Heavens' Light is Our Guide

Rajshahi University of Engineering & Technology

Department of Computer Science & Engineering

CSE 3106
Computer Interfacing & Embedded Systems

Lab 3: Potentiometer, LDR Sensor, Temperature Sensor

Submitted by

Kaif Ahmed Khan

Roll: 2103163

Submitted to

Nasif Osman Khansur
Lecturer

May 27, 2025

Contents

List of Listings

List of Figures

1 LED Brightness Control Using Potentiometer

1.1 Introduction

A potentiometer is a variable resistor commonly used to provide adjustable voltage in electronic circuits. In this experiment, it is used as an analog input device that allows the user to control the brightness of an LED. By rotating the potentiometer's knob, the voltage at its output (wiper) changes smoothly between 0V and the supply voltage.

This varying voltage is read by the microcontroller through one of its analog input pins. The microcontroller then maps the input value to a range suitable for PWM (Pulse Width Modulation) output, which is used to control the LED. A higher input voltage results in a higher PWM duty cycle, making the LED appear brighter, while a lower voltage dims it. This setup demonstrates how a simple user interface element like a potentiometer can be used to control an output device in real-time.

1.2 Procedure

1. Gathered an STM32F103C6 blue pill board, potentiometer, LED, 220Ω resistor, breadboard, and jumper wires.

2. Placed the potentiometer on the breadboard. Connected the middle pin to PA0 (analog input), one side to 3.3V, and the other to GND.

3. Connected the LED's anode to PB0 (PWM output) through a 220Ω resistor, and the cathode to GND.

4. Verified all connections were secure and correct.

5. Uploaded the code via PlatformIO. The LED brightness responded to the potentiometer's wiper position.

1.3 Diagram

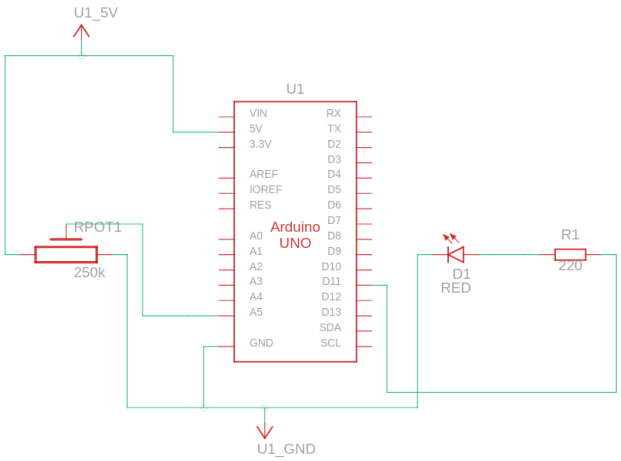

Figure 1: *Schematic diagram of the circuit using Arduino in TinkerCAD*

1.4 Source Code

Listing 1: *Controlling LED brightness using potentiometer*

```
#include <Arduino.h>
#define POT_PIN PA0
#define LED PB0
void setup() {
  pinMode(POT_PIN, INPUT);
  pinMode(LED, OUTPUT);
}

void loop() {
  int value = analogRead(POT_PIN);
  value = map(value, 0, 1023, 0, 255);
  analogWrite(LED, value);
  delay(500);
}
```

1.5 Discussion

In this experiment, I successfully used a potentiometer to control the brightness of an LED through PWM using an STM32 microcontroller and the Arduino framework. The code read the analog voltage from the potentiometer, mapped it to a 0–255 range, and used analogWrite() to adjust the LED's brightness accordingly. As expected, turning the potentiometer smoothly increased or decreased the brightness, confirming that the analog input was correctly processed and translated into a PWM output.

The results matched the intended behavior, showing a clear and consistent change in LED brightness relative to the potentiometer position. This demonstrated the effective use of analog-to-digital conversion and pulse-width modulation in a microcontroller environment.

Through this experiment, I learned how to read analog values from potentiometer, map them for output control, and use PWM to vary LED intensity. It also reinforced my understanding of basic circuit connections and how software and hardware interact to produce real-time responses.

2 LDR Sensor

2.1 Introduction

A Light Dependent Resistor (LDR) is a type of photoresistor whose resistance changes based on the intensity of light falling on it. In this experiment, the LDR is used as an analog input sensor that allows ambient light levels to automatically control the brightness of an LED. As the surrounding light increases or decreases, the voltage across the LDR changes accordingly.

This changing voltage is read by the microcontroller through one of its analog input pins. The input value is then mapped to a range suitable for Pulse Width Modulation (PWM), which is used to adjust the LED brightness. In low light, the LED becomes brighter, and in bright light, it dims. This setup demonstrates how sensors like LDRs can be used to creat e responsive and automatic lighting systems based on environmental conditions.

2.2 Procedure

1. Gathered an STM32F103C6 blue pill board, LDR, LED, 220Ω resistor, breadboard, and jumper wires.

2. Placed the LDR on the breadboard. Connected one side to 3.3V and the other side to PB1 (analog input) and also to GND through a fixed resistor to form a voltage divider.

3. Connected the LED's anode to PB0 (PWM output) through a 220Ω resistor, and the cathode to GND.

4. Verified all connections were secure and correct.

5. Uploaded the code via PlatformIO. The LED brightness changed automatically in response to ambient light levels detected by the LDR.

2.3 Diagram

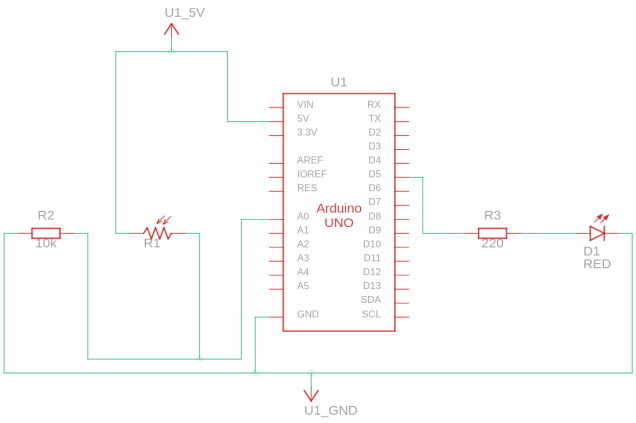

Figure 2: *Schematic diagram of the circuit using Arduino in TinkerCAD*

2.4 Source Code

Listing 2: *LED control using LDR sensor*

```
#include <Arduino.h>
#define LED PB0
#define LDR PB1
void setup() {
  pinMode(LED, OUTPUT);
  pinMode(LDR, INPUT);
}

void loop() {
  int value = analogRead(LDR);
  value = map(value, 0, 1023, 255, 0);
  analogWrite(LED, value);
  delay(500);
}
```

2.5 Discussion

In this experiment, I successfully used an LDR sensor to automatically control the brightness of an LED through PWM using an STM32 microcontroller and the Arduino framework. The code read the analog voltage from the LDR, mapped it inversely to a 0–255 range, and used analogWrite() to adjust the LED's brightness accordingly. As expected, the LED became brighter in low light and dimmer in bright light , confirming that the analog input was correctly processed and translated into a PWM output.

The results matched the intended behavior, showing a clear and consistent change in LED brightness based on ambient light levels detected by the LDR. This demonstrated the practical application of analog-to-digital conversion and pulse-width modulation in responsive lighting control.

Through this experiment, I learned how to read and interpret sensor data, apply mapping for output control, and use PWM to adjust LED intensity dynamically. It also reinforced my understanding of sensor integration and the interaction between hardware and software to create automated systems.

3 Temperature Sensor

3.1 Introduction

An RGB LED combines red, green, and blue light-emitting diodes into a single package, allowing various colors to be displayed based on the combination of the three. In this experiment, a common anode RGB LED is used as a visual indicator to represent different temperature ranges detected by a temperature sensor. The anode is connected to a positive voltage, and individual colors are controlled by pulling their respective cathodes low.

The analog signal from the temperature sensor is read by the microcontroller and compared against predefined threshold values. Depending on the temperature reading, the RGB LED lights up in blue, green, or red to indicate cold, normal, or high temperature, respectively. The logic is designed for common anode configuration, where writing LOW to a pin turns that LED color on. This setup provides a simple and effective way to visualize sensor data using colored light, demonstrating how analog inputs and digital outputs can be used together in real-time embedded systems.

3.2 Procedure

1. Gathered an STM32F103C6 Blue Pill board, common anode RGB LED, temperature sensor, three 220Ω resistors, breadboard, and jumper wires.

2. Connected the common anode pin of the RGB LED to 3.3V. Connected the red, green, and blue cathode pins to PB0, PB4, and PB3 respectively, each through a 220Ω resistor.

3. Connected the temperature sensor's output to PB1 (analog input), VCC to 3.3V, and GND to ground.

4. Verified all wiring for correctness and secure connections on the breadboard.

5. Uploaded the code via PlatformIO. The RGB LED lit up in different colors (blue, green, or red) based on the temperature sensor's reading, indicating low, normal, or high temperature.

3.3 Diagram

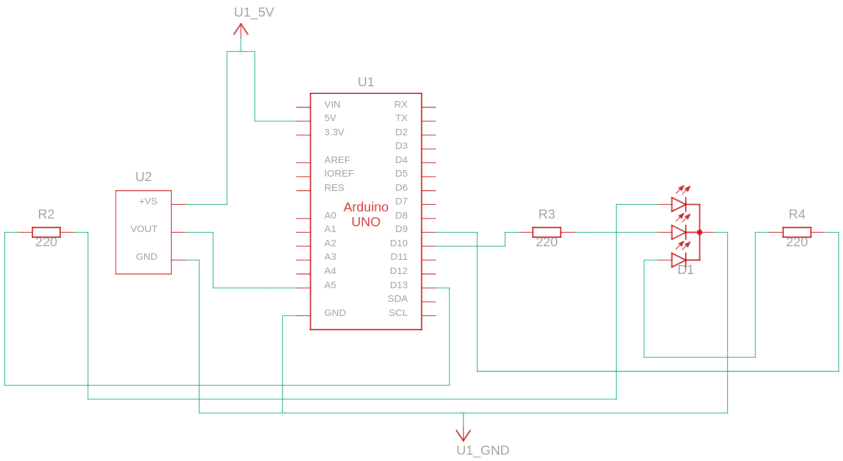

Figure 3: *Schematic diagram for Temperature sensor controlled RGB LED circuit using common cathode configuration in TinkerCAD*

3.4 Source Code

Listing 3: *Temperature sensor*

```
#include <Arduino.h>
#define RED PB0
#define BLUE PB3
#define GREEN PB4
#define TMP PB1
#define LOWER_BOUND 139
#define UPPER_BOUND 147
void setup() {
  pinMode(RED, OUTPUT);
  pinMode(BLUE, OUTPUT);
  pinMode(GREEN, OUTPUT);
  pinMode(TMP, INPUT);
}

void loop() {
  int value = analogRead(TMP);
  // common anode configuration
  if(value < LOWER_BOUND){
    // BLUE LED on
    digitalWrite(BLUE, LOW);
    digitalWrite(RED, HIGH);
    digitalWrite(GREEN, HIGH);
  }
  else if(value > UPPER_BOUND){
    // RED LED on
    digitalWrite(BLUE, HIGH);
    digitalWrite(RED, LOW);
```

```
28      digitalWrite(GREEN, HIGH);
29    }
30    else {
31      // GREEN LED on
32      digitalWrite(BLUE, HIGH);
33      digitalWrite(RED, HIGH);
34      digitalWrite(GREEN, LOW);
35    }
36  }
```

3.5 Discussion

In this experiment, I successfully used a temperature sensor and a common anode RGB LED to visually indicate temperature ranges using an STM32 microcontroller and the Arduino framework. The analog temperature readings were processed and compared against predefined lower and upper thresholds. Based on these readings, the microcontroller lit up the RGB LED in blue for low temperature, green for normal, and red for high temperature by controlling the cathode pins accordingly.

The behavior matched the expected outcome, with each color responding appropriately to changes in the temperature sensor's output. Since a common anode RGB LED was used, the LED colors were controlled by writing LOW to turn them on and HIGH to turn them off, which was correctly implemented in the code.

This experiment helped me understand how to interpret analog sensor data, apply conditional logic in embedded systems, and control multi-color LEDs. It also reinforced the importance of knowing hardware configurations like common anode vs. common cathode when writing control logic for LEDs.