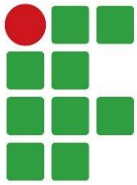


IFPE – Campus Garanhuns

Análise e desenvolvimento de sistemas

Carlos Gabryel Borges Cavalcante

Controle de LED RGB via MQTT no ESP32 com PWM

Garanhuns

2025

1. Introdução

Este relatório técnico detalha o desenvolvimento e a implementação de um sistema de controle de iluminação inteligente, concebido como um projeto da disciplina de Sistemas Embarcados. O objetivo principal do projeto é desenvolver um sistema embarcado para controle remoto de uma fita de LED RGB de alta potência (24V) via Wi-Fi, onde é utilizado um ESP32 e o protocolo MQTT, com integração a um broker público e um frontend web que possibilita que o usuário envie os comandos e controle a iluminação de maneira mais visual e tenha status do sistema, permitindo ter uma visão do funcionamento atual da fita LED.

O sistema foi desenvolvido para resolver o controle remoto de LEDs RGB com feedback em tempo real, permitindo ajustes precisos de cor, brilho e sequências personalizadas. Ele elimina a necessidade de interação física com o dispositivo, sincronizando perfeitamente o estado entre o hardware e a interface web. Além disso, supera desafios de comunicação confiável em redes Wi-Fi, garantindo que todos os comandos sejam executados mesmo em condições de conexão instável. Por fim, oferece uma solução escalável para automação residencial ou aplicações de iluminação inteligente.

Contexto:

- Aplicação em automação residencial ou cenários que demandam controle dinâmico de iluminação.

- Uso de transistores para chaveamento de cargas de alta tensão (24V) a partir de sinais PWM do ESP32 (3.3V).

Escopo:

- Hardware: ESP32, LED RGB, transistores NPN, resistores, fonte dual (5V/24V).

- Software: Firmware desenvolvido no Arduino IDE, broker MQTT público (HiveMQ), e protocolo JSON para comunicação.

2. Desenvolvimento

2.1 Componentes do Sistema

Componente	Função	Detalhes
ESP32-WROOM-32	Microcontrolador	Wi-Fi, Bluetooth, 3.3V logic, PWM.
LED RGB	Iluminação	Não endereçável, 24V, anodo comum.
Transistores 2SC2688	Chaveamento	NPN, $\beta \approx 100$, V_{CE} max = 50V.
Resistores	Limitar corrente de base	1kΩ (calculado para $I_B \approx 3mA$).
Fonte 24V	Alimentação do LED	Fonte externa dedicada.

2.2 Mapa de Pinagem do ESP32

Pino ESP32	Conexão	Função

GPIO14	Conectado à Base do Transistor do LED Vermelho.	Saída Digital (PWM)
GPIO27	Conectado à Base do Transistor do LED Verde.	Saída Digital (PWM)
GPIO26	Conectado à Base do Transistor do LED Azul.	Saída Digital (PWM)
GND	Conectado ao GND da fonte de alimentação de 24V.	Aterramento

2.3 Protocolos e Comunicação

- MQTT:
  - Broker: broker.hivemq.com (porta 1883)
  - Tópicos:
    - carlos/rgb_pwm (comandos)
    - carlos/rgb_pwm/status (feedback do sistema)
  - Mensagens: String simples ou JSON (ex: {“brightness”: 75}
- Wi-Fi:
  - Conexão direta à rede local (STA mode)

2.4 Diagrama de funcionamento

1. ESP32 conecta-se ao Wi-Fi e MQTT.
2. Recebe comandos via tópico MQTT através da interface web.
3. Gera sinais PWM nos GPIOs correspondentes.
4. Transistores amplificam o sinal para o LED RGB.

3. Resultados

3.1 Funcionamento do Sistema

- Controle remoto:
  - Comandos via MQTT alteram cores, modos e brilho em tempo real.
  - Latência medida: < 500ms (depende da rede Wi-Fi).
- Modos Implementados:

```
enum {
 MODE_OFF,
 MODE_FAST_COLOR_CHANGE,  // Transição rápida entre cores
 MODE_SLOW_COLOR_CHANGE,  // Transição lenta
 MODE_CUSTOM_SEQUENCE     // Sequência personalizada via JSON
};
```

O sistema foi testado e demonstrou total funcionalidade.

- Comunicação Eficiente: A latência entre o comando na aplicação web e a resposta no LED RGB foi mínima, provando a eficiência do protocolo MQTT para este tipo de aplicação.
- Controle Flexível: Os diferentes modos de operação (mudança rápida, mudança lenta, piscar vermelho) e o controle de brilho funcionaram conforme o esperado.
- Sequência Personalizada: A funcionalidade de sequência personalizada (enviada via JSON) provou a robustez do sistema para comandos mais complexos.
- Sincronização: A aplicação web conseguiu se manter sincronizada com o estado do ESP32, mesmo em caso de reconexão, garantindo uma experiência de usuário consistente.

3.2 Desafios e Soluções

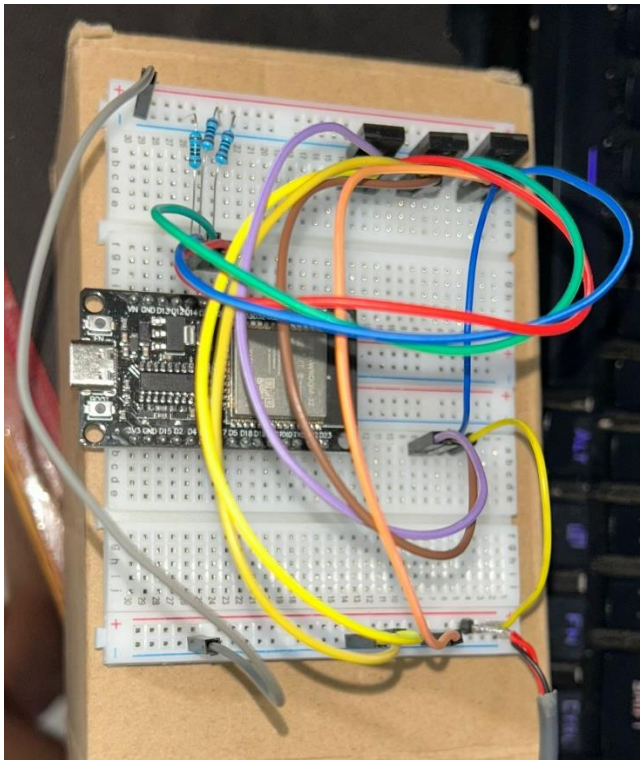
- Alimentação única e controle PWM:
  - Tive problema em achar o controlador de tensão para alimentar com uma única fonte tanto o ESP32 quanto a fita LED, uma vez que o ESP32 suporta até 12V, enquanto a fita LED necessita de no mínimo 24V para ligar. Dessa forma, a solução adotada foi fazer de forma independente a alimentação do ESP e alimentação da fita de LED.
- Queda da conexão MQTT
  - Para evitar que o sistema fique fora do ar e não tenha uma forma automática de reconexão desenvolvi no código a função *reconnectMQTT()*. A função *reconnectMQTT()* é essencial para garantir que o ESP32 se reconecte ao broker MQTT caso a conexão seja perdida (por falha de rede, reinício do broker, etc.).
  - Porque essa implementação é importante?
    - Resiliência:
      - Garante que o ESP32 tente se reconectar automaticamente em caso de quedas do broker ou Wi-Fi.
    - Evita Travamentos:
      - O while tem um timeout (5 segundos), impedindo loops infinitos.
    - Feedback Útil:
      - Logs no Serial Monitor ajudam a diagnosticar problemas (ex: erro rc=-2 indica falha na rede).

Imagem 02: Código do reconnectMQTT()

```
void reconnectMQTT() {
  unsigned long startAttemptTime = millis();

  while (!client.connected() && millis() - startAttemptTime < MQTT_TIMEOUT_MS) {
    Serial.print("Conectando ao MQTT...");

    if (client.connect("ESP32RGBClient")) {
      Serial.println("Conectado!");
      client.subscribe(mqtt_topic);
      shouldPublishStatus = true; // Publicar status ao reconectar
    } else {
      Serial.print("Falhou, rc=");
      Serial.print(client.state());
      Serial.println(" Tentando novamente em 5 segundos");
      delay(5000);
    }
  }

  if (!client.connected()) {
    Serial.println("Timeout na conexão MQTT");
  }
}
```

3.3 Imagens do Projeto

Imagem 01: Imagem das conexões no ESP e na protoboard



- Três transistores são usados como chaves para controlar a corrente do LED. A lógica de controle do ESP32 (3.3V) é aplicada à base do transistor, que por sua vez controla a corrente de 24V que flui para o LED.

- Três resistores de 1kΩ (faixas de cor: marrom, preto, vermelho e dourado) são utilizados para limitar a corrente da base dos transistores, protegendo os pinos do ESP32.
- Fonte de Alimentação de 24V: Fornece a energia necessária para o LED RGB, que não pode ser alimentado diretamente pelo ESP32.
- Protoboards e Jumpers: Utilizados para a montagem do circuito.

- Descrição da Montagem:
  1. O terminal positivo (ânodo comum) do LED RGB é conectado ao terminal positivo da fonte de 24V.
  2. O terminal negativo (catodo) de cada cor do LED (R, G, B) é conectado ao Coletor de um transistor NPN.
  3. O Emissor de cada transistor é conectado ao terra (GND) da protoboard, que por sua vez está conectado ao GND do ESP32 e ao GND da fonte de 24V.
  4. Cada um dos pinos GPIO 14, 27 e 26 do ESP32 é conectado, através de um resistor de 1kΩ, à Base de um dos três transistores.

- Descrição do Desenvolvimento
  1. O firmware foi desenvolvido utilizando o Arduino IDE e bibliotecas C++.
     - Bibliotecas e APIs:
       - *WiFi.h*: API padrão para gerenciamento de conexões Wi-Fi.
       - *PubSubClient.h*: Biblioteca para comunicação com o broker MQTT.
       - *ArduinoJson.h*: Biblioteca para manipulação de mensagens no formato JSON, o que permite o envio de comandos mais complexos (como brilho e sequências personalizadas).
     - Comunicação MQTT:
       - Tópico de Comando (carlos/rgb_pwm): O ESP32 assina este tópico para receber mensagens da aplicação web. As mensagens podem ser strings simples ("off", "fast_colors") ou objetos JSON para configurações mais avançadas.
       - Tópico de Status (carlos/rgb_pwm/status): O ESP32 publica neste tópico para informar seu estado atual (modo, brilho), permitindo que a aplicação web e outros dispositivos possam se manter atualizados.
       - Controle de Saídas (PWM): O código utiliza a função analogWrite() para aplicar a modulação por largura de pulso (PWM) nos pinos do ESP32, o que permite controlar a intensidade de cada cor e, consequentemente, misturar cores e ajustar o brilho geral.

  2. Aplicação Web: O frontend foi desenvolvido com tecnologias web padrão, sem a necessidade de um servidor backend dedicado.

     - Tecnologias:

- HTML5, CSS3, JavaScript: Para a estrutura, estilo e lógica da interface de usuário.
- Paho MQTT (JavaScript): Biblioteca cliente que permite a comunicação direta entre o navegador e o broker MQTT. A comunicação é realizada através de WebSockets, permitindo a conexão com o broker HiveMQ na porta 8000.
- APIs e Protocolos: A aplicação utiliza o protocolo MQTT para:
    - Publicar comandos: O JavaScript envia mensagens para o tópico carlos/rgb_pwm quando o usuário interage com os botões ou sliders.
    - Assinar o tópico de status: A aplicação se inscreve no tópico carlos/rgb_pwm/status para receber feedback do ESP32 em tempo real e atualizar a interface de forma dinâmica.

Imagem 02: Imagem da aplicação web, mostrando o status atual do sistema

Imagem 03: Imagem da aplicação web, mostrando as opções possíveis

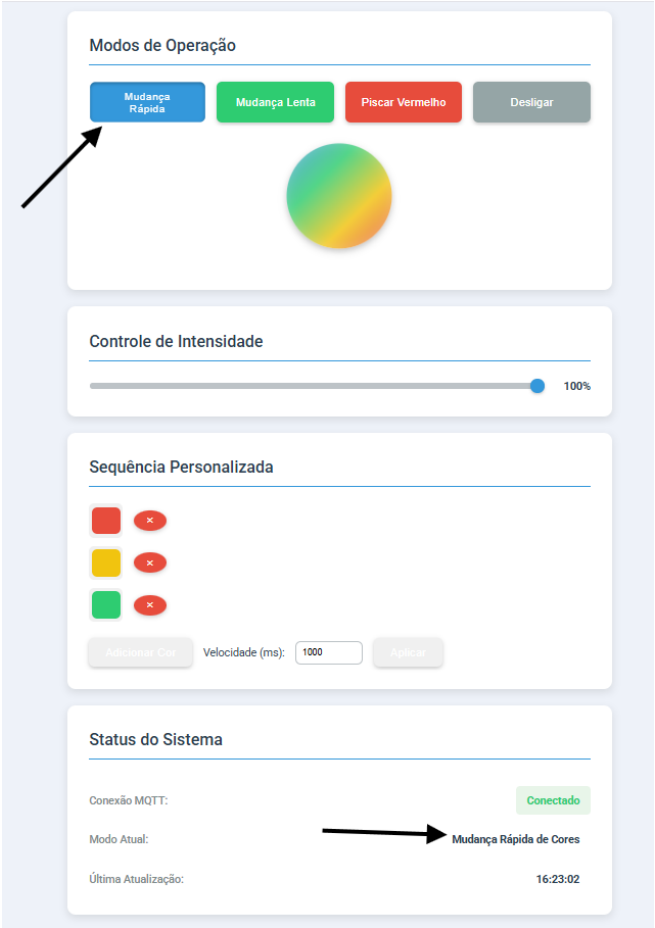


Imagem 04: Imagem da aplicação web, mostrando a sequência personalizada e o status do sistema

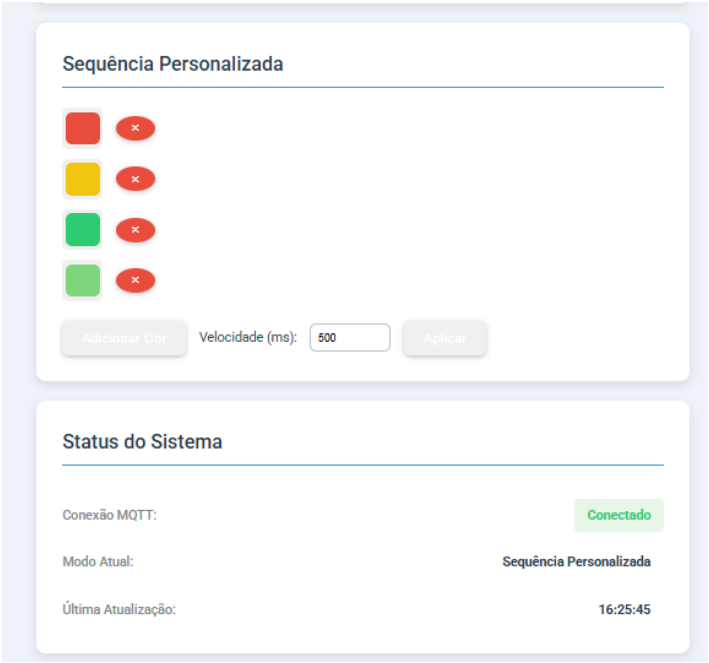


Imagem 05: Mensagens recebidas no Serial Monitor do Arduino IDE

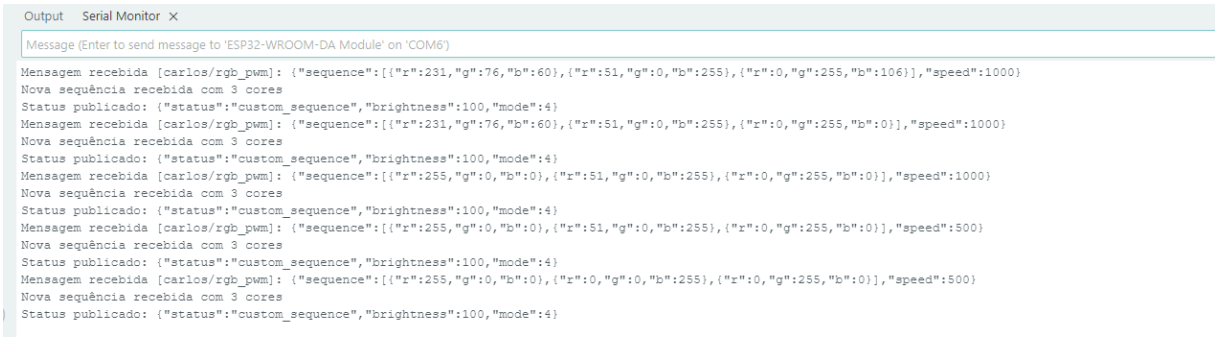


Imagem 06: Mensagens recebidas no Serial Monitor do Arduino IDE

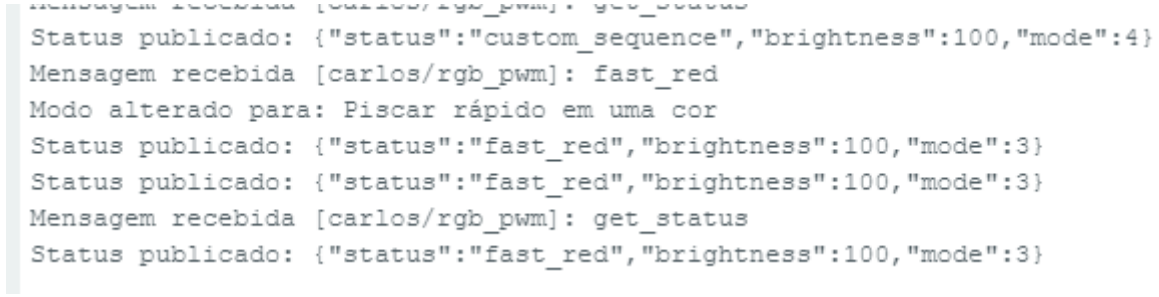
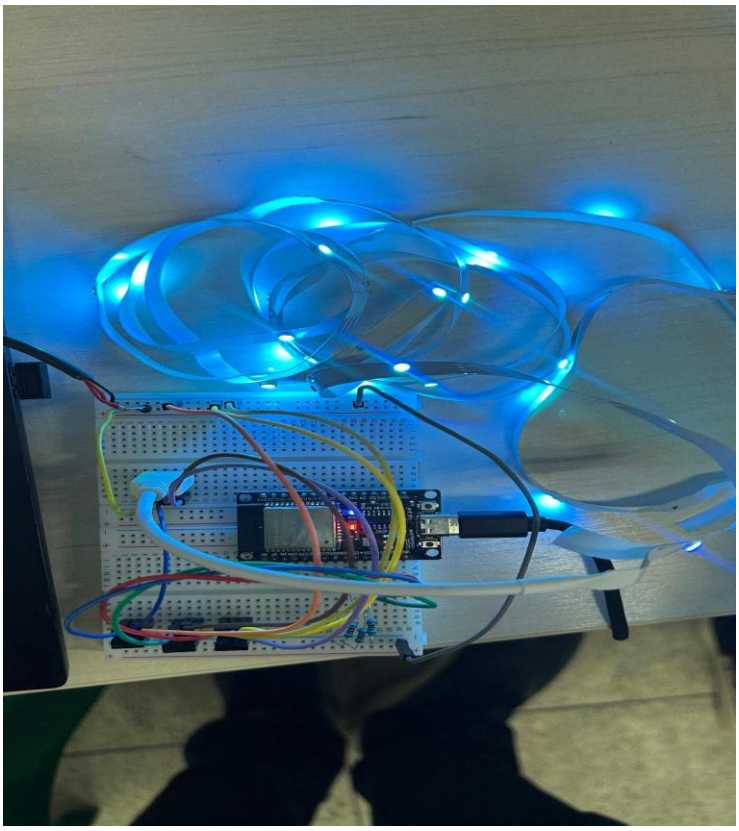


Imagem 07: Imagem do projeto em execução, com o LED ligado



4. Conclusão

O projeto demonstrou a viabilidade de construir um sistema completo e funcional com componentes de baixo custo. A utilização do ESP32 como base, combinada com a arquitetura de mensagens do MQTT, resultou em uma solução robusta, flexível e escalável. O projeto pode ser expandido facilmente para controlar múltiplos LEDs ou outros dispositivos, além de integrar-se a outros sistemas de automação residencial. A comunicação bidirecional e assíncrona, mediada pelo broker MQTT, provou ser uma solução eficaz para o controle em tempo real, eliminando os problemas de latência e travamento frequentemente associados a abordagens de requisição e resposta síncronas.

O sucesso do projeto não se limita apenas ao funcionamento da automação da iluminação. Ele exemplifica a aplicação de conceitos de engenharia de software e hardware, como a temporização não bloqueante no firmware do ESP32 para garantir a responsividade do dispositivo, e a manipulação de dados em formato JSON para comandos complexos. A escolha de tecnologias leves e flexíveis, como o JavaScript no frontend com a biblioteca Paho MQTT, valida a premissa de que soluções de IoT podem ser desenvolvidas com ferramentas acessíveis e sem a necessidade de uma infraestrutura de backend onerosa.

A escalabilidade é um dos pontos fortes desta arquitetura. O sistema pode ser facilmente expandido para controlar múltiplos dispositivos, monitorar sensores ou integrar-se a plataformas de automação residencial mais amplas, bastando adicionar novos tópicos e lógicas de processamento no ESP32.