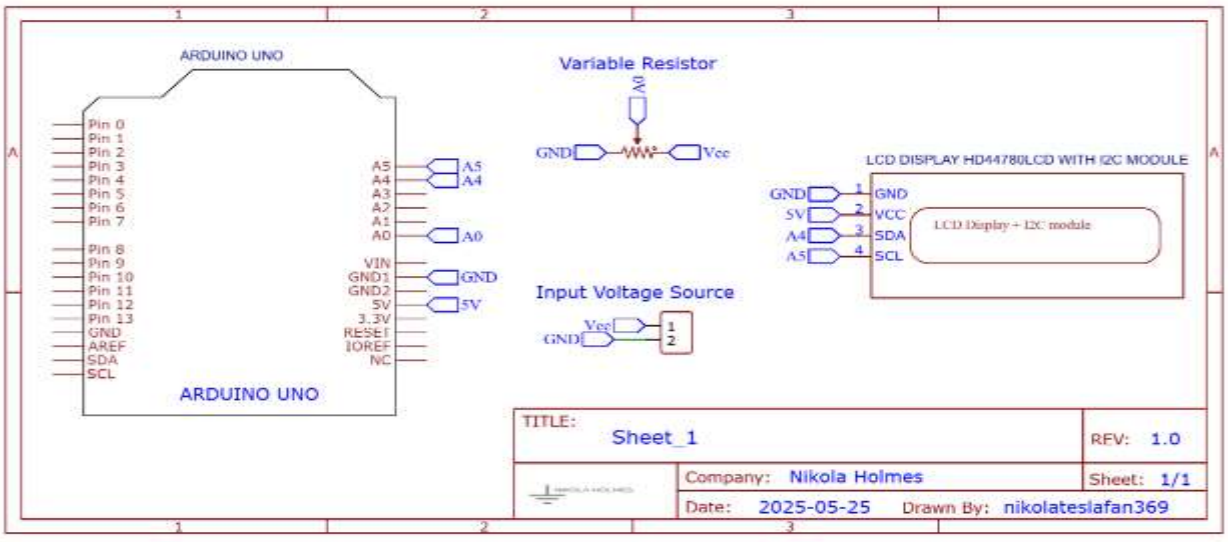

Figure 08.04: Circuit Diagram for Digital Arduino voltmeter

Experimental Setup:

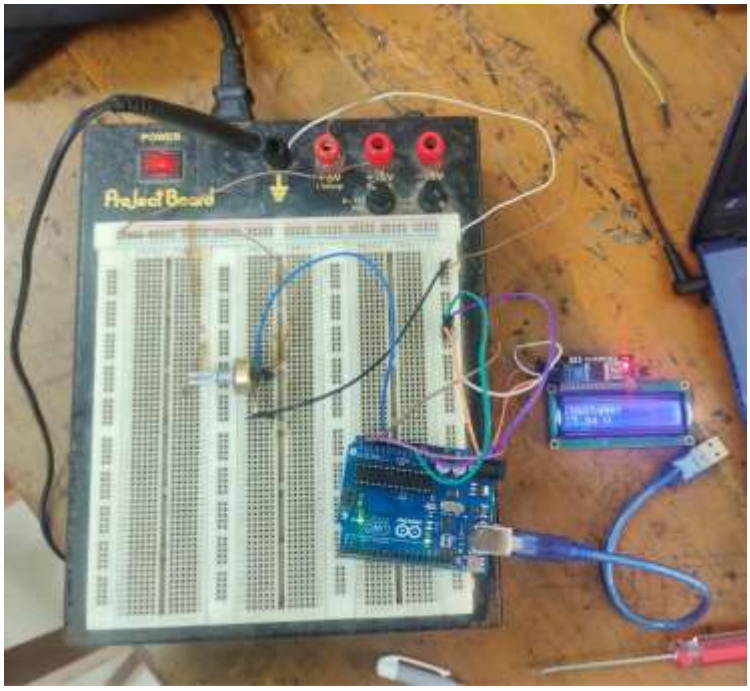

Figure 08.05: Experimental setup of Digital Arduino voltmeter

Code:

```
#include <Wire.h>
#include <LiquidCrystal_I2C.h>
LiquidCrystal_I2C lcd(0x27, 16, 2);
int analog_value = 0;
float voltage = 0.0;
float actual_voltage = 0.0;
void setup() {
  lcd.begin(16, 2);
  lcd.backlight();
  lcd.setCursor(0, 0);
  lcd.print("Digital Voltmeter");
  delay(2000);
  lcd.clear();
}
```

```
void loop() {
  analog_value = analogRead(A0);
  voltage=(analog_value*5.0)/1024.0;
  actual_voltage=voltage* 10.9555555;
  lcd.setCursor(0, 0);
  lcd.print("Voltage:  ");
  lcd.setCursor(9,0);
  lcd.print(actual_voltage, 2);
  lcd.print("V");
  delay(500);
}
```

Experimental Procedure:

- Connect a **voltage divider circuit** using two resistors to scale down the input voltage (up to 50V) to a safe 0–5V range for the Arduino analog pin.
- Connect the **output of the voltage divider** to **analog pin A0** of the Arduino Uno.
- Interface the **16x2 LCD display** with the Arduino using the **I2C module:**
    - Connect **SDA (I2C data line)** to **A4** of the Arduino.
    - Connect **SCL (I2C clock line)** to **A5** of the Arduino.
    - Connect **VCC** of the I2C module to **5V** and **GND** to **GND** of the Arduino.

- Write and upload the Arduino sketch that:
  - Reads analog voltage from A0.
  - Converts the raw analog value to a scaled voltage using the formula: V=(analog value×5\ 1024) ×scaling factor
  - Displays the calculated voltage on the LCD with two decimal precision.
- Power the Arduino using a USB cable or a suitable external power source (7–12V input).
- Apply a known DC voltage to the **input of the voltage divider** and observe the real-time voltage reading displayed on the LCD.
- Vary the input voltage and verify the accuracy and responsiveness of the voltmeter readings displayed.

Hands-on Learning Projects for Arduino based Voltmeter

Task No.: 01

Task Name: save the voltage values in memory and show it in display

Objective: to know the function of EEPROM in case of voltmeter

Code:

- To save the voltage values in EEPROM

```
#include <EEPROM.h>
void setup() {
  Serial.begin(9600);
  delay(1000);
  Serial.println("Enter index (0-199) to read voltage from EEPROM:");
}
void loop() {
  if (Serial.available() > 0) {
    int index = Serial.parseInt();
    while (Serial.available() > 0) {
      Serial.read();
    }
    if (index >= 0 && index < 200) {
      int address = index * 4;
      float savedVoltage;
      EEPROM.get(address, savedVoltage);
      if (isnan(savedVoltage)) {
        Serial.println("Warning: Read value is NaN (not a number). EEPROM data may be invalid.");
      } else {
        Serial.print("Voltage at index ");
        Serial.print(index);
        Serial.print(" (EEPROM address ");
        Serial.print(address);
        Serial.print("): ");
        Serial.println(savedVoltage, 4);
      }
    } else {
      Serial.println("Invalid index! Please enter a value between 0 and 199.");
    }
    Serial.println("\nEnter another index (0-199):");
  }
}
```

- To show the all the saved index

```
#include <Wire.h>
#include <LiquidCrystal_I2C.h>
#include <EEPROM.h>
LiquidCrystal_I2C lcd(0x27, 16, 2);
int analog_value = 0;
float voltage = 0.0;
  lcd.print(" V");
  Serial.println("==== EEPROM Startup Info ====");
  Serial.print("Last saved index: ");
  Serial.println(currentIndex);
  Serial.print("Last saved voltage: ");
  Serial.println(lastSavedVoltage, 2);
```

```
float actual_voltage = 0.0;
const int maxReadings = 200;
const unsigned long writeInterval
= 5000;
unsigned long lastWriteTime = 0;
int currentIndex = 0;
const int indexAddress = 800;
void EEPROMWriteFloat(int address,
float value) {
  byte           *data           =
(byte*)(void*)&value;
  for (int i = 0; i < 4; i++) {
    EEPROM.write(address   +   i,
data[i]);
  }
}
float EEPROMReadFloat(int address)
{
  float value = 0.0;
  byte           *data           =
(byte*)(void*)&value;
  for (int i = 0; i < 4; i++) {
    data[i]  =  EEPROM.read(address
+ i);
  }
  return value;
}
void setup() {
  lcd.begin(16, 2);
  lcd.backlight();
  Serial.begin(9600);
  currentIndex                    =
EEPROM.read(indexAddress);
  if (currentIndex >= maxReadings)
currentIndex = 0;
  float     lastSavedVoltage     =
EEPROMReadFloat(currentIndex * 4);
  lcd.setCursor(0, 0);
  lcd.print("Last Voltage:");
  lcd.setCursor(0, 1);
  lcd.print(lastSavedVoltage, 2);
Serial.println("============================");
  delay(3000);
  lcd.clear();
}
void loop() {
  analog_value = analogRead(A0);
  voltage = (analog_value * 5.0) / 1024.0;
  actual_voltage = voltage * 11.0;
  lcd.setCursor(0, 0);
  lcd.print("Voltage:       ");
  lcd.setCursor(9, 0);
  lcd.print(actual_voltage, 2);
  lcd.print("V");
  Serial.print("Current Voltage: ");
  Serial.println(actual_voltage, 2);
  if (millis() - lastWriteTime >= writeInterval)
{
    int address = currentIndex * 4;
    EEPROMWriteFloat(address, actual_voltage);
    EEPROM.write(indexAddress, currentIndex);
    Serial.print("Saved at index ");
    Serial.print(currentIndex);
    Serial.print(" → EEPROM address ");
    Serial.print(address);
    Serial.print(" : ");
    Serial.println(actual_voltage, 2);
    currentIndex++;
    if (currentIndex >= maxReadings) currentIndex
= 0;
    lastWriteTime = millis();
  }
  delay(500);
}
```

- To get the input from user and show that specific saved index value

```
#include <EEPROM.h>
void setup() {
  Serial.begin(9600);
  delay(1000);
  Serial.println("Enter index (0-199)
to read voltage from EEPROM:");
}
void loop() {
  if (Serial.available() > 0) {
    int index = Serial.parseInt();
    while (Serial.available() > 0) {
      if (isnan(savedVoltage)) {
        Serial.println("Warning: Read value is
NaN  (not  a  number).  EEPROM  data  may  be
invalid.");
      } else {
        Serial.print("Voltage at index ");
        Serial.print(index);
        Serial.print(" (EEPROM address ");
        Serial.print(address);
        Serial.print("): ");
        Serial.println(savedVoltage, 4);
```

```
      Serial.read();
    }
    if (index >= 0 && index < 200) {
      int address = index * 4;
      float savedVoltage;
      EEPROM.get(address,
savedVoltage);
      }
    } else {
      Serial.println("Invalid  index!  Please
enter a value between 0 and 199.");
    }
    Serial.println("\nEnter another index (0-
199):");
  }
}
```

Task No.: 02

Task Name: Reading the value of Designed Voltmeter in mobile app using Bluetooth module

Objective: to adapt the function ability of Bluetooth module with Arduino based voltmeter

Code:

```
#include <Wire.h>
#include <LiquidCrystal_I2C.h>
#include <SoftwareSerial.h>
LiquidCrystal_I2C lcd(0x27, 16,
2);
SoftwareSerial BTSerial(10, 11);
int analog_value = 0;
float voltage = 0.0;
float actual_voltage = 0.0;
void setup() {
  Serial.begin(9600);
  BTSerial.begin(9600);
  lcd.begin(16, 2);
  lcd.backlight();
  lcd.setCursor(0, 0);
  lcd.print("Digital Voltmeter");
  delay(2000);
  lcd.clear();
}
```

```
void loop() {
  analog_value = analogRead(A0);
  voltage = (analog_value * 5.0) / 1024.0;
  actual_voltage = voltage * 10.9555555;
  lcd.setCursor(0, 0);
  lcd.print("Voltage:       ");
  lcd.setCursor(9, 0);
  lcd.print(actual_voltage, 2);
  lcd.print("V");
  BTSerial.print("Voltage: ");
  BTSerial.print(actual_voltage, 2);
  BTSerial.println(" V");
  Serial.print("Voltage: ");
  Serial.print(actual_voltage, 2);
  Serial.println(" V");
  delay(500);
}
```

Experimental Result:

- The Arduino-based **Digital Voltmeter** was successfully implemented using a voltage divider circuit and an analog input pin (A0) to measure DC voltages up to **50V.**

  - The analog readings were accurately converted to voltage values using the known scaling factor derived from the resistor ratio.
  - The measured voltage was displayed on a **16×2 I2C LCD** in real time with two-decimal precision.
  - The readings were found to be consistent and matched the values from a standard digital multimeter within an acceptable error margin.

- In the **first additional task**, voltage readings were stored periodically in the **Arduino's internal EEPROM:**

- Each new reading was written to a sequential memory address, allowing storage of multiple voltage values.
- Upon user request (through a code trigger or button input), the saved values were retrieved and displayed along with their **EEPROM memory index**, confirming successful non-volatile data logging.
- The data remained intact even after a power reset, demonstrating EEPROM persistence.

- In the **second additional task**, an **HC-05 Bluetooth module** was integrated for wireless transmission:

  - Real-time voltage readings were sent over UART serial communication to a mobile device via a **Bluetooth terminal app.**
  - The smartphone successfully received and displayed the transmitted voltage values continuously, with no noticeable lag or data corruption.
  - This setup enabled wireless voltage monitoring, which can be useful in remote or inaccessible measurement scenarios.

Discussions:

The Arduino-based digital voltmeter worked successfully and could measure DC voltages up to 50V using a simple voltage divider and the Arduino's ADC. The readings shown on the LCD were quite accurate and closely matched those from a standard digital multimeter,with only small errors caused by resistor tolerances,despite many faults and difficulties.

Justification of CO2 and PO(j):

Through this experiment, we were able to use a voltage divider circuit to safely measure higher voltages, apply the Arduino Uno and its ADC to accurately measure and convert voltage values and gain practical experience with LCD display, EEPROM and Bluetooth module interfacing for data display, storage and transmission. In addition by designing and debugging a functional embedded system, we became familiar with real-world electrical measurement systems.

Conclusion:

In this lab,we successfully designed and implemented a digital voltmeter using Arduino Uno. The circuit was able to measure DC voltages up to 50V by applying a voltage divider and converting the scaled values through the Arduino's ADC.

Reference: https://www.electronicshub.org/digital-arduino-voltmeter.