Análisis Detallado del Firmware: Control de LED RGB con MQTT

Laura Daniela Alarcón Castaño (Autor del Proyecto)

Plataforma: ESP32-C6 (ESP-IDF v5.5)
Archivo Principal: `station_example_main.c`

Índice

1. Introducción y Visión General del Firmware

El archivo `station_example_main.c` constituye el código principal de la aplicación. Su objetivo es transformar el microcontrolador **ESP32-C6** en un cliente IoT que se conecta a Wi-Fi, controla un LED RGB (WS2812) y publica su estado en tiempo real a través de **MQTT** a un Broker remoto (AWS EC2).

2. Configuración del Entorno y Dependencias

2.1. Configuración de Componentes y Versiones

- **Target del Chip:** `esp32c6`.
- **Versión del Framework:** `IDF version: 5.5.0`.
- **Componente Externo Clave:** `espressif/led_strip` versión `2.5.5`. Este componente es esencial para manejar el protocolo de comunicación de los LEDs WS2812 (basado en RMT).

2.2. Distribución de Memoria (Flasheo)

La configuración de flasheo utiliza el cargador de arranque (`bootloader`), la tabla de particiones (`partition-table`) y el binario de la aplicación principal (`mqtt-server.bin`).

Cuadro 1: Mapa de Distribución de Memoria Flash (Extraído de `flasher_args.json`)

Offset	Archivo Binario	Propósito
0x0	bootloader/bootloader.bin	Código de inicio del sistema.
0x8000	partition_table/partition-table.bin	Define las regiones de memoria (NVS, App).
0x10000	mqtt-server.bin	Binario de la aplicación.

3. Análisis de Concurrencia y FreeRTOS

El sistema operativo en tiempo real (RTOS) FreeRTOS gestiona las tareas del sistema.

3.1. Gestión de Tareas (Tasks)

La lógica principal se ejecuta en una tarea de FreeRTOS para asegurar un comportamiento predecible y no bloquear la pila de red.

Cuadro 2: Tareas Principales del Firmware

Tarea	Función Principal	Mecanismo de Retardo
app_main	Inicializa todo el sistema (NVS, WiFi, MQTT, Tareas).	N/A
led_strip_task	Lógica central: Ciclo de colores y publicación MQTT.	`vTaskDelay` (3000 ms)
WiFi/MQTT (SDK)	Manejo de protocolos y comunicaciones de red.	Ejecución asíncrona por FreeRTOS.

3.2. Sincronización

El uso de `vTaskDelay` (‘vTaskDelay(DELAY_COLOR_MS / portTICK_PERIOD_MS)‘) garantiza que la tarea cede el control al *scheduler* de FreeRTOS, permitiendo que las tareas de la pila de red se ejecuten de manera eficiente entre cambios de color.

4. Control del Periférico (LED RGB WS2812)

4.1. API de Control del LED

El control del LED RGB se realiza mediante el driver **RMT (Remote Control)**, que garantiza la precisión en la temporización del protocolo WS2812.

- **Pin de Control:** `GPIO 8`.
- **Función Clave:** `led_strip_refresh`, que es la encargada de enviar físicamente los datos de color al LED.

4.2. Función de Paleta de Colores

La función `led_set_color` aplica los valores RGB correspondientes a cada color:

- **Pink (Rosa):** (255, 20, 147)
- **Purple (Púrpura):** (128, 0, 128)
- **Blue (Azul):** (0, 0, 255)

```
static void led_set_color(const char* color) {
    if (strcmp(color, "pink") == 0) {
        // Rosa (Pink): R=255, G=20, B=147
        ESP_ERROR_CHECK(led_strip_set_pixel(led_strip, 0, 255, 20, 147));
    } else if (strcmp(color, "purple") == 0) {
        // P rpura (Purple): R=128, G=0, B=128
        ESP_ERROR_CHECK(led_strip_set_pixel(led_strip, 0, 128, 0, 128));
    } else if (strcmp(color, "blue") == 0) {
        // Azul (Blue): R=0, G=0, B=255
        ESP_ERROR_CHECK(led_strip_set_pixel(led_strip, 0, 0, 255, 0));
    }
    ESP_ERROR_CHECK(led_strip_refresh(led_strip));
}
```

Listing 1: Función led_set_color (Extracto con valores RGB)

5. Conectividad y Protocolo MQTT

5.1. Flujo de Conexión

1. **NVS Init:** Inicialización del almacenamiento no volátil (NVS) para guardar credenciales Wi-Fi. 2. **Wi-Fi Init:** Configuración del chip como estación (STA) y espera del evento ‘IP$_{E}VENT_{S}TA_{G}OT_{I}P$‘,3.**MQTT Start:**$Conexiónalbrokerdefinidoen‘BROKER_{U}RI‘(protocolopordefecto$

5.2. API del Cliente MQTT

El cliente utiliza la API de Espressif `esp-mqtt` y publica en el tópico `.esp32/led"` con un **QoS 0** (At most once), priorizando la baja latencia sobre la garantía de entrega.

6. Lógica Central de la Aplicación (‘led$_{s}trip_{t}ask$‘)

La tarea principal ejecuta el ciclo de estados de la aplicación.

6.1. Bucle de Estados

```
void led_strip_task(void *pvParameters)
{
    // ... Espera a que MQTT est  conectado ...
    while (1) {
        // 1. Ciclo Rosa (Pink)
        led_set_color("pink");
        esp_mqtt_client_publish(mqtt_client, PUBLISH_TOPIC, "pink", 0, 0, 0);
        vTaskDelay(DELAY_COLOR_MS / portTICK_PERIOD_MS);

        // 2. Ciclo P rpura (Purple)
        led_set_color("purple");
        esp_mqtt_client_publish(mqtt_client, PUBLISH_TOPIC, "purple", 0, 0, 0);
        vTaskDelay(DELAY_COLOR_MS / portTICK_PERIOD_MS);

        // 3. Ciclo Azul (Blue)
        led_set_color("blue");
        esp_mqtt_client_publish(mqtt_client, PUBLISH_TOPIC, "blue", 0, 0, 0);
        vTaskDelay(DELAY_COLOR_MS / portTICK_PERIOD_MS);
    }
}
```

Listing 2: Bucle de la Tarea led_strip_task

6.2. Diagrama de Flujo del Firmware

El flujo de la tarea principal es secuencial y cíclico:

7. Pruebas y Troubleshooting

7.1. Verificación y Debugging

La operación correcta se confirma buscando los siguientes mensajes de registro (logs):

- **Conectividad:** ‘WIFI: GOT IP‘ y ‘MQTT$_{E}VENT_{C}ONNECTED$‘.
- **Lógica:** ‘Color publicado: [pink/purple/blue]‘ cíclico cada 3 segundos.

7.2. Diagnóstico de Fallos Comunes

Cuadro 3: Diagnóstico de Problemas de Firmware y Conectividad

Síntoma	Causa Raíz Típica	Solución Sugerida
No hay conexión WiFi.	Credenciales incorrectas o problema de configuración de la estación.	Revisar las constantes de SSID/-Password en la configuración de `menuconfig` o en las macros.

Cuadro 3: Diagnóstico de Problemas de Firmware y Conectividad

Síntoma	**Causa Raíz Típica**	**Solución Sugerida**
El LED no se enciende/cambia.	'LED_STRIP_GPIO' incorrecto o error en la inicialización RMT.	Confirmar el GPIO correcto (`GPIO 8`) para el WS2812 en la placa ESP32-C6. Revisar logs por errores RMT.
No hay mensajes MQTT en el servidor.	El firewall (Security Group) de AWS está bloqueando el puerto 1883.	Abrir el puerto `1883` TCP en las reglas de seguridad de la instancia EC2.