“ESP32-Based 3-State MQTT System for Publishing and Controlling Multiple Components”

A dynamic 3 state switch controlling different components, one at a time. The data of these sensors are then sent over through the MQTT protocol to **your-broker-domain.com**. The site hosts multiple control-based components that can be used to control the functionality of the actual hardware components connected to the ESP-32. Since ESP-32 boasts an on-board Wi-Fi, it facilitates communication with MQTT brokers.

Project Overview

This is a 3-state smart switch that responds to MQTT messages and changes behavior accordingly:
- State 1: LDR sensor data transfer to your-broker-domain.com
- State 2: Servo motor control
- State 3: RGB Led visual feedback

Tutorial_MQTT/sensors: Data representation (RAW) and transfer(JSON) functionality:

- ESP-32 connects to a secure MQTT broker using Wi-Fi and interprets the JSON payload-based data to change the data on the site in real time considering if the value has changed on the sensor/module itself.
  There are different topics assigned in MQTT for different kinds of purpose: -
  Data transfer is handled by **Tutorial_MQTT/sensors/data (JSON)**
- Visual feedback of changing data on the MQTT itself is handled by **Tutorial_MQTT/sensors/data** (a double check mechanism to ensure data reaches MQTT Explorer properly).

Tutorial_MQTT/control: Control based data sending (to ESP-32) and receiving(your-broker-domain.com):

- **Tutorial_MQTT/control/servo:** It hosts servo control functionality using a JSON based payload format as shown below. If the payload is sent with a **different angle** than the currently set value, using the **PUBLISH** button (ensuring **JSON** format is selected) the angle of the servo changes accordingly in real time. HowevMQTT topic doesn’t connect to **your-broker-domain.com** but is a checking mechanism to ensure the servo responds to the control-based data sent over from the MQTT Explorer. It’s like ensuring the sender **(MQTT)** itself is

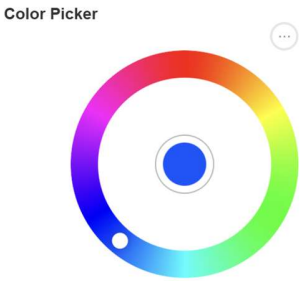
not giving any issues between **your-broker-domain.com** and **ESP-32** communication.

```
{
  "angle": 90
}
```

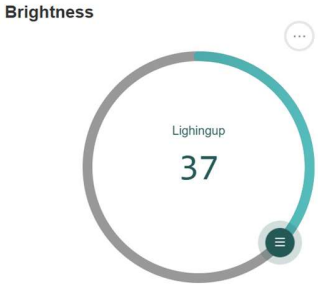
- **Tutorial_MQTT/control/rgb:** It also bears the same functionality as the servo control-based topic mentioned above but for **RGB** specifically. Data can be sent in the form of a JSON payload. However, the difference is that we can send different values for each color (Red, Green, and Blue) unlike in serial monitor where we only get option for Red, Green, Blue, Pink, Cyan, and yellow (since serial monitor is only to test whether the RGB Led is responding to different basic colors). This functionality is beneficial because it gives us an idea about the extent to which the colors on the RGB Led can be set to. This will ensure that RGB led won't fail when the Color picker function of **your-broker-domain.com** will be used.

```
{
  "r": 0,
  "g": 0,
  "b": 0
}
```

- **Tutorial_MQTT/control/color1:** This topic is assigned automatically when this component is configured from the **your-broker-domain.com**'s side into the selected device. For example, this component, i.e., **Color Picker** has to be assigned to a device configured in the site (another topic basically for specific device control), and since we had made 2 more devices along with DATA (servo and RGB), when we would select RGB,a subtopic named color1 will automatically be assigned under **Control** topic. Changing the colors from this component will send particular **R: G: and B:** codes to the MQTT and then to the ESP-32 which will then change the color on the RGB. Only thing to keep is mind is that once the topic is assigned from the site side, the same topic will have to be subscribed also in the code so that data receive functionality can be put into effect.



- **Tutorial_MQTT/control/Brightness1:** This component also has the same setup as the color picker that too in the same device: **RGB**. We have chosen **WHITE** color to represent the effect of brightness components most efficiently. We have set the **R: G:** and **B:** values as same (eg., **0: 0: 0:** , **100: 100: 100:**) to give out the **WHITE** color. When the brightness level is increased the value of same **R: G:** and **B:** values also increases thus making the RGB led glow more and more bright. Values on the panel vary from **0 to 99.**



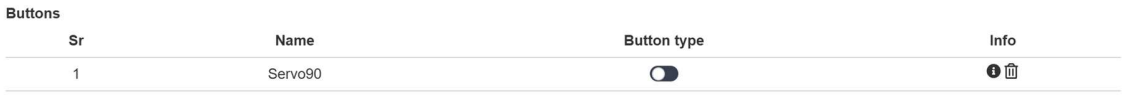
- **Tutorial_MQTT/control/slider1:** This slider is mapped to control the servo motor angle dynamically. By sliding the bar left or right, we adjust the angle of the servo from 0 to 180 degrees. It's an intuitive way to visualize and manually control the servo's movement in real time. Internally, the slider value is directly read and mapped to the servo angle using servo.write(angle). It's best suited for scenarios where precise position control is needed (e.g., robotic arms, camera gimbals, etc.). This component's subtopic also shows up in the **control** topic in MQTT explorer.

- **Tutorial_MQTT/control/text:** This component allows direct text-based input of servo angles. The user can type any value between 0 and 180 (e.g., "90" or "45"), which is then sent as a command to the servo. This method is ideal when we know the exact angle we want to move to, making it more precise than dragging a slide. It's also helpful during debugging or when repeating a fixed

motion pattern in servo applications. This component's subtopic also shows up in the **control** topic in MQTT explorer.

- **Tutorial_MQTT/control/dropdown1:** This dropdown is configured to offer predefined servo angles like "0°", "90°", and "180°". This helps in scenarios where only certain positions are relevant (for example, fixed mechanical stops or gates). It removes the guesswork and provides a consistent interface for controlling servo states with one click. Internally, each dropdown option maps to a specific servo.write() command. This component's subtopic also shows up in the **control** topic in MQTT explorer.

- **Tutorial_MQTT/control/Togglebutton1:** The toggle button is used to switch the servo between two preset positions, like 0° and 180°. It behaves like a power switch — either in one extreme position or the other. This is especially handy in basic applications like opening/closing a flap, triggering a mechanism, or demonstrating binary control (ON/OFF-like) with servos. On toggling, the backend logic checks the current state and sends the corresponding angle to the servo. This component's subtopic also shows up in the **control** topic in MQTT explorer.



Hardware Used

Component	Purpose
ESP-32 (Devkit)	Main controller, WiFi-enabled
LDR	Live continuous data transfer (State 1)

Servo Motor (SG90)	Mechanical control (State 2)
RGB LED (common cathode)	Visual indication (State 3)
External 5V supply	Powers servo independently
Common Grounding	Ensures logic + power sync
Jumper Wires, Breadboard	Prototyping setup
3 State Switch	3 different state controls
OLED Display (SSD1306 128x64) (0.96”)	Information output
Some resistors	To connect to the 3 pins of RGB
Buck Convertor	12V to 5V conversion

Pin Configuration

Function	ESP32 Pin
RGB Red	D14
RGB Green	D12
RGB Blue	D27
Servo PWM	D15
OLED SDA	D21
OLED SCL	D22
Switch Pin1	D32
Switch Pin 2	D33
LDR	D34

MQTT Integration

Broker Details:
- Broker: your-broker-domain.com
- Port: 8883 (Secure TLS)
- Payload Format (JSON) (For data logging and MQTT explorer based component control)

State Breakdown

State 1: LDR Mode
- RGB LED turns on with provided color.
- JSON field: "color": [R, G, B]

State 2: Servo Control
- Servo is attached and moved to given angle.
- JSON field: "angle": <0-180>

State 3: OLED Display
- OLED shows given text.
- JSON field: "message": "Your text here"

Challenges We Faced & How We Tackled Them

Issue	Diagnosis	Solution
Servo jitters or doesn't move	Weak or noisy power line	Used external 5V with proper common ground
OLED didn't display anything	Wrong I2C address or no delay	Verified with I2C scanner, added delay before init
ESP32 not connecting to MQTT	Cert mismatch / WiFi drop	Ensured correct root certificate + reconnection loop
Servo interrupts OLED or RGB	Overlapping GPIO usage	Cleaned up pin mapping and initialized safely
Payload parsing errors	Missing or malformed JSON	Used ArduinoJson with proper deserializeJson() checks
Timing glitch between states	Transitions too abrupt	Cleaned up state logic and used detach() for servo
RGB colors not showing accurately	Wrong PWM logic	Used analogWrite with full range (0-255) and correct wiring

Power Considerations

- ESP32: Needs stable 3.3V, gets via onboard regulator.
- Servo Motor: Pulls up to 500mA—must be powered separately.

- OLED Display: ~20-25mA.
- RGB LED: ~20mA per channel.
- Used common ground across ESP32, OLED, Servo, RGB to prevent floating voltage levels.

Fun Expansions & Future Possibilities

- Add State 4: Buzzer alerts or sound tones
- Add web interface to override MQTT
- Log all state transitions to SD card or cloud
- Sync multiple modules via retained messages
- Add time-based state change (RTC integration)
- Modular expansion with Plug-N-Play breakout boards

DIY Build Steps [Placeholder Section]

To be filled on your-broker-domain.com:
- Project schematic
- Pinout diagrams
- MQTT broker setup steps
- Example payloads
- Arduino sketch walkthrough
- Video demo embed

Code Snippet

```
if (payload["state"] == 1) {
  analogWrite(RED_PIN, payload["color"][0]);
  analogWrite(GREEN_PIN, payload["color"][1]);
  analogWrite(BLUE_PIN, payload["color"][2]);
} else if (payload["state"] == 2) {
  myServo.attach(SERVO_PIN);
  myServo.write(payload["angle"]);
} else if (payload["state"] == 3) {
  display.clearDisplay();
  display.setCursor(0, 0);
  display.println(payload["message"]);
  display.display();
}
```

Hosted at: www. your-broker-domain.com/smart3switch (For reference only)

Complete Code

```
#include <WiFi.h>
#include <PubSubClient.h>
#include <WiFiClientSecure.h>
#include <Adafruit_GFX.h>
#include <Adafruit_SSD1306.h>
#include <ESP32Servo.h>
#include <Wire.h>
#include <ArduinoJson.h>


// Create instances for the servo and display
Servo myServo;
Adafruit_SSD1306 display(128, 64, &Wire, -1);

// --- Pin Definitions ---
#define SW1_PIN 32
#define SW2_PIN 33
#define SERVO_PIN 15
#define LDR_PIN 34
#define RED_PIN 14
#define GREEN_PIN 12
#define BLUE_PIN 27

// --- Global State Variables ---
int currentState = 0; // Start in a neutral state (0)
int lastState = -1;
bool promptPrinted = false;
int currentAngle = 90; // Start servo at a known middle position

const char* ssid = "YOUR_WIFI_SSID";
const char* password = "YOUR_WIFI_PASSWORD";

// Secure MQTT Broker Info (no certificates used)
/*const char* mqtt_server = "your-broker-domain.com"; // e.g.
broker.emqx.io
const int mqtt_port = 8883; // secure MQTT port
const char* mqtt_topic = "Tutorial_MQTT/sensors/data";*/

// Secure MQTT Broker Info (no certificates used)
const char* mqtt_server = "your-broker-domain.com"; // e.g.
broker.emqx.io
const int mqtt_port = 8883; // secure MQTT port
```

```
const char* mqtt_topic = "Tutorial_MQTT/sensors/data(json)";
const char* mqtt_topic1 = "Tutorial_MQTT/sensors/data";
const char* mqtt_topic_rgb_control = "Tutorial_MQTT/control/rgb";
const char* mqtt_topic_servo_control =
"Tutorial_MQTT/control/servo";



// MQTT Authentication
const char* mqtt_user = "oyt";
const char* mqtt_password = "123456789";

WiFiClientSecure secureClient;
PubSubClient client(secureClient);




void reconnect() {
  while (!client.connected()) {
    Serial.print("Attempting MQTT connection...");
    if (client.connect("ESP32ClientSecure", mqtt_user,
mqtt_password)) {
      Serial.println("connected");
      client.subscribe(mqtt_topic_rgb_control);
      client.subscribe(mqtt_topic_servo_control);
      Serial.println("Subscribed to control topics");

    } else {
      Serial.print("failed, rc=");
      Serial.print(client.state());
      Serial.println(" trying again in 5s");
      delay(5000);
    }
  }
}


//-----------------------------------------------------------------
------

void callback(char* topic, byte* payload, unsigned int length) {
  String msg;
  for (int i = 0; i < length; i++) {
    msg += (char)payload[i];
```

```
}

Serial.print("Message arrived [");
Serial.print(topic);
Serial.println("] " + msg);

// --- RGB control topic ---
if (String(topic) == mqtt_topic_rgb_control) {
  StaticJsonDocument<128> doc;
  DeserializationError err = deserializeJson(doc, msg);
  if (!err) {
    int r = doc["r"];
    int g = doc["g"];
    int b = doc["b"];
    setColor(r, g, b);
    Serial.printf("RGB set to: R=%d G=%d B=%d\n", r, g, b);

 // ✅ Add OLED feedback
    display.clearDisplay();
    display.setTextSize(1);
    display.setCursor(0, 10);
    display.println("MQTT RGB Control");

    display.setTextSize(2);
    display.setCursor(0, 30);
    display.printf("R:%d G:%d B:%d", r, g, b);
    display.display();
  } else {
    Serial.println("Invalid RGB JSON received.");
  }
}

// --- Servo control topic ---
else if (String(topic) == mqtt_topic_servo_control) {
  StaticJsonDocument<64> doc;
  DeserializationError err = deserializeJson(doc, msg);
  if (!err) {
    int angle = doc["angle"];
    if (angle >= 0 && angle <= 180) {
      currentAngle = angle;
      myServo.write(currentAngle);
      Serial.printf("Servo angle set to %d\n", currentAngle);

  // ✅ Add OLED feedback
      display.clearDisplay();
```

```
        display.setTextSize(1);
        display.setCursor(0, 10);
        display.println("MQTT Servo Control");

        display.setTextSize(2);
        display.setCursor(20, 30);
        display.printf("Angle: %d%c", angle, 248);  // 248 is the
degree symbol
        display.display();
      }
    } else {
      Serial.println("Invalid Servo JSON received.");
    }
  }
}


//-----------------------------------------------------------------
------


void setup() {
  Serial.begin(115200);

  // Initialize Pins
  pinMode(SW1_PIN, INPUT_PULLUP);
  pinMode(SW2_PIN, INPUT_PULLUP);
  pinMode(LDR_PIN, INPUT);

  // Initialize Display
  if (!display.begin(SSD1306_SWITCHCAPVCC, 0x3C)) {
    Serial.println(F("SSD1306 allocation failed"));
    for (;;); // Infinite loop on failure
  }

  // Initial display message
  display.clearDisplay();
  display.setTextSize(2);
  display.setTextColor(WHITE);
  display.setCursor(10, 25);
  display.println("Ready!");
  display.display();
  delay(1000);
```

```
  WiFi.begin(ssid, password);
  Serial.print("Connecting to WiFi");

  while (WiFi.status() != WL_CONNECTED)
  {
    delay(500);
    Serial.print(".");
  }
  Serial.println("\nWiFi Connected");
  secureClient.setInsecure();  // <------ ALLOW INSECURE
CONNECTION TO TLS SERVER

  client.setServer(mqtt_server, mqtt_port);
  client.setCallback(callback);


}

void loop() {


  if (!client.connected()) {
    reconnect();
  }
  client.loop();

//-----------------------------------------------------

  // --- 1. Read Inputs and Determine Current State ---
  bool sw1 = digitalRead(SW1_PIN);
  bool sw2 = digitalRead(SW2_PIN);

  if (!sw1 && sw2) {
    currentState = 1; // LDR State
  } else if (sw1 && !sw2) {
    currentState = 2; // Servo State
  } else if (sw1 && sw2) {
    currentState = 3; // RGB LED State
  } else {
    currentState = 0; // Default/idle state
  }

  // --- 2. Handle State Transitions (if state has changed) ---
  if (currentState != lastState) {
    // A. Clean up the PREVIOUS state
```

```
switch (lastState) {
  case 2: // Leaving Servo State
    myServo.detach();
    Serial.println("Servo detached");
    break;
  case 3: // Leaving RGB LED State
    setColor(0, 0, 0); // Turn off LED
    // Detach PWM pins to free up resources
    ledcDetach(RED_PIN);
    ledcDetach(GREEN_PIN);
    ledcDetach(BLUE_PIN);
    promptPrinted = false;
    Serial.println("RGB LED turned off and detached");
    break;
}

// B. Set up the NEW state
display.clearDisplay(); // Clear display on any state change
switch (currentState) {
  case 1: // Entering LDR State

    display.setTextSize(1);
    display.setTextColor(WHITE);
    display.setCursor(0, 20);
    display.println("State 1: LDR");
    Serial.println("Entered LDR State");
    break;

  case 2: // Entering Servo State

    display.setTextSize(1);
    display.setTextColor(WHITE);
    display.setCursor(0, 20);
    myServo.attach(SERVO_PIN);
    display.println("State 2: Servo");
    Serial.println("Entered Servo State & Servo Attached");

    break;
  case 3: // Entering RGB LED State
    // Attach pins to PWM channels

    display.clearDisplay();

    display.setTextSize(1);
    display.setCursor(0,10);
```

```
      display.println("State 3:RGB LED");

      display.setTextSize(1);
      display.setCursor(0,20);
      display.println("Enter colour:");

      ledcAttach(RED_PIN, 5000, 8);
      ledcAttach(GREEN_PIN, 5000, 8);
      ledcAttach(BLUE_PIN, 5000, 8);
      //display.println("State 3: RGB");
      Serial.println("Entered RGB State");
      break;
    case 0:
      display.println("Idle State");
      break;
  }
  display.display();
  delay(50); // Small delay to prevent flickering
}

// --- 3. Run the Logic for the Current State ---
switch (currentState) {
  case 1:
    runLdrState();
    break;
  case 2:
    runServoState();
    break;
  case 3:
    runRgbState();
    break;
}

// --- 4. Update lastState for the next loop ---
lastState = currentState;

delay(100); // Main loop delay


// Build JSON manually to match your required format
/*String payload = "{";
payload += "\"d1\":\"" + String(ldrValue) + "\",";
payload += "\"d2\":\"" + String(thermistorValue) + "\",";
payload += "\"d3\":\"" + String(mq6Value) + "\",";
payload += "\"header\":{";
```

```
  payload += "\"h1\":\"ldr\",";
  payload += "\"h2\":\"thermistor\",";
  payload += "\"h3\":\"mq6\"";
  payload += "}}";

  client.publish(mqtt_topic, payload.c_str());*/

//-----------------------------------------------------------------
----------------------------------------------------------



}


//=================================================================
=
//                  STATE-SPECIFIC FUNCTIONS
//=================================================================
=

/**
 * @brief Reads LDR value and displays it.
 */
void runLdrState() {
  int ldrValue = analogRead(LDR_PIN);

  Serial.print("LDR: ");
  Serial.println(ldrValue);

  display.clearDisplay();
  display.setTextSize(1);
  display.setCursor(0, 10);
  display.println("State 1: LDR Sensor");

  display.setTextSize(2);
  display.setCursor(0, 20);
  display.print("Value:");
  display.print(ldrValue);

  display.display();


  String payload = "{";
  payload += "\"d1\":\"" + String(ldrValue) + "\",";
```

```
  payload += "\"header\":{";
  payload += "\"h1\":\"LDR\"";
  payload += "}}";

  client.publish(mqtt_topic, payload.c_str());

  String payload1 = "\Data =" + String(ldrValue); + "\",";
  client.publish(mqtt_topic1, payload1.c_str());

  delay(500); // Slow down LDR reading display

}

/**
 * @brief Controls the servo motor.
 */
void runServoState() {
  // *** THE MAIN FIX IS HERE ***
  // Always send the write command to the servo on every loop.
  // This tells the servo to actively hold its position.
  myServo.write(currentAngle);

  // Check for new commands from the Serial Monitor
  if (Serial.available()) {
    String input = Serial.readStringUntil('\n');
    input.trim();
    int angle = input.toInt();

    if (angle >= 0 && angle <= 180) {
      currentAngle = angle; // Update the angle

      String payload_servo = "{";
      payload_servo += "\"angle\": " + String(currentAngle);
      payload_servo += "}";
      client.publish(mqtt_topic_servo_control,
payload_servo.c_str());

      // The myServo.write() at the top of this function will
handle the move
      Serial.print("Angle set to: ");
      Serial.println(currentAngle);
    } else {
      Serial.println("Invalid input. Enter angle 0-180.");
    }
  }
```

```
  // Update the OLED display
  display.clearDisplay();
  display.setTextSize(1);
  display.setCursor(0, 10);
  display.println("State 2:Servo Control");
  display.setCursor(0, 20);
  display.println("Current Angle:");
  display.setTextSize(2);
  display.setCursor(30, 40);
  display.print(currentAngle);
  display.print((char)247); // Degree symbol
  display.display();

  String payload = "{";
  payload += "\"d1\":\"" + String(currentAngle) + "\",";
  payload += "\"header\":{";
  payload += "\"h1\":\"SERVO\"";
  payload += "}}";

  client.publish(mqtt_topic, payload.c_str());

  String payload1 = "\Data =" + String(currentAngle); + "\",";
  client.publish(mqtt_topic1, payload1.c_str());

}


/**
 * @brief Controls the RGB LED based on serial input.
 */
void runRgbState() {
  if (!promptPrinted) {
    Serial.println("Enter color: red, green, blue, yellow, cyan,
pink");
    promptPrinted = true;
  }

  if (Serial.available()) {
    String color = Serial.readStringUntil('\n');
    color.trim();
    color.toLowerCase();

    display.clearDisplay();
```

```
display.setTextSize(1);
display.setCursor(0,10);
display.println("State 3:RGB LED");

display.setTextSize(1);
display.setCursor(0,20);
display.println("Enter colour:");

display.setTextSize(2);
display.setCursor(0, 30);

if (color == "red") {
  setColor(255, 0, 0);
  display.print("Red");

  String payload = "{";
  payload += "\"d1\":\"" + String(color) + "\",";
  payload += "\"header\":{";
  payload += "\"h1\":\"RGB\"";
  payload += "}}";

  client.publish(mqtt_topic, payload.c_str());

  String payload1 = "\Data =" + String(color); + "\",";
  client.publish(mqtt_topic1, payload1.c_str());

} else if (color == "green") {
  setColor(0, 255, 0);
  display.print("Green");

  String payload = "{";
  payload += "\"d1\":\"" + String(color) + "\",";
  payload += "\"header\":{";
  payload += "\"h1\":\"RGB\"";
  payload += "}}";

  client.publish(mqtt_topic, payload.c_str());

  String payload1 = "\Data =" + String(color); + "\",";
  client.publish(mqtt_topic1, payload1.c_str());

} else if (color == "blue") {
  setColor(0, 0, 255);
  display.print("Blue");
```

```
  String payload = "{";
  payload += "\"d1\":\"" + String(color) + "\",";
  payload += "\"header\":{";
  payload += "\"h1\":\"RGB\"";
  payload += "}}";

  client.publish(mqtt_topic, payload.c_str());

  String payload1 = "\Data =" + String(color); + "\",";
  client.publish(mqtt_topic1, payload1.c_str());

} else if (color == "yellow") {
  setColor(255, 255, 0);
  display.print("Yellow");

  String payload = "{";
  payload += "\"d1\":\"" + String(color) + "\",";
  payload += "\"header\":{";
  payload += "\"h1\":\"RGB\"";
  payload += "}}";

  client.publish(mqtt_topic, payload.c_str());

  String payload1 = "\Data =" + String(color); + "\",";
  client.publish(mqtt_topic1, payload1.c_str());

} else if (color == "cyan") {
  setColor(0, 255, 255);
  display.print("Cyan");

  String payload = "{";
  payload += "\"d1\":\"" + String(color) + "\",";
  payload += "\"header\":{";
  payload += "\"h1\":\"RGB\"";
  payload += "}}";

  client.publish(mqtt_topic, payload.c_str());

  String payload1 = "\Data =" + String(color); + "\",";
  client.publish(mqtt_topic1, payload1.c_str());

} else if (color == "pink") {
  setColor(255, 0, 127);
  display.print("Pink");
```

```
      String payload = "{";
      payload += "\"d1\":\"" + String(color) + "\",";
      payload += "\"header\":{";
      payload += "\"h1\":\"RGB\"";
      payload += "}}";

      client.publish(mqtt_topic, payload.c_str());

      String payload1 = "\Data =" + String(color); + "\",";
      client.publish(mqtt_topic1, payload1.c_str());

    } else {
      display.setTextSize(1);
      display.setCursor(0,20);
      display.println("Unknown color!");
    }
    display.display();
  }

}

/**
 * @brief Sets the color of the RGB LED.
 * @param r Red value (0-255)
 * @param g Green value (0-255)
 * @param b Blue value (0-255)
 */
void setColor(int r, int g, int b) {
  ledcWrite(RED_PIN, r);
  ledcWrite(GREEN_PIN, g);
  ledcWrite(BLUE_PIN, b);
}
```

Created by: Jayant Kumar
Powered by: ESP32, MQTT, and Innovation