Smart Color Mixer using RGB LED

Session: 2022-2026

Submitted by:

Muhammad Wali Ahmad	2022-CS-65
Bisma Fajar	2022-CS-66
Leena Zaheer	2022-CS-72
Amna Nadeem	2022-CS-96

Submitted to:

Prof. Syed Tehseen Ul Hasan Shah

Department of Computer Science

University Of Enigneering And Technology,

Lahore, Pakistan

Contents

List of Figures

Smart Color Mixer using RGB LED

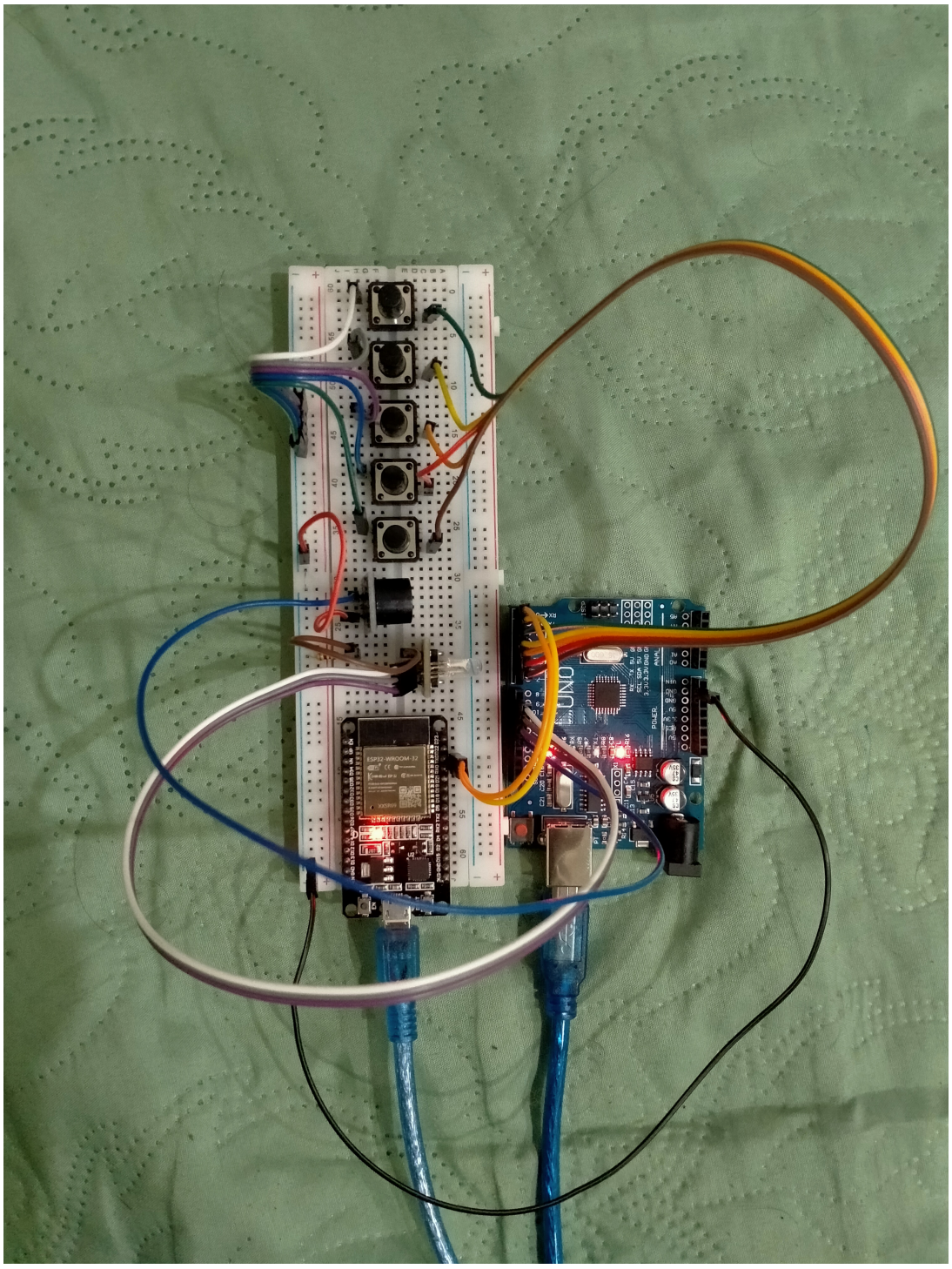

Figure 1: Colored Photo of Project

Description

The Smart Color Mixer project represents a cutting-edge exploration in the realm of color control, leveraging the capabilities of an Arduino UNO and an ESP32. This endeavor is rooted in the idea of providing users with a dynamic and interactive means to manipulate an RGB LED, generating an extensive spectrum of colors effortlessly. The central control hub for this innovation is the MQTT Dashboard app, a user-friendly interface that facilitates seamless communication between the user's smartphone and the RGB LED system.

1 Features and Modes

The project incorporates a range of features, each enhancing the user experience. Five distinct modes have been meticulously crafted to cater to different color preferences and dynamic visual effects:

1.1 Mode 1: Yellow Color Production (Button#1)

Upon pressing Button#1, the RGB LED seamlessly blends the red and green components, producing a vibrant yellow color.

1.2 Mode 2: Magenta Color Production (Button#2)

Button#2 triggers the combination of red and blue LEDs, resulting in an alluring magenta hue.

1.3 Mode 3: White Color Production (Button#3)

Button#3 orchestrates the illumination of all three LEDs simultaneously, generating a pure white light.

1.4 Mode 4: Sequential LED Animation (Button#4)

Engaging Button#4 initiates a captivating sequence where all LEDs turn on and off in a loop, with a half-second delay between each LED.

1.5 Mode 5: All LEDs Off (Button#5)

Button#5 offers a quick and efficient way to turn off all LEDs, providing users with control over ambient lighting conditions.

2 Startup Sequence and Buzzer Feedback

Upon powering up, the system greets users with a distinctive three-second beep from the integrated buzzer, creating an audible indication of the system initialization. The RGB LED illuminate all three LEDs simultaneously, generating a pure white light and then turn off after 2 seconds.

3 IoT Integration

The integration of an ESP32 as an IoT module introduces a layer of remote control and monitoring to the Smart Color Mixer. Users can seamlessly interact with the system through the MQTT Dashboard app, issuing color commands and receiving real-time feedback on the current operational mode. This innovative integration not only enhances user convenience but also expands the potential use cases for the RGB LED system.

4 Cloud Connectivity

In addition to remote control, the IoT module facilitates cloud connectivity through ThingSpeak. The current RGB LED color values are periodically sent to ThingSpeak, creating a log of color transitions over time. This data logging feature enables users to track and analyze the historical usage patterns of the Smart Color Mixer, offering valuable insights into lighting preferences and usage trends.

5 User-Friendly Interface

The MQTT Dashboard app's dashboard serves as a centralized control panel, presenting users with an intuitive interface. The five buttons corresponding to each mode. This visual clarity enhances the overall user experience, making the Smart Color Mixer accessible and enjoyable for users of all technical backgrounds.

6 Future Scope

The Smart Color Mixer project lays the foundation for future enhancements and expansions. Potential avenues for development include additional color modes, integration with voice commands for hands-free operation, and compatibility with other IoT platforms. The modular design of the system encourages further innovation and customization to meet the evolving needs of users and the broader IoT ecosystem.

Methodology

The Smart Color Mixer project adopts a systematic methodology that encompasses both hardware and software components. The following sections provide a detailed account of the methodology employed for the development and implementation of this innovative RGB LED control system.

1 Hardware Components Selection and Configuration

The first step involves the careful selection and configuration of hardware components. The Arduino UNO serves as the AVR module, responsible for local control of the RGB LED. The ESP32, acting as the IoT module, facilitates remote control and monitoring. Additional components include a buzzer module, a breadboard, pin-to-pin jumper wires, and a 100-ohm resistor for current control. Each component is selected based on its specific role in the system.

2 Assembly and Circuit Design

The hardware components are assembled on a breadboard, following a meticulously designed circuit. The RGB LED module, with individual Red, Green, and Blue LEDs, is connected to specific pins on the Arduino UNO. Button inputs are integrated to provide a tactile means of user interaction. The EasyEDA platform is utilized for creating a comprehensive circuit diagram, ensuring proper voltage regulation, current limiting, and logical connections.

3 AVR Module Programming

The AVR module, driven by the Arduino UNO, is programmed in assembly language in Microchip Studio. The code encompasses color-mixing algorithms, LED animations, and button-based functionalities. This module is responsible for local control, ensuring precise manipulation of the RGB LED based on user inputs and system states.

4 IoT Module Programming

The IoT module, featuring the ESP32, is programmed in C in Arduino IDE. The code establishes communication between the MQTT Dashboard app and the Arduino UNO. It fetches color commands, transmits them securely over MQTT to the AVR module. Additionally, the IoT module sends RGB LED color data to ThingSpeak for cloud logging.

5 System Integration and Testing

The integrated system undergoes rigorous testing to ensure proper interoperability between the local and remote control modules. Color commands are verified for accuracy, and the RGB LED responds as expected. The startup sequences, LED animations, and cloud connectivity are thoroughly tested to validate the robustness and reliability of the Smart Color Mixer.

6 Documentation and Reporting

Detailed documentation is an integral part of the methodology, ensuring transparency and reproducibility. LaTeX is employed to create a comprehensive project report that includes detailed explanations of the circuit design, programming logic, and system functionalities.

Data Flow Diagram

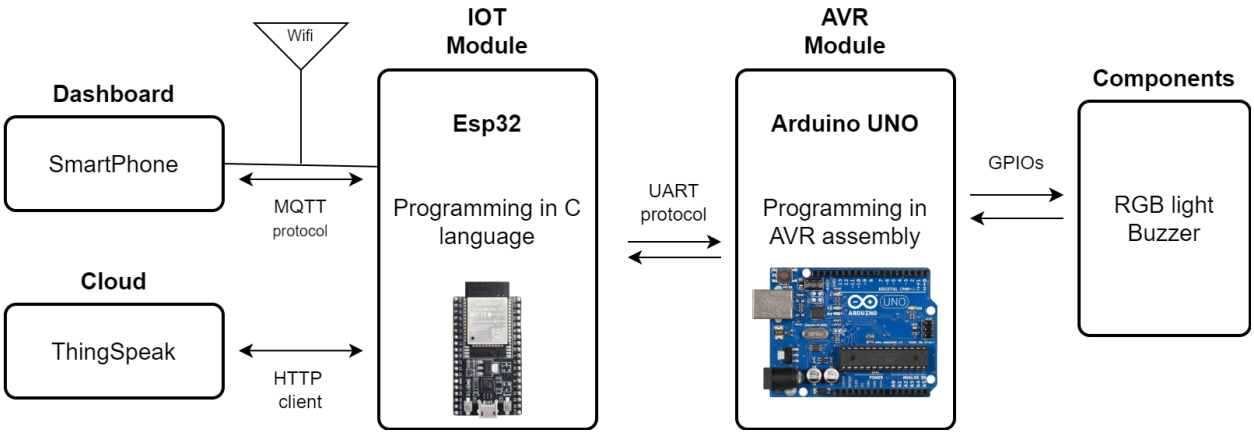

Figure 2: Data Flow Diagram

Detailed Flow Chart

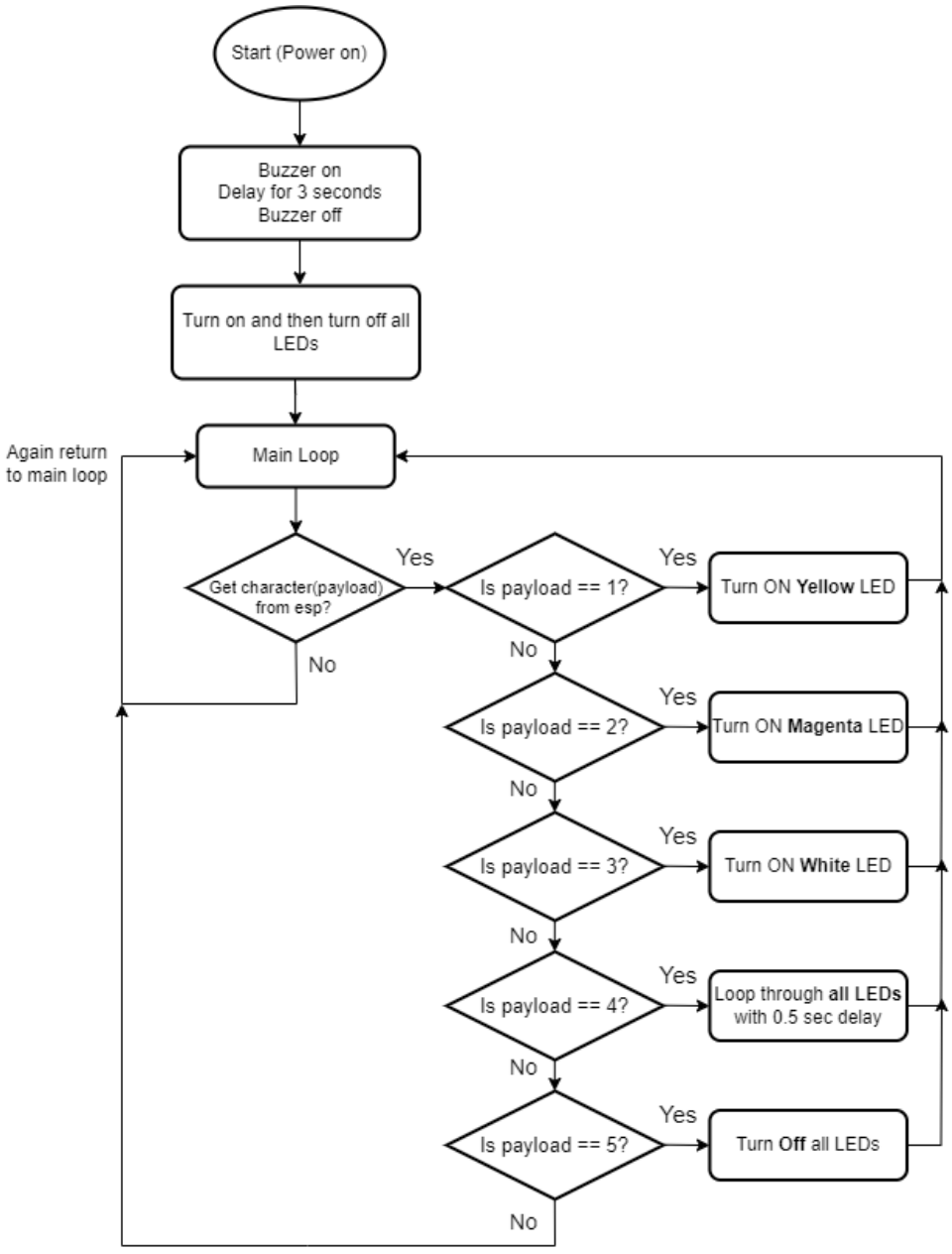

Figure 3: Arduino Uno Data Flow Chart

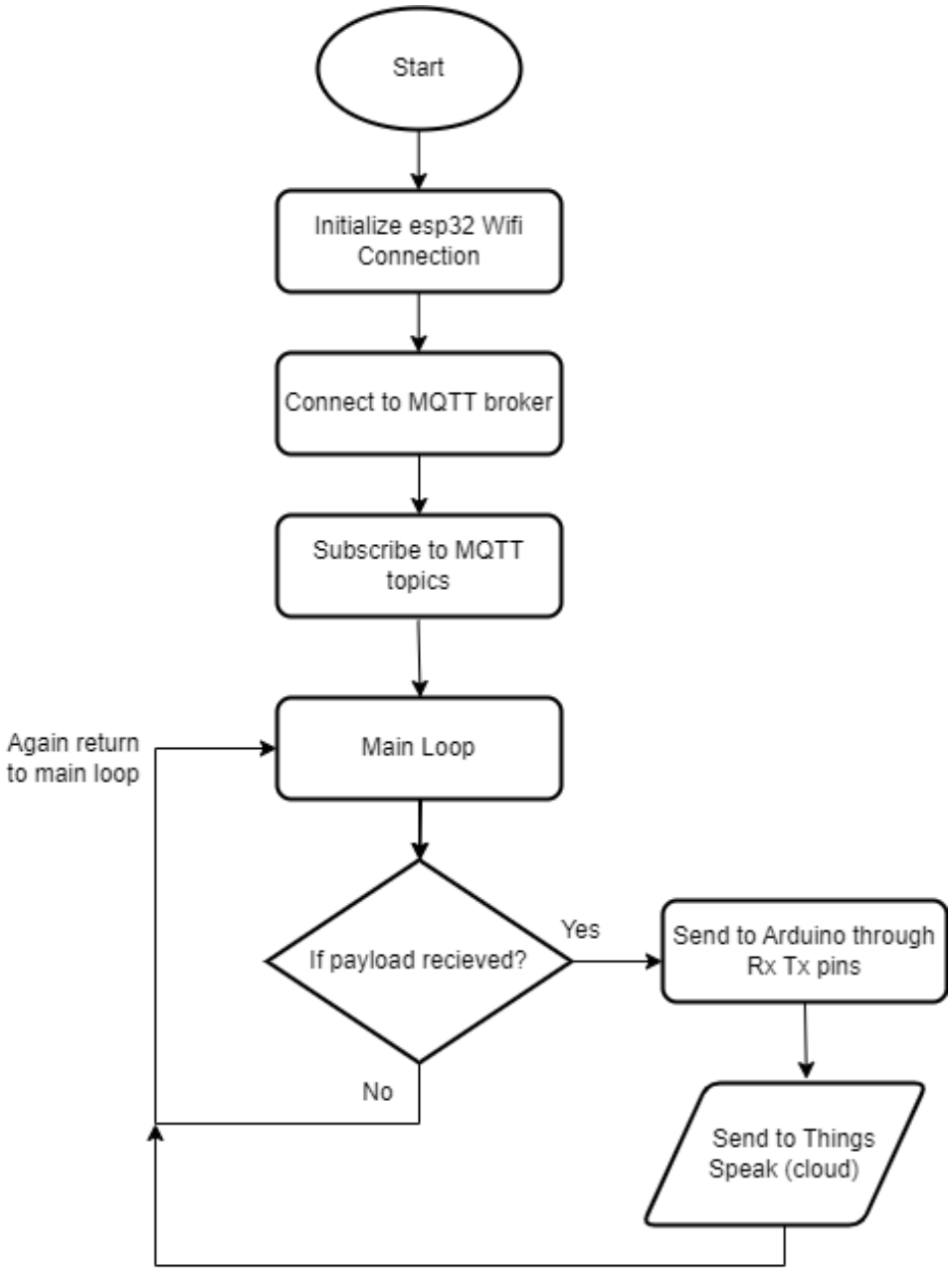

Figure 4: ESP32 Data Flow Chart

Circuit Diagram

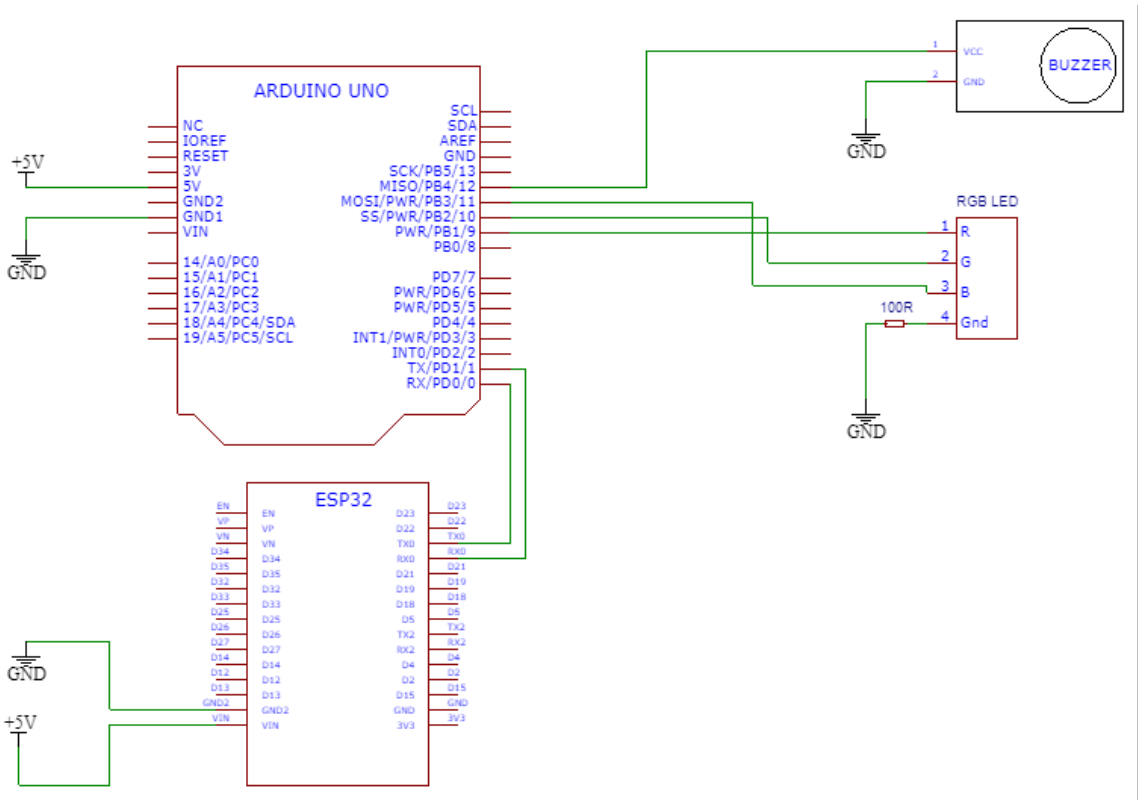

Figure 5: Circuit Diagram

Detailed Explanation of Components

The Smart Color Mixer project integrates various hardware components, each with a distinct role in achieving the system's functionality. The following subsections provide a comprehensive breakdown of the workings of each component, elucidating their specific functions and contributions to the overall system.

1 Arduino UNO (AVR Module)

The Arduino UNO serves as the AVR (Advanced Virtual RISC) module, functioning as the local controller for the RGB LED. Its primary responsibilities include reading user inputs from the buttons, executing color-mixing algorithms, and controlling the RGB LED based on the selected mode.

1.1 Button Input Processing

The Arduino UNO is equipped with digital pins configured to read input from buttons. Each button corresponds to a specific mode (yellow, magenta, white, sequential animation, and all LEDs off). The Arduino detects button presses and triggers the associated mode accordingly.

1.2 RGB LED Control

The RGB LED module consists of individual Red, Green, and Blue LEDs. The Arduino UNO controls each LED's intensity to achieve the desired color. For example, to produce yellow, both the red and green LEDs are illuminated simultaneously.

1.3 Startup Sequence

Upon power-up, the Arduino UNO initiates a distinctive three-second beep through the buzzer module and executes a brief LED animation. This sequence serves as a visual and auditory indication of the system's readiness.

2 ESP32 (IoT Module)

The ESP32 acts as the IoT module, facilitating remote control and communication with the MQTT Dashboard app. It bridges the gap between the MQTT Dashboard app and the Arduino UNO, enabling users to interact with the RGB LED system from their smartphones.

2.1 MQTT Communication

The ESP32 establishes a secure MQTT connection with the MQTT Dashboard app. It subscribes to color commands and publishes real-time feedback on the current operational

mode. MQTT ensures reliable and efficient communication between the IoT module and the app.

2.2 Color Command Processing

Upon receiving color commands from the MQTT Dashboard app, the ESP32 forwards these commands to the Arduino UNO. It interprets the commands and triggers the corresponding mode, instructing the Arduino UNO on the desired color or animation sequence.

2.3 ThingSpeak Integration

The ESP32 periodically sends RGB LED color data to ThingSpeak, a cloud platform. This integration allows for the logging and tracking of color transitions over time, providing users with insights into historical usage patterns.

3 Buzzer Module

The buzzer module adds an auditory element to the Smart Color Mixer, enhancing user feedback and system interaction.

3.1 Startup Beep

During the startup sequence, the buzzer emits a distinctive three-second beep, signaling the initiation of the system. This audible feedback complements the visual LED animation, ensuring users are aware of the system's operational state.

4 RGB LED Module

The RGB LED module is at the heart of the project, responsible for producing a wide range of colors based on user inputs and operational modes.

4.1 Color Mixing

The RGB LED module consists of three primary color LEDs—Red, Green, and Blue. By controlling the intensity of each LED, the module achieves various color combinations. For example, activating the Red and Green LEDs simultaneously produces yellow.

4.2 Sequential Animation

In the sequential animation mode, the RGB LED module cycles through all three primary colors and the colors produced from these colors in a loop with a specified delay. This mode introduces a dynamic visual element, creating an animated lighting effect.

5 Buttons and Breadboard

Physical buttons are integrated into the system to provide users with a tangible means of input. These buttons are connected to digital pins on the Arduino UNO and are strategically placed for user accessibility. The breadboard serves as the platform for assembling and connecting various components, facilitating a neat and organized circuit layout.

5.1 User Input

The buttons act as user inputs, allowing users to select different color modes and initiate specific operations. Each button is associated with a particular mode, and pressing the button triggers the corresponding action.

5.2 Circuit Organization

The breadboard serves as the physical platform for assembling the circuit. It enables the neat arrangement of components and ensures proper connectivity, minimizing the risk of errors during the assembly process.

This detailed explanation provides insights into how each component contributes to the overall functionality of the Smart Color Mixer. Understanding the roles and interactions of these components is crucial for both system development and troubleshooting.

AVR Code

```
.include "m328pdef.inc"
.include "UART_Macros.inc"
.include "delay.inc"

.def UART_CHAR = r21

.org 0x00
                                        ; UART Configuration
        SBI DDRD, PD1 ; Set PD1 (TX) as Output
        CBI PORTD, PD1 ; TX Low (initial state)
        CBI DDRD, PD0 ; Set PD0 (RX) as Input
        SBI PORTD, PD0 ; Enable Pull-up Resistor on RX
        Serial_begin ; Initialize UART Protocol

                                        ; OUTPUT CONFIGURATION
        SBI DDRB, PB1 ; PB1 set as OUTPUT Pin RED LIGHT
        CBI PORTB, PB1 ; LED OFF

        SBI DDRB, PB2 ; PB2 set as OUTPUT Pin GREEN LIGHT
        CBI PORTB, PB2 ; LED OFF

        SBI DDRB, PB3 ; PB3 set as OUTPUT Pin BLUE LIGHT
        CBI PORTB, PB3 ; LED OFF

        SBI DDRB, PB4 ; PB3 set as OUTPUT Pin for BUZZER

        SBI PORTB, PB4 ; TURN BUZZER ON
        delay 1500
        delay 1500
        CBI PORTB, PB4 ; TURN BUZZER OFF

        SBI PORTB, PB1  ;RED LED ON
        SBI PORTB, PB2  ;GREEN LED ON
        SBI PORTB, PB3  ;BLUE LED ON
        delay 2000
        CBI PORTB, PB1  ;RED LED ON
        CBI PORTB, PB2  ;GREEN LED ON
        CBI PORTB, PB3  ;BLUE LED ON
```

```
MAIN:
        Serial_readChar UART_CHAR
MID:
        CPI UART_CHAR, '1'      ; Yellow COLOR
    BREQ YELLOW_LED

    CPI UART_CHAR, '2'  ; Magenta COLOR
    BREQ MAGENTA_LED

        CPI UART_CHAR, '3'      ; White COLOR
    BREQ WHITE_LED

        CPI UART_CHAR, '4'  ; Loop LEDs
    BREQ LOOP_LEDS

        CPI UART_CHAR, '5'  ; OFF LEDs
    BREQ LEDS_OFF
rjmp MAIN

LEDS_OFF:
        CBI PORTB, PB1  ;RED LED OFF
        CBI PORTB, PB2  ;GREEN LED OFF
        CBI PORTB, PB3  ;BLUE LED OFF
rjmp MAIN

YELLOW_LED:
        SBI PORTB, PB1  ;RED LED ON
        SBI PORTB, PB2  ;GREEN LED ON
        CBI PORTB, PB3  ;BLUE LED OFF
rjmp MAIN

MAGENTA_LED:
        SBI PORTB, PB1  ;RED LED ON
        CBI PORTB, PB2  ;GREEN LED OFF
        SBI PORTB, PB3  ;BLUE LED ON
rjmp MAIN

WHITE_LED:
        SBI PORTB, PB1  ;RED LED ON
        SBI PORTB, PB2  ;GREEN LED ON
        SBI PORTB, PB3  ;BLUE LED ON
rjmp MAIN
```

```
LOOP_LEDS:
        Serial_readChar UART_CHAR

        CPI UART_CHAR, '1'       ; Yellow COLOR
    BREQ MID

    CPI UART_CHAR, '2'   ; Magenta COLOR
    BREQ MID

        CPI UART_CHAR, '3'       ; White COLOR
    BREQ MID

        CPI UART_CHAR, '4'   ; Loop LEDs
    BREQ MID

        CPI UART_CHAR, '5'   ; OFF LEDs
    BREQ MID

        SBI PORTB, PB1   ;RED LED ON
        CBI PORTB, PB2   ;GREEN LED OFF
        CBI PORTB, PB3   ;BLUE LED OFF

        delay 500

        CBI PORTB, PB1   ;RED LED OFF
        SBI PORTB, PB2   ;GREEN LED ON
        CBI PORTB, PB3   ;BLUE LED OFF

        delay 500

        CBI PORTB, PB1   ;RED LED OFF
        CBI PORTB, PB2   ;GREEN LED OFF
        SBI PORTB, PB3   ;BLUE LED ON

        delay 500

        SBI PORTB, PB1   ;RED LED ON
        SBI PORTB, PB2   ;GREEN LED ON
        CBI PORTB, PB3   ;BLUE LED OFF

        delay 500

        SBI PORTB, PB1   ;RED LED ON
```

```
        CBI PORTB, PB2  ;GREEN LED OFF
        SBI PORTB, PB3  ;BLUE LED ON

        delay 500

        CBI PORTB, PB1  ;RED LED OFF
        SBI PORTB, PB2  ;GREEN LED ON
        SBI PORTB, PB3  ;BLUE LED ON

        delay 500

        SBI PORTB, PB1  ;RED LED ON
        SBI PORTB, PB2  ;GREEN LED ON
        SBI PORTB, PB3  ;BLUE LED ON

        delay 500

        CBI PORTB, PB1  ;RED LED OFF
        CBI PORTB, PB2  ;GREEN LED OFF
        CBI PORTB, PB3  ;BLUE LED OFF

        Serial_readChar UART_CHAR

        delay 500

rjmp LOOP_LEDS
```

IoT Code

```
#include <WiFi.h>
#include <PubSubClient.h>
#include <HTTPClient.h>

// WiFi
const char *ssid = "Leena";   // Enter your Wi-Fi name
const char *password = "11223344";  // Enter Wi-Fi password

// MQTT Broker
const char *mqtt_broker = "Broker.hivemq.com";
const char *topic = "rgb";
const char *mqtt_username = "";
const char *mqtt_password = "";
const int mqtt_port = 1883;

// ThingSpeak
const char *thingSpeakApiKey = "BLVZ15542DF9TI7K";
const char *thingSpeakUrl = "http://api.thingspeak.com/update?
   api_key=";

WiFiClient espClient;
PubSubClient client(espClient);

void setup() {
  Serial.begin(9600);
  WiFi.mode(WIFI_STA);
  WiFi.onEvent(ConnectedToAP_Handler,
     ARDUINO_EVENT_WIFI_STA_CONNECTED);
  WiFi.onEvent(GotIP_Handler, ARDUINO_EVENT_WIFI_STA_GOT_IP);
  WiFi.onEvent(WiFi_Disconnected_Handler,
     ARDUINO_EVENT_WIFI_STA_DISCONNECTED);
  WiFi.begin(ssid, password);
  Serial.println("Connecting to WiFi Network ..");

  //connecting to an MQTT broker
  client.setServer(mqtt_broker, mqtt_port);
  client.setCallback(callback);
  while (!client.connected()) {
    String client_id = "esp32-client-";
```

```
    client_id += String(WiFi.macAddress());
    Serial.printf("The client %s connects to the public MQTT
       broker\n", client_id.c_str());
    if (client.connect(client_id.c_str(), mqtt_username,
       mqtt_password)) {
      Serial.println("Public EMQX MQTT broker connected");
    } else {
      Serial.print("failed with state ");
      Serial.print(client.state());
      delay(2000);
    }
  }

  client.subscribe(topic);
}

void callback(char *topic, byte *payload, unsigned int length)
    {
  for (int i = 0; i < length; i++) {
    Serial.write((char)payload[i]);
    sendDataToThingSpeak(String((char)payload[i]));  // Send
       data to ThingSpeak
  }
  Serial.println();
}

void loop() {
  client.loop();
  delay(2000);
}

void ConnectedToAP_Handler(WiFiEvent_t wifi_event,
   WiFiEventInfo_t wifi_info) {
  Serial.println("Connected To The WiFi Network");
}

void GotIP_Handler(WiFiEvent_t wifi_event, WiFiEventInfo_t
   wifi_info) {
  Serial.print("Local ESP32 IP: ");
  Serial.println(WiFi.localIP());
}

void WiFi_Disconnected_Handler(WiFiEvent_t wifi_event,
```

```
   WiFiEventInfo_t wifi_info) {
  Serial.println("Disconnected From WiFi Network");
  // Attempt Re-Connection
  WiFi.begin(ssid, password);
}

void sendDataToThingSpeak(String data) {
  // Create a HTTP client object
  HTTPClient http;
  // Construct the ThingSpeak URL with your API key and data
  String url = "";

  if (data == "1") {
    url = String(thingSpeakUrl) + String(thingSpeakApiKey) +
      "&field1=" + "1" + "&field2=" + "1" + "&field3=" + "0"+
       "&field4=" + "0";
  }
  if (data == "2") {
    url = String(thingSpeakUrl) + String(thingSpeakApiKey) +
      "&field1=" + "1" + "&field2=" + "0" + "&field3=" + "1"+
       "&field4=" + "0";
  }
  if (data == "3") {
    url = String(thingSpeakUrl) + String(thingSpeakApiKey) +
      "&field1=" + "1" + "&field2=" + "1" + "&field3=" + "1"+
       "&field4=" + "0";
  }
  if (data == "4") {
    url = String(thingSpeakUrl) + String(thingSpeakApiKey) +
      "&field1=" + "0" + "&field2=" + "0" + "&field3=" + "0"+
       "&field4=" + "1";
  }
  if (data == "5") {
    url = String(thingSpeakUrl) + String(thingSpeakApiKey) +
      "&field1=" + "0" + "&field2=" + "0" + "&field3=" + "0"+
       "&field4=" + "0";
  }

  // Begin the HTTP request
  http.begin(url);

  // Send the HTTP GET request
  int httpCode = http.GET();
```

```
  // Check the HTTP response code
  if (httpCode == HTTP_CODE_OK) {
    Serial.println("Data sent to ThingSpeak successfully");
  } else {
    Serial.print("Failed to send data to ThingSpeak. HTTP code
       : ");
    Serial.println(httpCode);
  }

  // End the HTTP request
  http.end();
}
```

Code Documentation

1 IOT module Code Documentation

1.1 Overview

This Arduino code connects an ESP32 device to an MQTT broker and ThingSpeak cloud service to receive and forward sensor data. It subscribes to an MQTT topic, processes incoming payloads, and sends corresponding data to ThingSpeak channels.

1.2 Key Libraries:

- **WiFi.h:** Enables Wi-Fi connection and configuration.
- **PubSubClient.h:** Implements MQTT client functionality.
- **HTTPClient.h:** Facilitates HTTP requests to ThingSpeak API.

1.3 Global Constants:

- **ssid:** Holds the Wi-Fi network name.
- **password:** Holds the Wi-Fi password.
- **mqtt_broker:** Specifies the MQTT broker address.
- **topic:** Defines the MQTT topic to subscribe to.
- **mqtt_username:** Stores the MQTT username (if required).
- **mqtt_password:** Stores the MQTT password (if required).
- **mqtt_port:** Specifies the MQTT broker port.
- **thingSpeakApiKey:** Holds the ThingSpeak API key.
- **thingSpeakUrl:** Stores the base URL for ThingSpeak API requests.

1.4 Main Functions:

setup()

- Initializes serial communication.
- Configures Wi-Fi connection and sets up event handlers.
- Connects to the MQTT broker.
- Subscribes to the specified MQTT topic.

loop()

- Maintains the MQTT connection.
- Implements a 2-second delay for resource management.

callback(char topic, byte payload, unsigned int length)

- Processes incoming MQTT messages.
- Prints payload to the serial monitor.
- Calls `sendDataToThingSpeak()` for each payload byte.

sendDataToThingSpeak(String data)

- Constructs ThingSpeak URL based on the data value.
- Sends an HTTP GET request to ThingSpeak API.
- Handles response success or failure.

1.5 Additional Functions

ConnectedToAP_Handler(),GotIP_Handler()
WiFi_Disconnected_Handler()

- Handle Wi-Fi connection events and provide status updates.

1.6 Code Style and Best Practices

- **Indentation:** Consistent 2-space indentation for readability.
- **Comments:** Explain key code blocks and logic.
- **Meaningful Variable Names:** Descriptive names for variables and functions.
- **Error Handling:** Includes basic error handling for MQTT and ThingSpeak connections.

1.7 Possible Improvements

- **More Detailed Comments:** Expand comments for better understanding.
- **Logging:** Implement logging for debugging and monitoring.
- **Input Validation:** Validate user inputs for Wi-Fi credentials and ThingSpeak API key.
- **Error Handling:** Enhance error handling for robustness.
- **Security:** Consider security measures for sensitive data.

2 AVR module code documentation

2.1 Overview

This AVR assembly code controls RGB LEDs and a buzzer based on commands received via UART communication. It initializes UART, configures I/O pins, and implements a command-driven loop to manage LED and buzzer behavior.

2.2 Key Includes

- **m328pdef.inc:** Provides definitions for registers and I/O ports for the ATmega328P microcontroller.
- **UART_Macros.inc:** Contains macros for simplifying UART operations.
- **delay.inc:** Offers functions for creating delays.

2.3 Global Constants

- **UART_CHAR:** Register r21, used for storing received UART characters.

2.4 Initialization

UART Configuration

- Sets PD1 as output for TX.
- Sets PD0 as input for RX with a pull-up resistor.
- Calls `Serial_begin` to initialize UART.

Output Configuration

- Sets PB1, PB2, and PB3 as outputs for RGB LEDs.
- Sets PB4 as output for the buzzer.

2.5 Main Loop

Main

- Reads a character from UART and stores it in `UART_CHAR`.
- Compares `UART_CHAR` with various commands:
  - '1': Activates yellow LED.
  - '2': Activates magenta LED.
  - '3': Activates white LED.
  - '4': Enters a loop for cycling through LED colors.

  - '5': Turns off all LEDs.
- Jumps back to `MAIN` to continue listening for commands.

2.6 Command-Specific Functions

- **LEDS_OFF:** Turns off all LEDs.
- **YELLOW_LED:** Turns on red and green LEDs, creating yellow.
- **MAGENTA_LED:** Turns on red and blue LEDs, creating magenta.
- **WHITE_LED:** Turns on all LEDs to produce white.
- **LOOP_LEDS:** Continuously cycles through LED colors:
  - Red, green, blue, yellow, magenta, cyan, white, and all off.
  - Checks for UART commands to break out of the loop.

2.7 Code Style

- Uses uppercase for register names and labels.
- Employs consistent indentation for readability.
- Includes comments to explain key code sections.

2.8 Possible Improvements

- **More Extensive Comments:** Add comments to explain individual instructions and logic.
- **Robust Error Handling:** Implement routines to handle UART communication errors or invalid commands.
- **Modular Structure:** Refactor code into functions for better organization and reusability.
- **Lookup Table for Colors:** Use a lookup table to simplify LED color control, reducing code size and potential errors.

MQTT App's Dashboard Screenshot

Figure 6: MQTT app Screenshot

Project video Links

Here is the link of the Youtube Video: https://youtu.be/pDtTDaCUd7E

Here is the link of the LinkedIn Video: https://www.linkedin.com/posts/muhammadwaliahmad-arduino-esp32-rgbled-activity-7148242745965400064-PseI.

GitHub link

Here is the link to the GitHub repository: https://github.com/mwaliahmad/Coal-Final-Project

Reference

For ESP32:

- (Code to connect ESP to WiFi):
  https://deepbluembedded.com/esp32-connect-to-wifi-network-arduino/
- (Code to connect ESP to broker):
  https://iotdesignpro.com/projects/how-to-connect-esp32-mqtt-broker
- (Code to send data from ESP to ThingSpeak):
  https://iotdesignpro.com/projects/how-to-send-data-to-thingspeak-cloud-using-esp32

FOR AVR

- (We have used different macros from repo):
  https://github.com/TehseenHasan/AVR-Assembly-Example-Codes-for-Atmega328p