REPORTS ON MECHATRONICS SYSTEM INTEGRATION

REPORT 3A

SERIAL COMMUNICATION

SECTION 1, SEMESTER 2, 23/24

Date of Experiment:

Date of Submission: ***20/3/2024***

Team Members:

1. IKMAL FIKRI BIN KHAIRUL KHUBAIDILLAH (2218723)
1. MUHAMMAD IRFAN BIN ROSDIN (2214417)
2. ASYRAAF AIMAN MD HASSAN (2010705)
3. MUHAMMAD IKHWAN BIN MUHAMMAD FAUZI (22188845)

Table of Content

EXPERIMENT 3 : SERIAL COMMUNICATION

PART A : POTENTIOMETER AND LED

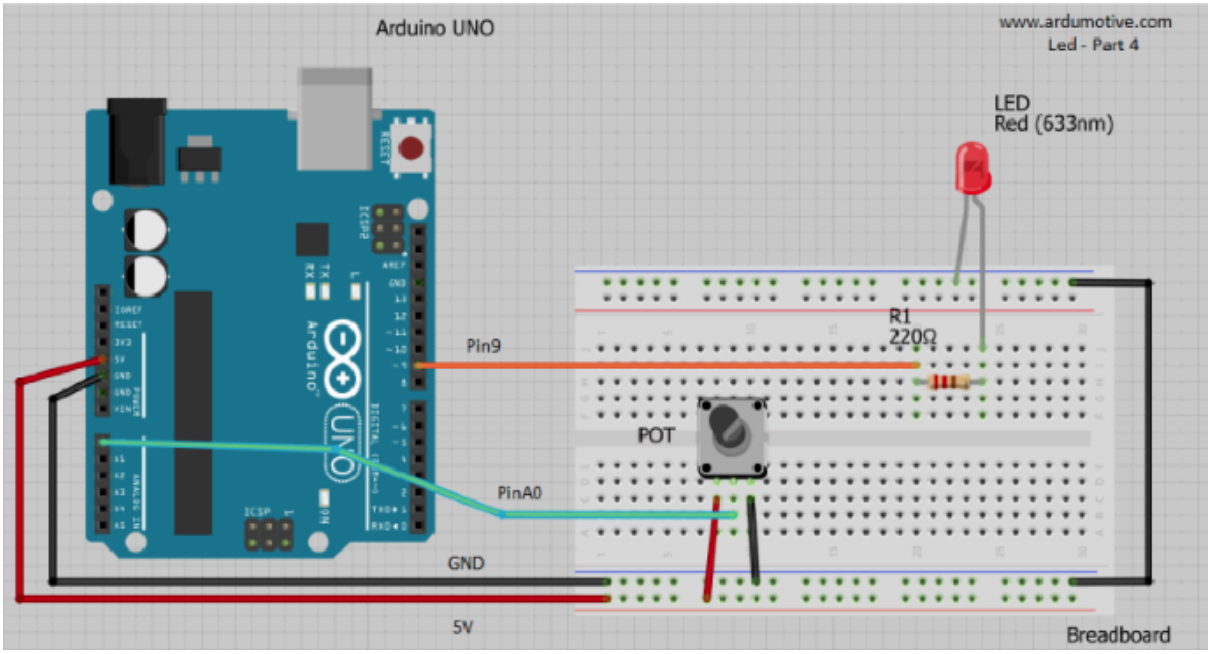
Introduction

The objective of this experiment is to exchange data between a computer and a microcontroller by using the potentiometer and LED . By sending the potentiometer readings from Arduino to a Python script through a USB connection , we will be able to understand how it exchanges data with the code that uploads in the microcontroller .

Materials and Equipment

Arduino Board	X 1
Potentiometer	X 1
Jumper Wires	X 5
LED	X 1
Breadboard	X 1

Experimental Setup

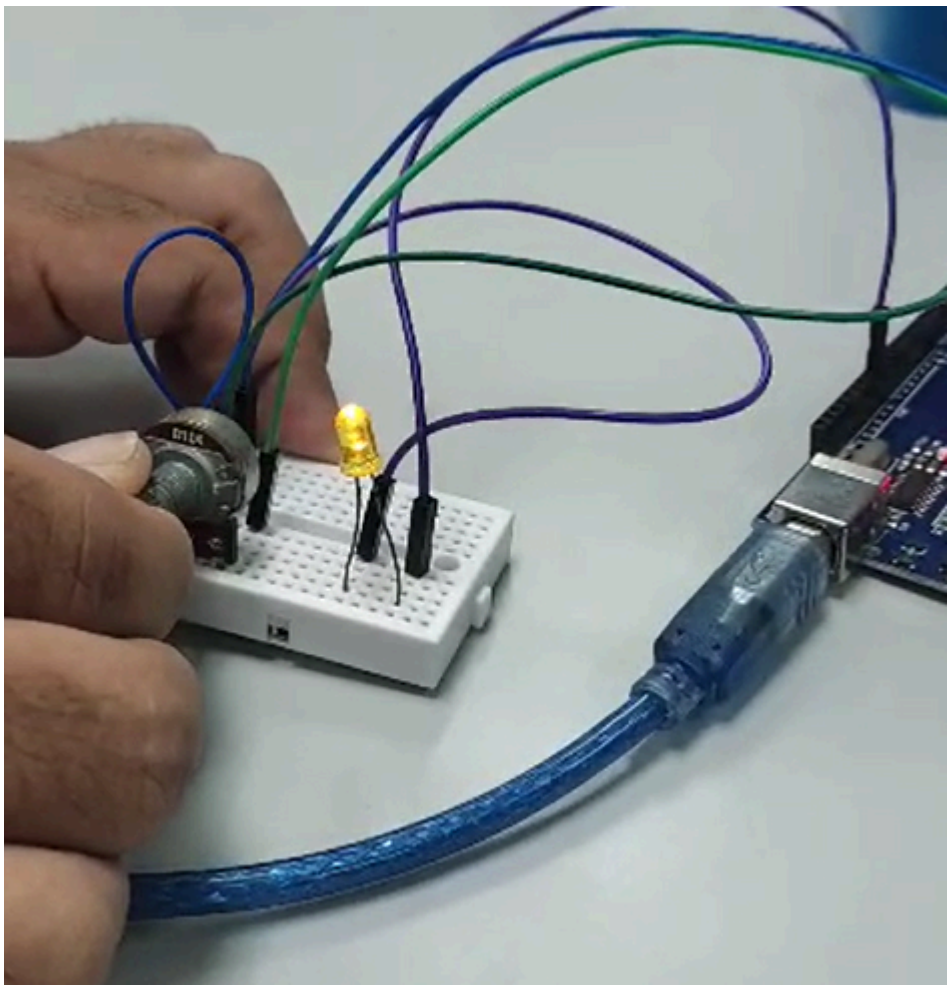


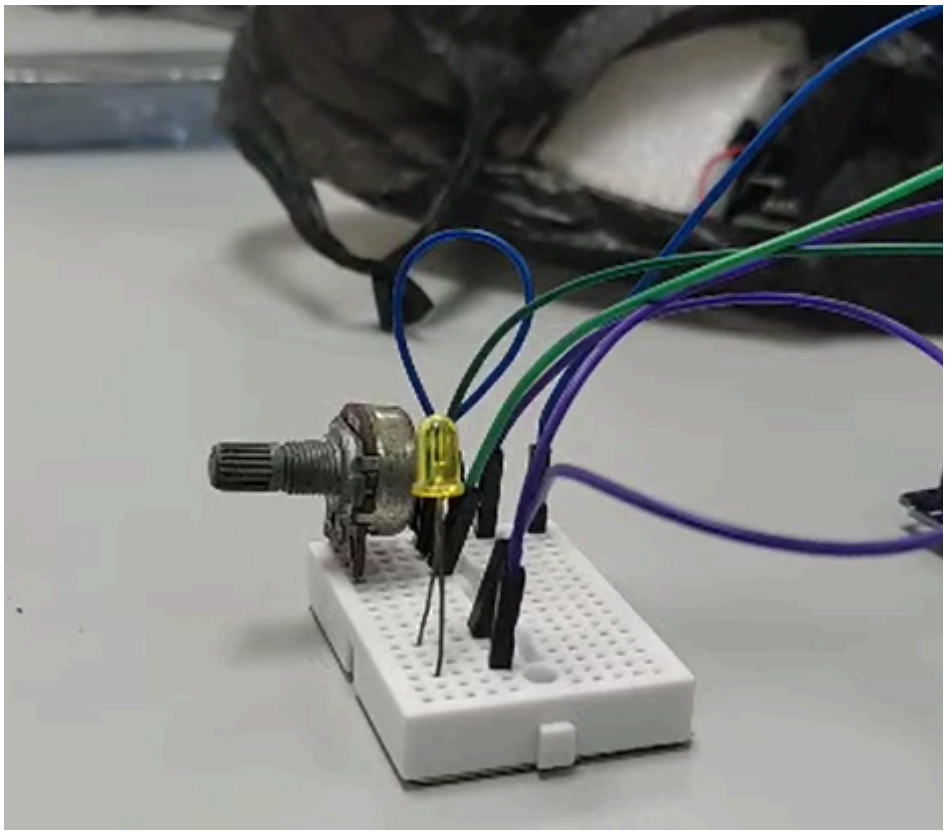
Methodology/Procedures

1. Connect the Arduino to your computer via a USB cable.
2. Power on the Arduino (upload the sketch to your Arduino using the Arduino IDE)
3. Run the Python script on your computer.
4. As you turn the potentiometer knob, you should see the potentiometer readings displayed in the Python terminal.
5. You can use these readings for various experiments, data logging, or control applications, depending on your project requirements.
6. Open the Serial Plotter: In the Arduino IDE, go to "Tools" -> "Serial Plotter."
7. Select the Correct COM Port: In the Serial Plotter, select the correct COM port to which your Arduino is connected.
8. Set the Baud Rate: Ensure that the baud rate in the Serial Plotter matches the one set in your Arduino code (e.g., 9600).
9. Read Real-Time Data: As you turn the potentiometer knob, the Serial Plotter will display the potentiometer readings in real-time, creating a graphical representation of the data. You can see how the values change as you adjust the potentiometer.
10. Customize the Plotter: You can customize the Serial Plotter by adjusting settings such as the graph range, labels, and colors.

Results

The results of this experiment were acquired by constructing a basic circuit using a potentiometer to adjust the brightness of an LED. The potentiometer reads the value, which is then used to compute the LED's brightness. The brightness is calculated by dividing the potentiometer value by four. This value is then passed as an input to the function that controls the LED's brightness. To debug the potentiometer, the value was printed to the serial monitor. Overall, the experiment successfully shows how to utilise a potentiometer to regulate the brightness of an LED.

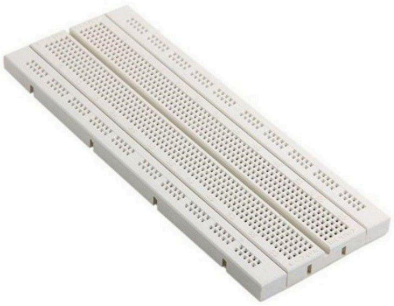




Discussion

Hardware

1) Breadboard



Breadboard is used to connect components with arduino through wire connection.

2) Arduino Mega 2560

Microcontroller used for our experiment is Arduino Mega 2560.

3) Male to male jumper wires

Male to male jumper wires are used to connect all of the components on the breadboard with Arduino Mega.

4) Potentiometer

a manually adjustable variable resistor with 3 terminals

5) LED

a semiconductor device that emits light when current flows through it

6) 220 Ohm resistor

used to resist the flow of electricity in a circuit. 220 Ohm rated.

Electrical

In this experiment, the electrical setup consisted of a potentiometer interfaced with an Arduino Mega to establish a functional circuit. The potentiometer served as a variable resistor, allowing for the adjustment of resistance by turning its knob. This variation in resistance was crucial as it facilitated the generation of analog voltage signals corresponding to the position of the potentiometer knob.

The potentiometer was connected to the Arduino board in a simple configuration: one leg was linked to the 5V power supply, another to the ground (GND), and the middle leg (wiper) to an analog input pin, in this case, A0. This arrangement created a voltage divider circuit, with the analog voltage at the middle leg varying between 0V and 5V based on the potentiometer's position.

The Arduino utilised its analog-to-digital converter (ADC) to measure this analog voltage. The analogRead() function in the Arduino code sampled the voltage at the analog pin and converted it into a digital value ranging from 0 to 1023, representing the voltage level. Serial communication was employed to transmit these digital values from the Arduino to the Python script running on the computer. By establishing a serial connection via USB, the Arduino sent the potentiometer readings as a stream of data, which was then decoded and displayed by the Python script.

It's worth noting the importance of ensuring compatibility between the baud rates set in both the Arduino code and the Python script. The baud rate is 9600. A mismatch in baud rates can lead to communication errors or data corruption.

Software

Arduino code

```
#define Led_pin 9                    #define Pot_pin A0
```

```
void setup()
{
  pinMode(Led_pin, OUTPUT);
  Serial.begin(9600);
}

void loop()
{
  int potVal = analogRead(Pot_pin);
  int brightness = potVal/4;
  analogWrite (Led_pin, brightness);


Serial.println(potVal);

  delay(100);
}
```

Python code

```
ser = serial.Serial('COMX', 9600)
while True:
 pot_value = ser.readline().decode().strip()
 print("Potentiometer Value:", pot_value)
except KeyboardInterrupt:
 ser.close()
 print("Serial connection closed.")
```

This Arduino and Python code combination demonstrates a simple communication setup between an Arduino board and a computer. The Arduino code reads the value from a potentiometer connected to analog pin A0, maps this value to a brightness level, and then sends this value over serial communication to the computer. The Python code, running on the computer, reads the serial data from the Arduino, decodes it, and prints the potentiometer value.

1. Arduino Code:
    a) The `analogRead(Pot_pin)` function reads the analog value from the potentiometer connected to pin A0.
    b) The `analogWrite(Led_pin, brightness)` function writes a PWM (Pulse Width Modulation) signal to pin 9 to control the brightness of an LED connected to that pin.
    c) The `Serial.begin(9600)` function initializes serial communication at a baud rate of 9600.

2. Python Code:

a) The `serial.Serial('COMX', 9600)` line initializes a serial connection with the Arduino. Replace ``'COMX'`` with the correct COM port.
b) The `ser.readline().decode().strip()` line reads a line of serial data from the Arduino, decodes it into a string, and removes any leading/trailing whitespace.
c) The `print("Potentiometer Value:", pot_value)` line prints the potentiometer value received from the Arduino.

3. Communication:

a) The Arduino continuously reads the potentiometer value, maps it to a brightness level, and sends it over serial.
b) The Python script reads the serial data and prints the potentiometer value.

4. Error Handling:

a) Both codes include `except KeyboardInterrupt` blocks to handle the case when the user interrupts the program (e.g., by pressing Ctrl+C). This block closes the serial connection and prints a message.

5. Improvements:

a) Error checking and handling could be improved in both codes. For example, checking if the serial connection is successful or handling unexpected data formats.
b) Adding a graphical interface to the Python code could make it more user-friendly.

Question

a) To present potentiometer readings graphically in your Python script, you may enhance your code by introducing the capability to generate and showcase a graph. This graphical visualization can deliver a more intuitive and informative perspective for data interpretation. Be sure to showcase the steps involved in your work (Hint: use matplotlib in your Python script).

Python Code

```
import matplotlib.pyplot as plt
import serial
import time
# Open serial port
ser = serial.Serial('COM3', 9600)  # Update 'COM3' with your Arduino's port
time.sleep(2)  # Wait for serial connection to establish

# Initialize empty lists to store data
x_data = []
y_data = []

# Set up the plot
plt.ion()  # Turn on interactive mode
fig, ax = plt.subplots()
line, = ax.plot(x_data, y_data)
ax.set_xlabel('Time (s)')
ax.set_ylabel('Potentiometer Reading')

# Main loop
try:
    while True:
        # Read potentiometer value from Arduino
        data = ser.readline().decode().strip()
        if data:
            pot_val = int(data)
            print("Potentiometer Reading:", pot_val)

            # Append data to lists
            x_data.append(time.time())  # Use time as x-axis
            y_data.append(pot_val)

            # Update plot
            line.set_xdata(x_data)
            line.set_ydata(y_data)
            ax.relim()
            ax.autoscale_view()
            fig.canvas.draw()
            fig.canvas.flush_events()

except KeyboardInterrupt:
    ser.close()
    print("Serial connection closed")
```

This Python code displays real-time data visualization with matplotlib'. By reading potentiometer values from an Arduino over serial transmission, Python demonstrates its ability for interactive data analysis. The code updates a plot dynamically, resulting in an understandable graphical representation of the potentiometer readings. This strategy is beneficial not only for monitoring sensor data, but also for teaching people how to interpret and apply real-time data visualization techniques. Furthermore, the code demonstrates

Python's easy connection with external hardware, transforming it into a powerful tool for data processing and visualization.

Conclusion

In conclusion, the given code demonstrates a basic but effective configuration using a potentiometer and a light-emitting diode (LED) controlled by an Arduino board. The Arduino reads the analogue input from the potentiometer and maps it to control the brightness of the LED. This analogue value is then sent via serial transmission to a Python script running on a computer. The Python software accepts the data and outputs the potentiometer value, demonstrating bidirectional connection between the Arduino and the PC.

This configuration serves as a starting point for more complicated applications that incorporate sensor inputs and output control. It showcases Arduino's versatility in interacting with external components, as well as Python's flexibility in processing and presenting data.