Singapore University of Technology and Design

30.201 Wireless Communications and Internet of Things

Spotify Mood Maestro

IOT Project

Authors:	*Student ID:*
Adeline Chew Li Lin	1005309
Akhil Jayadeep	1005399

Table of Contents

1 Introduction

In the era of the Internet of Things (IoT), the integration of smart devices and cloud services has transformed how we interact with our surroundings. This project aims to create a robust and innovative IoT system that seamlessly combines Spotify music integration, RFID-based user identification, and dynamic mood lighting control.

The primary objective is to develop a smart automation system that provides a personalized and immersive music-listening experience. By leveraging the ESP32-C6 microcontroller, we have designed a distributed architecture that allows efficient communication, data processing, and remote control of the lighting system.

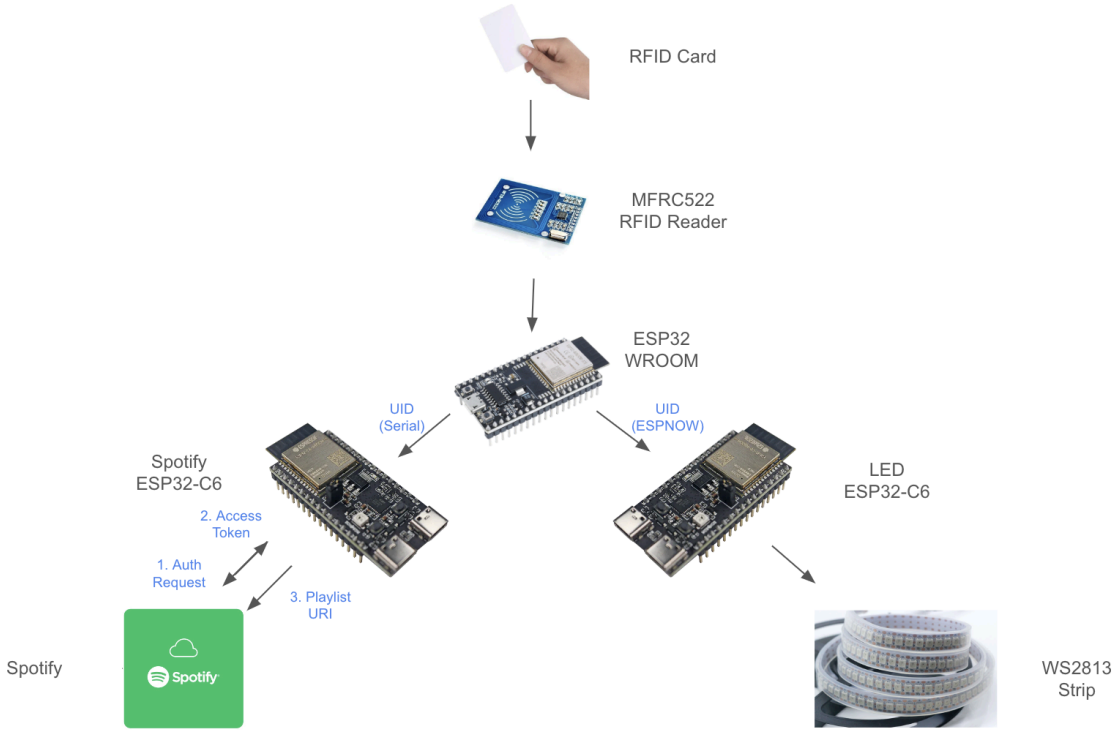
The integration of Spotify, a leading cloud-based music service, enables users to control their music playback directly. When an RFID tag is detected, the system identifies the playlist and adjusts the mood lighting to complement the music, creating a unique and personalized ambient environment.

This project aims to bring the nostalgic experience of physical media, such as vinyl records, but with a modern twist. By seamlessly integrating Spotify, RFID, and mood lighting, we strive to reduce the time users spend being distracted by phone apps and streaming service UIs, allowing them to focus on the music and its accompanying atmosphere.

The distributed nature of the system, utilizing multiple ESP32-C6 boards, enhances the overall reliability and scalability of the project. Robust communication protocols ensure reliable data transfer and remote control capabilities and prevent chances of complete system failure by separating the functionalities.

In this report, we will provide a comprehensive overview of the Spotify Mood Maestro, detailing the system architecture, hardware implementation, software development, and innovative features that make it a compelling IoT solution.

2 System Architecture



The Spotify Mood Maestro project employs a distributed architecture, utilizing three distinct ESP32-C6 microcontroller boards to handle different functionalities:

Spotify ESP32-C6:

This board is responsible for integrating with the Spotify API and controlling the music playback. It receives user input from the tapped RFID card. The Spotify ESP32-C6 board is connected to the RFID ESP32 board via a serial communication link to receive the UID of the tapped card.

RFID ESP32:

This board is connected to an RFID reader that detects user interactions, such as tapping an RFID tag or card. When an RFID tag is detected, the RFID ESP32 board retrieves the unique RFID UID (Unique Identifier) and transmits it serially to the Spotify ESP32 and wirelessly to the LED ESP32-C6 board using the ESP-NOW protocol. The RFID ESP32 board serves as the intermediary between the user input (RFID tag) and the playlist and mood lighting control.

LED ESP32-C6:

This board is responsible for controlling the LED strip, which serves as the mood lighting element. The LED ESP32-C6 board receives the RFID UID data from the RFID ESP32-C6 board via ESP-NOW and uses it to determine the appropriate lighting effects to be displayed.

By separating the LED control from the other functionalities, the system maintains modularity and simplifies the overall implementation.

Communication Protocols

The Spotify Mood Maestro project utilizes two primary communication protocols to enable data transfer between the various ESP32-C6 boards:

1. **Serial Communication:**

   The Spotify ESP32-C6 board and the RFID ESP32-C6 board are connected using a serial communication link. Serial communication is a widely-used protocol in embedded systems, allowing for reliable and low-latency data transfer between two directly connected devices.

   In this project, the serial communication is used to transmit the user's Spotify playback commands from the Spotify ESP32-C6 board to the RFID ESP32-C6 board. This direct serial connection ensures that the Spotify control commands are delivered efficiently, without the overhead of a wireless protocol.

2. **ESP-NOW:**

   The RFID ESP32-C6 board uses the ESP-NOW protocol to communicate with the LED ESP32-C6 board. ESP-NOW is a proprietary Wi-Fi-based direct communication protocol developed by Espressif, the manufacturer of the ESP32-C6 microcontroller. Unlike traditional Wi-Fi connections that require an access point or router, ESP-NOW enables direct peer-to-peer communication between ESP32-C6 boards. This wireless protocol is well-suited for IoT applications, as it provides a secure and efficient way to exchange data without the overhead of a full Wi-Fi network.

In the Spotify Mood Maestro project, the RFID ESP32 board uses ESP-NOW to transmit the RFID UID data to the LED ESP32-C6 board, which then uses this information to control the mood lighting.

3 Hardware Implementation

The Spotify Mood Maestro project utilizes the following hardware components:

Component	Quantity	Remarks
ESP32-C6 Board	2	For Spotify and LED Control
ESP32-WROOM-32D Board	1	For RFID Reader
RFID-RC522 Module	1	-
RFID Tags/Cards	6 Cards + 1 Tag	-
WS2813 LED Strip	1m Strip (60 leds)	-

Step by Step Guide on Implementation:

Spotify Control Board (ESP32-C6):

Connect the ESP32-C6 board to your development computer using a USB cable. Ensure that the necessary drivers and the ESP-IDF development framework are installed on your computer.

LED Control Board (ESP32-C6):

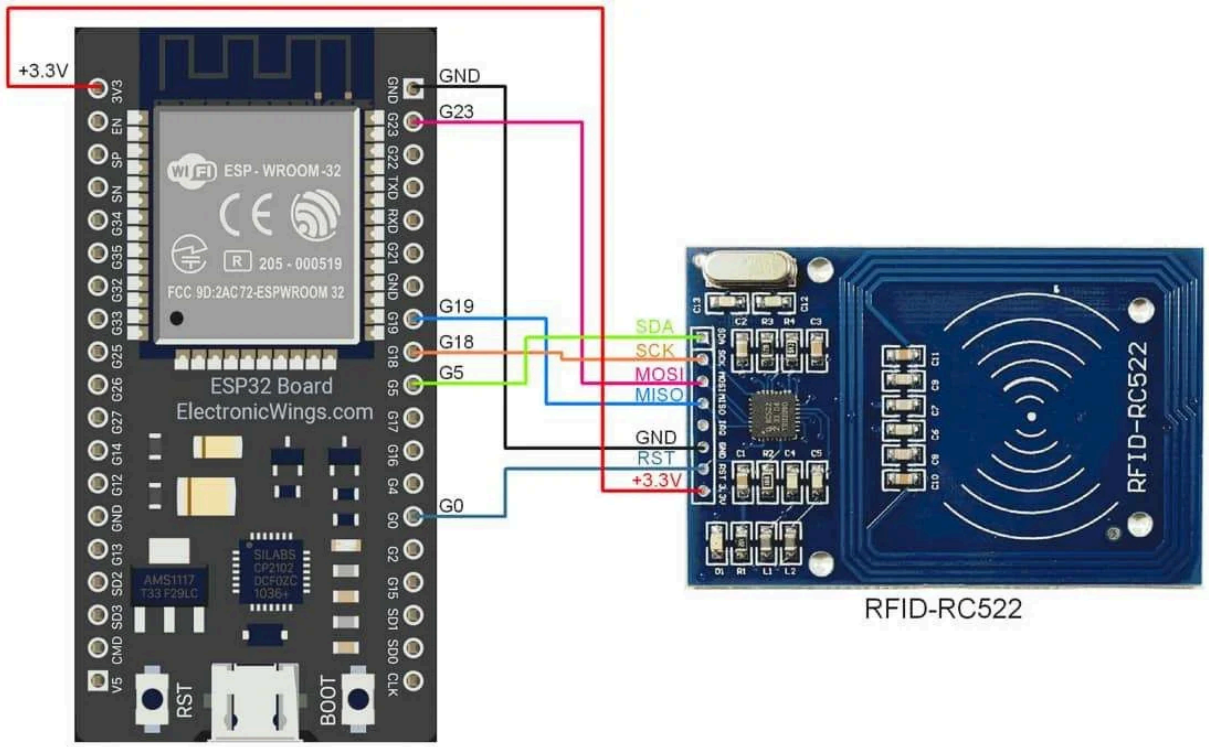
Connect the second ESP32-C6 board to your development computer using a USB cable. Ensure that the necessary drivers and the ESP-IDF development framework are installed on your computer.

Connect the WS2813 LED strip to the LED Control Board (ESP32-C6). Connect the Power and Ground lines to 5V and GND pins of the ESP32-C6. Connect both D0 and B0 data lines of the LED strip to GPIO pin 0.

RFID Reader Board (ESP32-WROOM-32D):

Connect the ESP32-WROOM-32D board to your development computer using a USB cable.
Ensure that the necessary drivers and the ESP-IDF development framework are installed on your computer.

Connect the RFID-RC522 module to the ESP32-WROOM-32D board using the following pin connections:



Connect the RX1and TX1 pins (16 & 17) of the ESP32-WROOM-32D to the UART pins defined for the Spotify ESP32-C6 (TX - 4, RX - 5).

After completing the hardware setup, you can proceed with the software implementation and integration of the various components to create the Spotify Mood Maestro system.

4 Software Implementation

The code has been uploaded to this Github repository:
https://github.com/alto934/esp-idf-spotify-mood-maestro.git

Prior to starting the software implementation you will have to create a Spotify account (preferably with the Premium Plan). Next in order to use the Web API for the project, follow the steps detailed in this page: Spotify Web API Access. Obtain the Client ID and Client Secret.

ESP-IDF Framework Integration

The project utilizes ESP-IDF to develop the application for the ESP32-C6 microcontroller. The use of ESP-IDF provides several benefits:

1. **FreeRTOS Integration:** The code leverages the FreeRTOS real-time operating system, which is integrated into the ESP-IDF. This allows the creation of multiple tasks, efficient task management, and inter-task communication.
2. **Networking Capabilities:** The project uses the esp_netif and esp_wifi components to handle the Wi-Fi connection and network-related functionality, such as making HTTP requests to the Spotify API.
3. **Event Management:** The esp_event module is used to handle various events, such as HTTP responses, which simplifies the implementation of event-driven programming.
4. **Logging and Debugging:** The esp_log component provides a flexible and customizable logging system, which is essential for debugging and monitoring the application's behavior.
5. **HTTP Client and Server:** The project utilizes the esp_http_client and esp_http_server components to handle HTTP communication with the Spotify API and potentially serve a web interface.
6. **RMT (Remote Control) Module:** The project utilizes the driver/rmt_tx module to control the LED strip, taking advantage of the RMT peripheral's high-resolution timing capabilities.

RFID ESP32 (Arduino Code)

This ESP32 runs is the only one that runs Arduino code. The reason behind is that there is no reliable port of the MFRC522 RFID library for ESP-IDF. As a result we decided to use the Arduino Library simply for the RFID reader.

1. **RFID Module Initialization:**
   The code initializes the MFRC522 RFID module by setting the Slave Select (SS) and Reset (RST) pins. The MFRC522 library is used to interact with the RFID reader. The MIFARE_Key structure is initialized with all-0xFF values, as this is a requirement for the MFRC522 library.

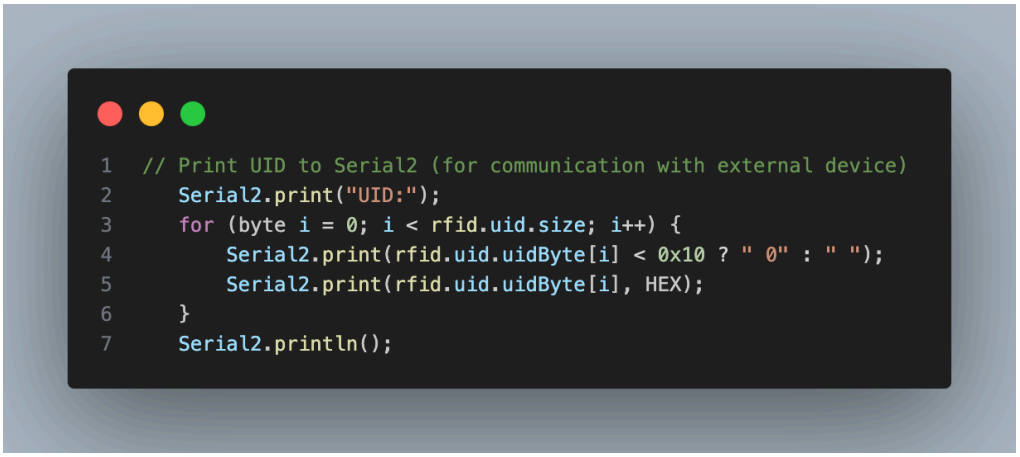
```
#define SS_PIN 5
#define RST_PIN 0
#define RXp2 16 // RX pin for Serial2
#define TXp2 17 // TX pin for Serial2

MFRC522 rfid(SS_PIN, RST_PIN); // Instance of the MFRC522 class
MFRC522::MIFARE_Key key;

// ESP-NOW peer address
uint8_t peer_mac[6] = {0x40, 0x4C, 0xCA, 0x51, 0x3A, 0xB0};

void setup() {
  Serial.begin(115200);
  Serial2.begin(115200, SERIAL_8N1, RXp2, TXp2); // Initialize Serial2 communication

  SPI.begin();
  rfid.PCD_Init();
  Serial.println(F("RFID reading via Serial2."));

  for (byte i = 0; i < 6; i++) {
    key.keyByte[i] = 0xFF;
  }

```

2. **Serial Communication:**
   The code sets up the Serial2 communication with a baud rate of 115200 bps, using the appropriate RX and TX pins (RXp2 and TXp2). This serial communication channel is used to transmit the RFID UID data to an external device, in this case, the Spotify control board.

3. **ESP-NOW Initialization and Peer Configuration:**
   The code initializes the ESP-NOW protocol by setting the Wi-Fi mode to Station (WIFI_STA). It then configures the ESP-NOW peer address, which corresponds to the MAC address of the LED control board. The ESP-NOW peer is added to enable wireless communication between the RFID board and the LED control board.

```
// Initialize and configure ESP-NOW
  WiFi.mode(WIFI_STA);
  if (esp_now_init() != ESP_OK) {
    Serial.println("Error initializing ESP-NOW");
    return;
  }

  // Add the peer to ESP-NOW
  esp_now_peer_info_t peer_info = {};
  memcpy(peer_info.peer_addr, peer_mac, 6);
  if (esp_now_add_peer(&peer_info) != ESP_OK) {
    Serial.println("Failed to add peer");
    return;
  }
}
```

4. **RFID Tag/Card Detection and UID Transmission:**
   In the main loop, the code checks if a new RFID card or tag is present using the PICC_IsNewCardPresent() and PICC_ReadCardSerial() functions. If a new card is detected, the UID of the card is printed to the Serial port for logging purposes. The UID is then formatted into a string with spaces between each byte and sent to the LED control board using the esp_now_send() function. After transmitting the UID, the PICC (Proximity Integrated Circuit Card) is halted, and the crypto1 encryption on the PCD (Proximity Coupling Device) is stopped.



Spotify ESP32-C6 (IDF)

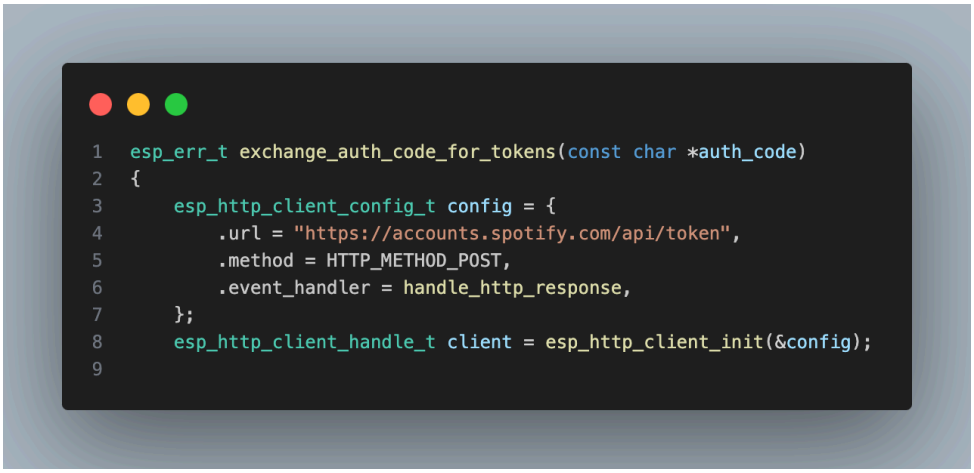
Spotify API Integration:

The core functionality of the project is the integration with the Spotify Web API, which allows the application to control music playback on a Spotify-enabled device. The key aspects of this integration are:

1. **Authorization Flow:** The application initiates the OAuth 2.0 authorization flow to obtain an access token from Spotify. This involves making an HTTP GET request to the Spotify authorization endpoint and handling the response. For this, you will have to specify a redirect URL for the Spotify API. In our case the URL is the IP address of the ESP32-C6. The ESP will start a web server which will read the returned authorization code from the redirect URL. The authorization code will then be used for token exchange.

```
ESP_LOGI(TAG, "Authorization URL: %s", url);
 // Perform HTTP GET request to authorization URL
 esp_err_t err = http_get_request(url);
 if (err != ESP_OK)
 {
   ESP_LOGE(TAG, "Failed to request authorization");
 }
}
```

2. **Token Exchange:** After obtaining the authorization code, the application makes an HTTP POST request to the Spotify token endpoint to exchange the authorization code for an access token and a refresh token.



```
esp_err_t exchange_auth_code_for_tokens(const char *auth_code)
{
    esp_http_client_config_t config = {
        .url = "https://accounts.spotify.com/api/token",
        .method = HTTP_METHOD_POST,
        .event_handler = handle_http_response,
    };
    esp_http_client_handle_t client = esp_http_client_init(&config);

```

3. **Spotify Device ID Lookup:** The application retrieves the device ID of the target Spotify device by sending an HTTP GET request to the "devices" endpoint and parsing the response.

```c
// Function to get Spotify device ID of a specific device by name
esp_err_t get_spotify_device_id(const char *access_token, const char* target_device_name) {
    ESP_LOGI(TAG, "Getting list of Spotify devices for device name: %s", target_device_name);

    const char *devices_url = "https://api.spotify.com/v1/me/player/devices";
    char auth_header[300];
    snprintf(auth_header, sizeof(auth_header), "Bearer %s", access_token);

    esp_http_client_config_t config = {
        .url = devices_url,
        .event_handler = handle_http_response,
    };
    esp_http_client_handle_t client = esp_http_client_init(&config);
```

4. **Spotify Playback Control:** The application uses the Spotify Web API to start playback of a specific Spotify URI on the previously retrieved device. It constructs an HTTP PUT request to the "play" endpoint with the necessary parameters, such as the device ID and the Spotify URI.

```c
static const char *TAG3 = "SPOTIFY_PLAY";

#define SPOTIFY_API_BASE_URL "https://api.spotify.com/v1/me/player/play"

esp_err_t play_spotify_album(const char *access_token, const char *device_id, const char *album_uri) {
    esp_http_client_config_t config = {
        .url = SPOTIFY_API_BASE_URL,
        .method = HTTP_METHOD_PUT,
        .event_handler = NULL,
        .user_data = NULL,
        .timeout_ms = 10000,  // Adjust the timeout value as needed
    };

    esp_http_client_handle_t client = esp_http_client_init(&config);
```

The integration with the Spotify API is crucial for the project, as it allows the application to control the music playback based on the RFID input received through the UART interface.

RFID Data Processing and Transmission:

The project processes the RFID tag data received through the UART interface and uses it to control the playback of Spotify content. The key aspects of this functionality are:

UART Receiver Task: The application creates a separate FreeRTOS task that listens for incoming UART data. This task receives the RFID tag IDs and passes them to the play_spotify_content_by_uid() function for further processing.

```c
static void rx_task(void *arg) {
    uint8_t data[BUF_SIZE];
    int length = 0;
    uart_event_t event;
    static uint8_t uid[5] = {0}; // Buffer to store the received UID
    static uint8_t uid_index = 0;

    for (;;) {
        if (xQueueReceive(uart_queue, (void *)&event, portMAX_DELAY)) {
            switch (event.type) {
                case UART_DATA:
                    length = uart_read_bytes(UART_NUM, data, event.size, portMAX_DELAY);
                    data[length] = 0; // Null-terminate the received data

                    // Process the received data to extract the UID
                    uint8_t *ptr = (uint8_t *)strstr((char *)data, "UID:");
                    if (ptr != NULL) {
                        ptr += 4; // Skip past "UID:"
                        uid_index = 0;
                        while (*ptr) {
                            if (isspace(*ptr)) {
                                ptr++; // Skip whitespace characters
                            } else if (isxdigit(*ptr)) {
                                char hex_byte[3] = {ptr[0], ptr[1], '\0'};
                                uid[uid_index++] = (uint8_t)strtoul(hex_byte, NULL, 16);
                                ptr += 3; // Skip past the hex byte and the space
                            } else {
                                break; // Stop processing if an unexpected character is encountered
                            }
                            if (uid_index >= sizeof(uid)) {
                                ESP_LOGW(TAG2, "UID buffer overflow");
                                break;
                            }
                        }
                        uid[uid_index] = 0; // Null-terminate the UID
                        print_uid(uid); // Print the UID for debugging
                        play_spotify_content_by_uid(uid); // Play Spotify content based on UID
                    }
                    break;
                default:
                    break;
            }
        }
    }
```

RFID-based Spotify Playback: The play_spotify_content_by_uid() function maps the received RFID tag IDs to specific Spotify content, such as albums or playlists, and initiates the playback using the play_spotify_album() function.

```c
void play_spotify_content_by_uid(const uint8_t *uid) {
    ESP_LOGI(TAG2, "First byte of UID: 0x%02X", uid[0]);
    // Assuming each UID corresponds to a unique Spotify URI
    // simple switch case based on the first byte of the UID
    switch(uid[0]) {
        case 0x33:
            play_spotify_album(access_token, saved_device_id, "spotify:album:4SZko61aMnmgvNhfhgTuD3");
```

The processing of RFID data and its integration with the Spotify playback control is a key feature of the project, as it enables the user to interact with the system using RFID tags and experience seamless music playback.

LED ESP32-C6 (IDF)

LED Strip Control:

1. **RMT Channel Configuration:** The application creates an RMT TX channel with the specified GPIO pin, memory block size, and resolution. This channel is used to transmit the LED strip data.

```
ESP_LOGI(TAG, "Create RMT TX channel");
    rmt_channel_handle_t led_chan = NULL;
    rmt_tx_channel_config_t tx_chan_config = {
        .clk_src = RMT_CLK_SRC_DEFAULT,
        .gpio_num = RMT_LED_STRIP_GPIO_NUM,
        .mem_block_symbols = 64,
        .resolution_hz = RMT_LED_STRIP_RESOLUTION_HZ,
        .trans_queue_depth = 4,
    };
    ESP_ERROR_CHECK(rmt_new_tx_channel(&tx_chan_config, &led_chan));

    ESP_LOGI(TAG, "Enable RMT TX channel");
    ESP_ERROR_CHECK(rmt_enable(led_chan));
```

2. **LED Strip Pixel Initialization:** The led_strip_pixels array is initialized with all pixels set to off (zero values for R, G, and B).

```
// Initialize the LED strip with all pixels off
    memset(led_strip_pixels, 0, sizeof(led_strip_pixels));
    ESP_ERROR_CHECK(rmt_transmit(led_chan, led_encoder, led_strip_pixels, sizeof(led_strip_pixels), &tx_config));
    ESP_ERROR_CHECK(rmt_tx_wait_all_done(led_chan, portMAX_DELAY));

```

3. **Mood Color Definitions:** The application defines an array of mood_color_t structures, which represent different color schemes that can be used for the LED strip. These colors are mapped to the received RFID tag IDs.

```
static const mood_color_t mood_colors[] = {
    {148, 0, 211},     // Graduation Purple
    {224, 255, 255},       // light cyan
    {0, 0, 255},       // dark blue
    {255, 69, 0},       // dark orange
    {255, 0, 0},      // red
    {0, 146, 0},     // Shrek Green
    {252, 3, 148}       // Pink
};
```

4. **LED Strip Fade Task:** The led_strip_fade_task() function is executed in a separate FreeRTOS task. This task is responsible for the following:
    a. Waiting for a new RFID tag ID to be received (new_uid_received flag).
    b. Mapping the received UID to a mood color index based on the first byte of the UID.
    c. Initializing the fade phase variables, including the fade start time and fade duration.
    d. Updating the LED strip pixels with the current mood color and a fading effect using a sine wave function.
    e. Transmitting the updated pixel data to the LED strip using the RMT channel.
    f. Checking for a new UID reception and breaking out of the continuous fade loop if a new UID is received.
    g. ESP-NOW Receive Callback: The espnow_receive_cb() function is registered as the callback for receiving ESP-NOW messages. When a message is received, the function extracts the RFID tag UID from the message and updates the received_uid variable and the new_uid_received flag.

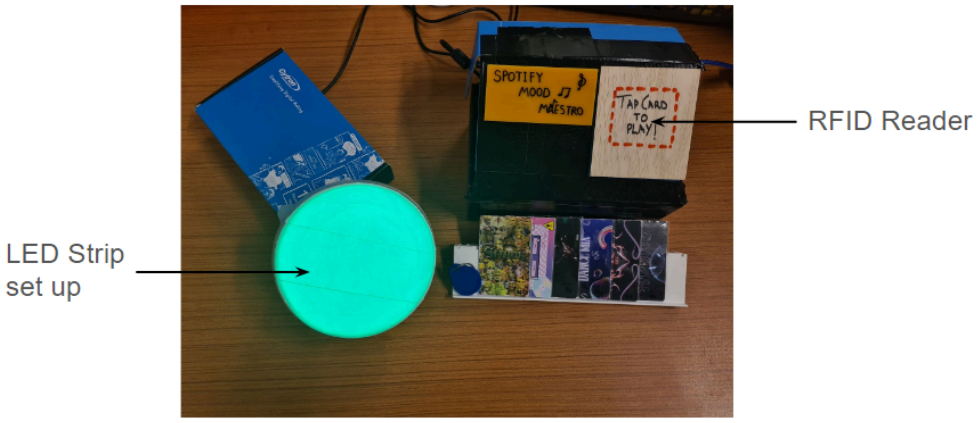
```
while (1) {
        // Wait for a new UID to be received
        while (!new_uid_received) {
            vTaskDelay(1); // Delay for 1ms
        }

        new_uid_received = false;

        // Print the first byte of the received UID
        ESP_LOGI(TAG, "First byte of received UID: 0x%02X", received_uid[0]);

        // Map the received UID to a mood color index based on the first byte
        switch (received_uid[0]) {
            case 0x33:
                current_mood_color_index = 0; // graduation purple
                break;
            case 0x93:
                current_mood_color_index = 1; // Utopia white
                break;
            case 0x8B:
                current_mood_color_index = 2; // sad light blue
                break;
            case 0x76:
```

The app_main() function is the entry point of the application. It performs the following tasks:

1. Creates the RMT TX channel for the LED strip.
2. Enables the RMT TX channel.
3. Prints the device's MAC address.
4. Initializes the NVS (Non-Volatile Storage) component.
5. Initializes the Wi-Fi in Station mode.
6. Initializes the ESP-NOW component and registers the receive callback.
7. Creates the led_strip_fade_task() as a separate FreeRTOS task.

5 Final Prototype

Github Repo: https://github.com/alto934/esp-idf-spotify-mood-maestro.git



Overall Setup: Mood Light Setup can be placed anywhere in the room within a radius of 200 m of the RFID Reader

RFID cards corresponding to different mood playlists and albums

A video of the prototype demonstration can be viewed here:
https://youtu.be/KNbntrkW-wc?si=nF_m_3VwhZ6bQx4v

6 Conclusion

This project has successfully integrated the ESP-IDF framework, Spotify Web API, RFID data processing, and LED strip control to create an interactive IoT system for hands-free music playback and visualization.

The key achievements include seamless Spotify integration, RFID-based playback control, and immersive LED strip effects. The modular and extensible software architecture allows for future enhancements, such as multi-user support, dedicated play/pause buttons along with volume control, wireless RFID data transmission, expanded lighting effects, and integration with smart home ecosystems.

***Project Allocation*:**

1. Akhil Jayadeep - Spotify API integration into ESP32-C6, RFID data processing and transmission using ESP32 WROOM.
2. Adeline Chew - LED strip integration into ESP32-C6, hardware prototyping of components.

7 Appendix

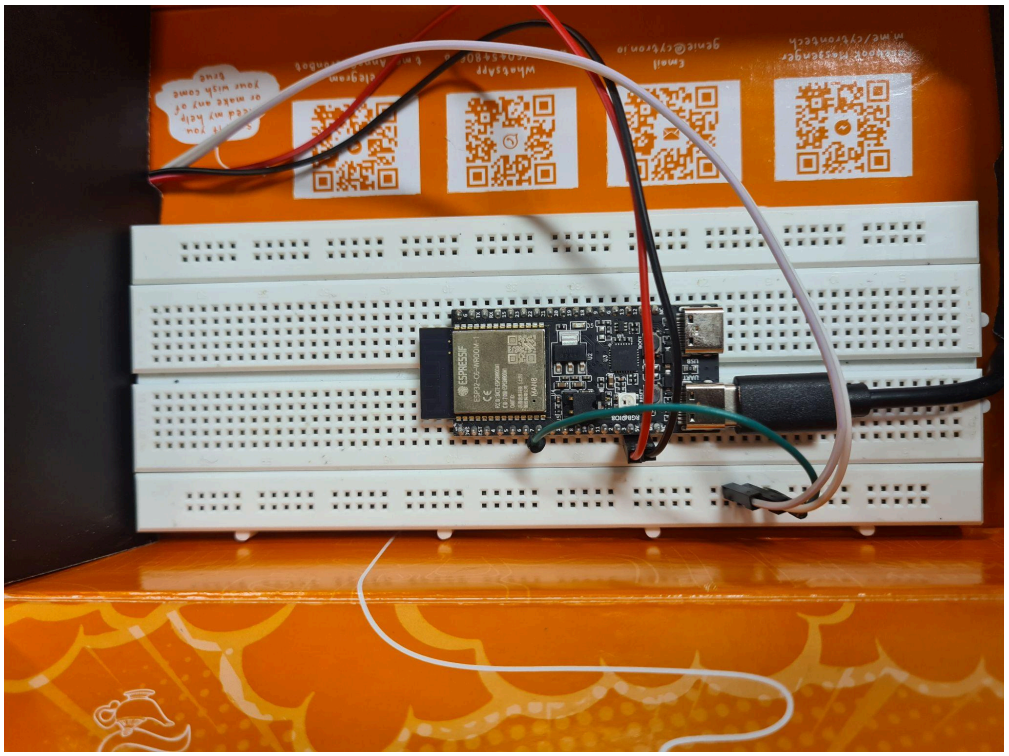


Fig. LED ESP32-C6

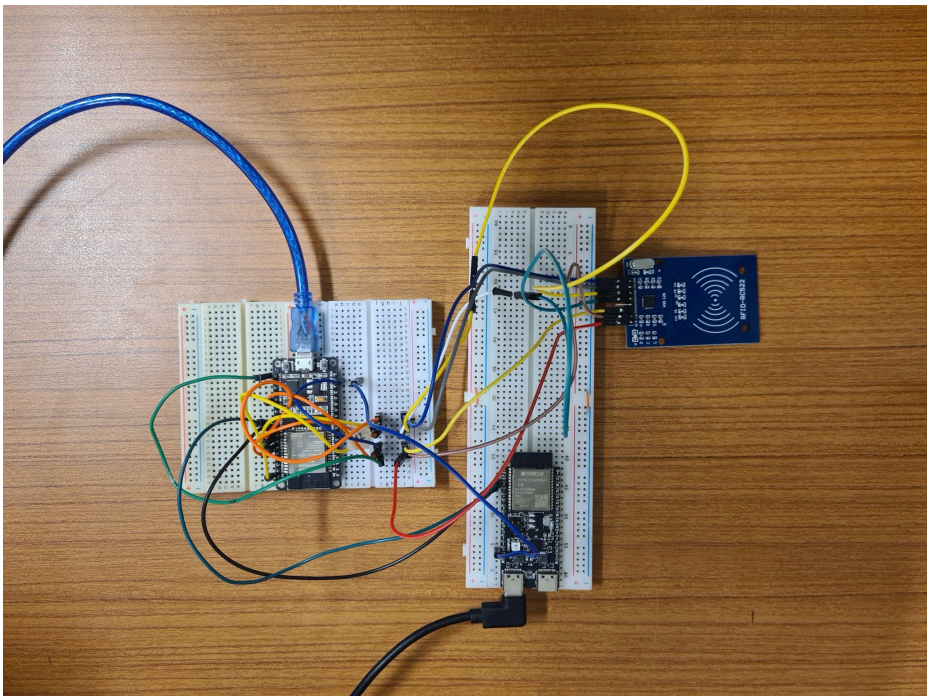


Fig. Spotify ESP32-C6 + RFID ESP32