ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH

TRƯỜNG ĐẠI HỌC CÔNG NGHỆ THÔNG TIN

KHOA KỸ THUẬT MÁY TÍNH

BÁO CÁO CUỐI KỲ

MÔN THIẾT KẾ HỆ THỐNG NHÚNG KHÔNG DÂY

CE232.O21

HỌ VÀ TÊN: **Vòng Chí Cường – 21521910**

Trần Văn Cường – 21521909

LỚP: **CE232.O21.2**

GIẢNG VIÊN HƯỚNG DẪN:

Trần Hoàng Lộc

TP. HỒ CHÍ MINH – Tháng 6 năm 2024

MỤC LỤC

MỤC LỤC HÌNH ẢNH

1. GIỚI THIỆU CHUNG

1.1. Tìm hiểu tổng quan

Với những tiến bộ công nghệ hiện nay, sự kết nối là lĩnh vực chính rất được quan tâm. Từ điện thoại đến các thiết bị điện tử khác, kết nối không dây đã trở thành 1 phần không thể thiếu trong kỷ nguyên IoT này. Trong đó, hệ thống âm thanh không dây là 1 công cụ hữu ích và đi kèm những hạn chế nhất định. Hiện nay, loa không dây hỗ trợ Bluetooth đã tràn ngập trên thị trường công nghệ, tuy nhiên chúng có 1 số hạn chế như: không hỗ trợ kết nối nhiều thiết bị, độ trễ do sự cố kết nối và độ phủ sóng nhỏ (dưới 20m).

Chúng tôi nảy ra ý tưởng để kết nối nhiều thiết bị âm thanh cùng lúc bằng cách sử dụng công nghệ không dây WiFi có độ phủ sóng rộng hơn rất nhiều và có thể xử lý truyền dữ liệu trên nhiều thiết bị mà không có bất kì độ trễ nào. Hệ thống cho phép người dùng truy cập từ bất kì đâu trên thế giới chỉ cần có kết nối Internet. Hệ thống được đặt tên là IoT Speaker.

1.2. Mục tiêu, giới hạn đề tài

Mục tiêu:

- Số lần thực hiện đúng chức năng: > 90%
- Độ phản hồi của Loa: < 2 giây
- Độ phản hồi của App: < 2 giây
- Độ trễ khi hẹn giờ dừng phát nhạc: < 2 giây

Giới hạn đề tài:

- Mức âm lượng: 1 → 10.
- Đóng gói sản phẩm: < 20x20cm^2

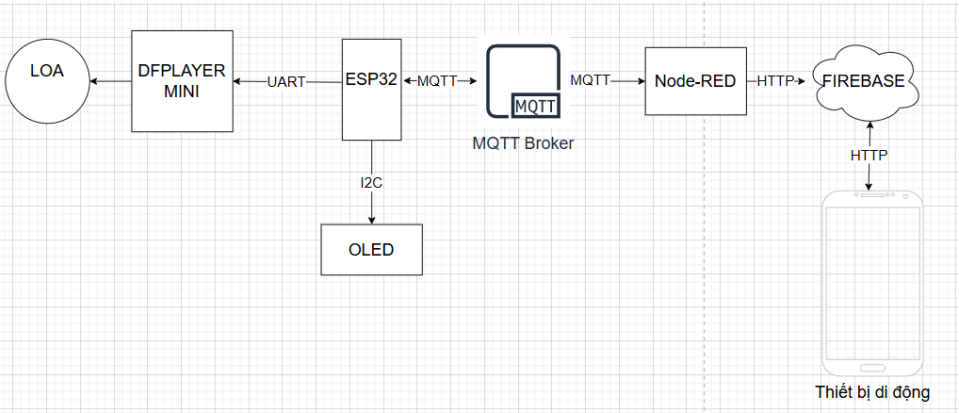
2. KIẾN TRÚC HỆ THỐNG

2.1. Tổng quan hệ thống

Hệ thống IoT Speaker gồm 3 phần chính : bộ xử lý âm thanh, bộ xử lý trung tâm và ứng dụng di động. Bộ xử lý trung tâm sử dụng ESP32 có chức năng nhận, xử lý dữ liệu và gửi tín hiệu điều khiển cho bộ xử lý âm thanh và tín hiệu thành công/thất bại cho ứng dụng di động. ESP32 có kết nối WiFi giao tiếp với ứng dụng di động thông qua Firebase RealTime Database và MQTT (Message Queuing Telemetry Transport). Khi ứng dụng gửi tín hiệu điều khiển lên Firebase, sử dụng Node-RED để lấy dữ liệu vừa thay đổi gửi vào MQTT-Broker với topic cụ thể. ESP32 đăng kí topic tương ứng và sẽ nhận được dữ liệu. ESP32 xử lý dữ liệu vừa nhận được và gửi tín hiệu điều khiển đến bộ xử lý âm thanh. Sau đó, ESP32 sẽ gửi tín hiệu thành công/thất bại lên MQQT-Broker với topic cụ thể và Node-RED sẽ gửi tín hiệu đó lên lại Firebase. Khi ứng dụng di động nhận được tín hiệu thành công/thất bại thì sẽ hiển thị tương ứng ra màn hình.

Bộ xử lý âm thanh sử dụng module DFPLayer Mini và loa có dây để thực hiện những chức năng như phát nhạc, dừng nhạc, tăng, giảm âm lượng… Khi nhận được tín hiệu từ bộ xử lý trung tâm thì DFPlayer Mini sẽ thực hiện chức năng tương ứng và sử dụng loa để phát ra âm thanh.

Ứng dụng di động sử dụng nền tảng Flutter để tạo ra ứng dụng đa nền tảng rất tiện lợi giúp hiển thị, thao tác các chức năng như chọn bài, bài nhạc đang phát, hẹn giờ…



Hình 1 – Sơ đồ thiết kế hệ thống

2.2. Mô hình giao tiếp hệ thống

2.2.1. WIFI

2.2.1.1. Khái niệm

Wi-Fi là một họ các giao thức mạng không dây, dựa trên các tiêu chuẩn của họ IEEE 802.11, được sử dụng rộng rãi trong cho việc kết nối không dây của thiết bị trong mạng nội bộ và việc kết nối Internet cho phép các thiết bị điện tử trong phạm vi ngắn chia sẻ dữ liệu thông qua sóng vô tuyến.

Generation	IEEE standard	Adopted	Maximum link rate (Mbit/s)	Radio frequency (GHz)
Wi-Fi 8	802.11bn	2028[44]	100,000[45]	2.4, 5, 6, 7, 42.5, 71[46]
Wi-Fi 7	802.11be	2024	1376–46,120	2.4, 5, 6[47]
Wi-Fi 6E	802.11ax	2020	574–9608[48]	6[b]
Wi-Fi 6		2019		2.4, 5
Wi-Fi 5	802.11ac	2014	433–6933	5[c]
Wi-Fi 4	802.11n	2008	72–600	2.4, 5
(Wi-Fi 3)*	802.11g	2003	6–54	2.4
(Wi-Fi 2)*	802.11a	1999		5
(Wi-Fi 1)*	802.11b	1999	1–11	2.4
(Wi-Fi 0)*	802.11	1997	1–2	2.4

2.2.1.2. *Cấu* hình WiFi

Hàm wifi_init_sta :

```
void wifi_init_sta(void)
```

```
{

    // 1. Wi-Fi/LwIP Init Phase
    ESP_ERROR_CHECK(esp_netif_init());
    ESP_ERROR_CHECK(esp_event_loop_create_default());
    esp_netif_create_default_wifi_sta();

    wifi_init_config_t cfg = WIFI_INIT_CONFIG_DEFAULT();
    ESP_ERROR_CHECK(esp_wifi_init(&cfg));

    // 2. Wi-Fi Configuration Phase
    esp_event_handler_instance_t instance_any_id;
    esp_event_handler_instance_t instance_got_ip;
    ESP_ERROR_CHECK(esp_event_handler_instance_register(WIFI_EVENT
,
                                                        ESP_EVENT_ANY_ID,
                                                        &wifi_event_handler,
                                                        NULL,
                                                        &instance_any_id));
    ESP_ERROR_CHECK(esp_event_handler_instance_register(IP_EVENT,
                                                        IP_EVENT_STA_GOT_IP,
                                                        &wifi_event_handler,
                                                        NULL,
                                                        &instance_got_ip));

    wifi_config_t wifi_config = {
        .sta = {
            .ssid = EXAMPLE_ESP_WIFI_SSID,
            .password = EXAMPLE_ESP_WIFI_PASS,
        },
    };
```

```
    ESP_ERROR_CHECK(esp_wifi_set_mode(WIFI_MODE_STA));
    ESP_ERROR_CHECK(esp_wifi_set_config(WIFI_IF_STA,
&wifi_config));

    // 3. Wi-Fi Start Phase
    ESP_ERROR_CHECK(esp_wifi_start());
    ESP_LOGI(TAG, "wifi_init_sta finished.");
}
```

1. ***Wi-Fi/LwIP Init Phase***

s1.1: Tác vụ chính gọi hàm esp_netif_init() để tạo một tác vụ lõi LwIP và khởi tạo công việc liên quan đến LwIP.

s1.2: Tác vụ chính gọi hàm esp_event_loop_create() để tạo một tác vụ sự kiện hệ thống và khởi tạo hàm gọi lại sự kiện của ứng dụng. Trong kịch bản trên, hàm gọi lại sự kiện của ứng dụng không làm gì ngoài việc chuyển tiếp sự kiện đến tác vụ ứng dụng.

s1.3: Tác vụ chính gọi hàm esp_netif_create_default_wifi_ap() hoặc esp_netif_create_default_wifi_sta() để tạo một instance giao diện mạng mặc định liên kết với station hoặc AP cùng với ngăn xếp TCP/IP.

s1.4: Tác vụ chính gọi hàm esp_wifi_init() để tạo tác vụ driver Wi-Fi và khởi tạo driver Wi-Fi.

s1.5: Tác vụ chính gọi API hệ điều hành để tạo tác vụ ứng dụng.

2. ***Wi-Fi Configuration Phase***

Khi driver Wi-Fi đã được khởi tạo, bạn có thể bắt đầu cấu hình driver Wi-Fi. Trong kịch bản này, chế độ là station, vì vậy bạn có thể cần gọi hàm esp_wifi_set_mode() (WIFI_MODE_STA) để cấu hình chế độ Wi-Fi là station.

3. Wi-Fi Start Phase

3.1: Gọi hàm esp_wifi_start() để khởi động driver Wi-Fi.

s3.2: Driver Wi-Fi gửi sự kiện WIFI_EVENT_STA_START đến tác vụ sự kiện; sau đó, tác vụ sự kiện sẽ thực hiện một số công việc chung và gọi hàm gọi lại sự kiện của ứng dụng.

s3.3: Hàm gọi lại sự kiện của ứng dụng chuyển tiếp sự kiện WIFI_EVENT_STA_START đến tác vụ ứng dụng.

Hàm wifi_event_handler :

```
void wifi_event_handler(void *arg, esp_event_base_t event_base,
                        int32_t event_id, void *event_data)
{
    // 3. Wi-Fi Start Phase
    if (event_base == WIFI_EVENT && event_id ==
WIFI_EVENT_STA_START)
    {
        // 4. Wi-Fi Connect Phase
        esp_wifi_connect();
    }

    // 6. Wi-Fi Disconnect Phase
    else if (event_base == WIFI_EVENT && event_id ==
WIFI_EVENT_STA_DISCONNECTED)
    {
        if (s_retry_num < EXAMPLE_ESP_MAXIMUM_RETRY)
        {
            esp_wifi_connect();
```

```
            s_retry_num++;
            ESP_LOGI(TAG, "retry to connect to the AP");
        }

        ESP_LOGI(TAG, "connect to the AP fail");
    }

    // 5. Wi-Fi 'Got IP' Phase
    else if (event_base == IP_EVENT && event_id ==
IP_EVENT_STA_GOT_IP)
    {
        ip_event_got_ip_t *event = (ip_event_got_ip_t *)event_data;
        ESP_LOGI(TAG, "got ip:" IPSTR, IP2STR(&event->ip_info.ip));
        s_retry_num = 0;
        // xEventGroupSetBits(s_wifi_event_group, WIFI_CONNECTED_BIT);
        mqtt_app_start();
    }
}
```

4. Wi-Fi Connect Phase

s4.1: Khi esp_wifi_connect() được gọi, driver Wi-Fi sẽ bắt đầu quá trình quét/kết nối nội bộ.

s4.2: Nếu quá trình quét/kết nối nội bộ thành công, sự kiện WIFI_EVENT_STA_CONNECTED sẽ được tạo ra. Trong tác vụ sự kiện, nó sẽ khởi động client DHCP, cuối cùng sẽ kích hoạt quá trình DHCP.

s4.3: Trong kịch bản đã đề cập ở trên, hàm gọi lại sự kiện của ứng dụng sẽ chuyển tiếp sự kiện đến tác vụ ứng dụng.

Ở bước 4.2, kết nối Wi-Fi có thể thất bại vì, ví dụ, mật khẩu sai hoặc không tìm thấy AP. Trong trường hợp này, sự kiện WIFI_EVENT_STA_DISCONNECTED sẽ xảy ra và lý do cho sự thất bại sẽ được cung cấp.

5. Wi-Fi 'Got IP' Phase

s5.1: Khi client DHCP được khởi tạo ở bước 4.2, giai đoạn nhận IP sẽ bắt đầu.

s5.2: Nếu địa chỉ IP được nhận thành công từ máy chủ DHCP, sự kiện IP_EVENT_STA_GOT_IP sẽ xảy ra và tác vụ sự kiện sẽ thực hiện các xử lý chung.

s5.3: Trong hàm gọi lại sự kiện của ứng dụng, sự kiện IP_EVENT_STA_GOT_IP được chuyển tiếp đến tác vụ mqtt_app_start() để cấu hình giao thức MQTT.

2.2.2. MQTT

2.2.2.1. Khái niệm

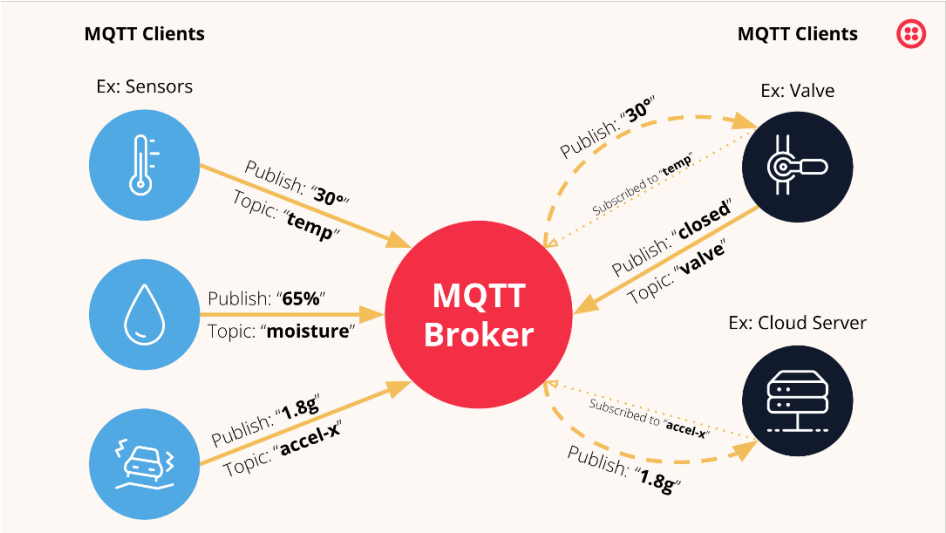
MQTT (Message Queueing Telemetry Transport) là một giao thức mạng kích thước nhỏ (lightweight), hoạt động theo cơ chế publish - subscribe theo tiêu chuẩn ISO (ISO/IEC 20922) và OASIS mở để truyền tin nhắn giữa các thiết bị. Giao thức này hoạt động trên nền tảng TCP/IP; tuy nhiên, bất kỳ giao thức mạng nào cung cấp các kết nối theo tuần tự, không mất dữ liệu (lossless), kết nối hai chiều đều có thể hỗ trợ MQTT. MQTT được thiết kế cho các kết nối cho việc truyền tải dữ liệu cho các thiết bị ở xa, các thiết bị hay vi điều khiển nhỏ có tài nguyên hạn chế, hoặc trong các ứng dụng có băng thông mạng bị hạn chế.

Thành phần:

- Client

- Publisher - Nơi gửi thông điệp (message) lên một/nhiều topic cụ thể.
- Subscriber - Nơi nhận thông điệp (message) từ topic này.

- Broker - Máy chủ môi giới : được coi như trung tâm, nó là điểm giao của tất cả các kết nối đến từ Client (Publisher/Subscriber). Nhiệm vụ chính của Broker là nhận thông điệp (message) từ Publisher, xếp vào hàng đợi rồi chuyển đến một địa điểm cụ thể.



Hình 2 – Các thành phần trong một mạng MQTT

2.2.2.2. Cấu hình MQTT

Cài đặt thông số MQTT trong project:

```
#define EXAMPLE_ESP_MQQT_BORKER_URI "mqtt://mqtt.flespi.io"
#define EXAMPLE_ESP_MQQT_BORKER_PORT 1883
#define EXAMPLE_ESP_MQQT_CREDENTIALS_USERNAME
"oiGjHdBbBIvM0gOgrc0oLFTFt5ev1frmO6r8SOQURW1Gr7qYjFflB5IdeKu
tDcUk"
```

➔ Sử dụng dịch vụ broker của flespi.io, với port 1883 và username như trên.

Cấu hình MQTT trong project:

```
void mqtt_app_start()
{
    esp_mqtt_client_config_t mqtt_cfg = {
        .broker.address.uri = EXAMPLE_ESP_MQQT_BORKER_URI,
        .broker.address.port = EXAMPLE_ESP_MQQT_BORKER_PORT,
        //.broker.address.transport = EXAMPLE_ESP_MQQT_BORKER_TRANSPORT,
        .credentials.username = EXAMPLE_ESP_MQQT_CREDENTIALS_USERNAME,
    };

    global_client = esp_mqtt_client_init(&mqtt_cfg);
    /* The last argument may be used to pass data to the event handler, in this example mqtt_event_handler */
    esp_mqtt_client_register_event(global_client, ESP_EVENT_ANY_ID, mqtt_event_handler, NULL);
    esp_mqtt_client_start(global_client);
}
```

⇨ Cấu hình MQTT, khởi tạo client, đăng ký callback xử lý sự kiện và bắt đầu client như sau:

Tạo một cấu trúc cấu hình esp_mqtt_client_config_t để thiết lập các thông số kết nối đến broker MQTT:

- broker.address.uri: Địa chỉ URI của broker MQTT.
- broker.address.port: Cổng của broker MQTT.

- broker.address.transport: Giao thức vận chuyển (được chú thích trong mã này, không được sử dụng).
- credentials.username: Tên người dùng cho xác thực MQTT.

Khởi tạo client MQTT:

```
global_client = esp_mqtt_client_init(&mqtt_cfg);
```

- Gọi hàm esp_mqtt_client_init(&mqtt_cfg) để khởi tạo client MQTT với cấu hình đã định nghĩa ở trên và lưu client này vào biến toàn cục global_client.

Đăng ký callback sự kiện:

```
esp_mqtt_client_register_event(global_client, ESP_EVENT_ANY_ID, mqtt_event_handler, NULL);
```

Đăng ký một callback sự kiện cho client MQTT. Hàm esp_mqtt_client_register_event được sử dụng để gán tất cả các sự kiện (ESP_EVENT_ANY_ID) đến hàm xử lý sự kiện mqtt_event_handler. Đối số cuối cùng (NULL) có thể được sử dụng để truyền dữ liệu đến handler, nhưng trong trường hợp này không có dữ liệu nào được truyền.

Bắt đầu client MQTT:

```
esp_mqtt_client_start(global_client);
```

Gọi hàm esp_mqtt_client_start(global_client) để bắt đầu kết nối và xử lý dữ liệu của client MQTT.

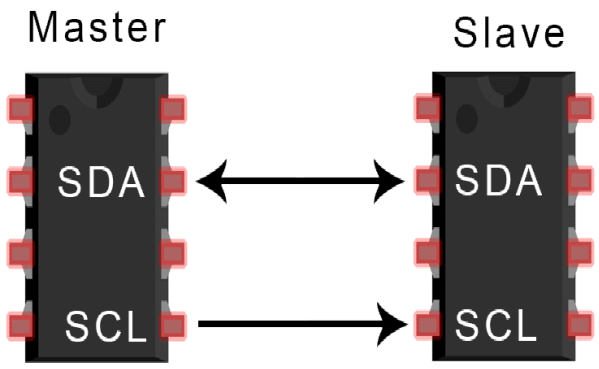
2.2.3. I2C

2.2.3.1. Khái niệm

I2C (viết tắt của từ Inter-Integrated Circuit) là một giao thức thức truyền thông nối tiếp (serial communication protocol) được sử dụng rộng rãi trong trong các hệ thống nhúng. Giao thức này được phát triển bởi Philips Semiconductor (tiền thân của NXP Semiconductors ngày nay) vào năm 1982.

I2C chỉ sử dụng hai dây để truyền dữ liệu giữa các thiết bị:

- SDA (Serial Data) - đường truyền cho master và slave để gửi và nhận dữ liệu.
- SCL (Serial Clock) - đường mang tín hiệu xung nhịp.



Hình 3 - Mô hình giao tiếp I2C

2.2.3.2. Cấu hình I2C

```
#define I2C_MASTER_FREQ_HZ 400000 // I2C clock of SSD1306 can run at 400 kHz max.
#define I2C_TICKS_TO_WAIT 100     // Maximum ticks to wait before issuing a timeout.
```

```
void i2c_master_init(SSD1306_t *dev, int16_t sda, int16_t scl, int16_t reset)
```

```
{
    ESP_LOGI(TAG, "Legacy i2c driver is used");
    i2c_config_t i2c_config = {
        .mode = I2C_MODE_MASTER,
        .sda_io_num = sda,
        .scl_io_num = scl,
        .sda_pullup_en = GPIO_PULLUP_ENABLE,
        .scl_pullup_en = GPIO_PULLUP_ENABLE,
        .master.clk_speed = I2C_MASTER_FREQ_HZ};
    ESP_ERROR_CHECK(i2c_param_config(I2C_NUM, &i2c_config));
    ESP_ERROR_CHECK(i2c_driver_install(I2C_NUM,
I2C_MODE_MASTER, 0, 0, 0));

    if (reset >= 0)
    {
        // gpio_pad_select_gpio(reset);
        gpio_reset_pin(reset);
        gpio_set_direction(reset, GPIO_MODE_OUTPUT);
        gpio_set_level(reset, 0);
        vTaskDelay(50 / portTICK_PERIOD_MS);
        gpio_set_level(reset, 1);
    }
    dev->_address = I2C_ADDRESS;
    dev->_flip = false;
}
```

2.2.4. UART

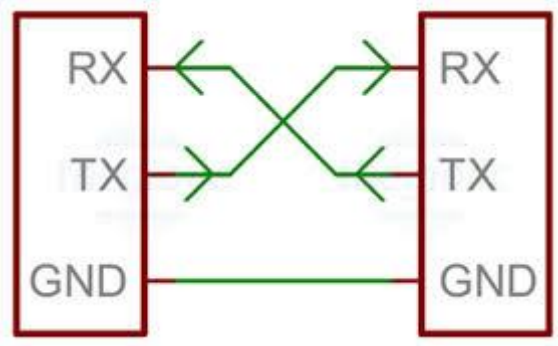
2.2.4.1. Khái niệm

UART là viết tắt của Universal Asynchronous Receiver / Transmitter. UART hoàn toàn khác biệt với chuẩn giao tiếp SPI hoặc I2C, những chuẩn này chỉ đơn

tuần là giao tiếp phần mềm. Mục đích chính của UART là truyền và nhận dữ liệu nối tiếp.

Chuẩn giao tiếp UART sử dụng 2 dây để truyền và nhận dữ liệu giữa các thiết bị:

1. TX (Transmiter) – Dây truyền dữ liệu.
2. RX (Receiver) – Dây nhận dữ liệu.



Hình 4 - Mô hình giao tiếp UART

Số dây sử dụng	2
Tốc độ	Từ 9600 bps -> 115200 bps
Phương thức truyền dữ liệu	Không đồng bộ
Kiểu truyền dữ liệu	Nối tiếp
Số lượng Master (thiết bị chủ)	1
Số lượng Slave (thiết bị tớ)	1

2.2.4.2. Cấu hình UART

```
void init_uart(DFPLAYER_Name *MP3)
```

```
{
    const uart_config_t uart_config = {
        .baud_rate = 9600,
        .data_bits = UART_DATA_8_BITS,
        .parity = UART_PARITY_DISABLE,
        .stop_bits = UART_STOP_BITS_1,
        .flow_ctrl = UART_HW_FLOWCTRL_DISABLE,
        .source_clk = UART_SCLK_APB,
    };
    ESP_ERROR_CHECK(uart_driver_install(MP3->DFP_UART,
RX_BUF_SIZE * 2, 0, 0, NULL, 0));
    ESP_ERROR_CHECK(uart_param_config(MP3->DFP_UART,
&uart_config));
    ESP_ERROR_CHECK(uart_set_pin(MP3->DFP_UART, TXD_PIN,
RXD_PIN, UART_PIN_NO_CHANGE, UART_PIN_NO_CHANGE));
}
```

1. **Cấu hình UART**:

Tạo một cấu trúc cấu hình uart_config_t để thiết lập các thông số cho giao tiếp UART:

- baud_rate: Tốc độ baud là 9600.
- data_bits: Số bit dữ liệu là 8 bit.
- parity: Tắt kiểm tra chẵn lẻ (parity).
- stop_bits: Số bit dừng là 1 bit.
- flow_ctrl: Tắt điều khiển luồng bằng phần cứng.
- source_clk: Sử dụng xung nhịp nguồn từ APB (Advanced Peripheral Bus).

2. **Cài đặt driver UART:**

```
    ESP_ERROR_CHECK(uart_driver_install(MP3->DFP_UART,
RX_BUF_SIZE * 2, 0, 0, NULL, 0));
```

Gọi hàm uart_driver_install() để cài đặt driver UART:

- MP3->DFP_UART: Cổng UART được sử dụng cho DFPlayer.
- RX_BUF_SIZE * 2: Kích thước bộ đệm nhận dữ liệu (gấp đôi kích thước bộ đệm nhận).
- 0: Không sử dụng bộ đệm gửi dữ liệu.
- 0: Không sử dụng bộ đệm nhận dữ liệu.
- NULL: Không sử dụng hàng đợi (queue) sự kiện.
- 0: Độ sâu hàng đợi sự kiện là 0.

3. **Cấu hình các thông số UART:**

```
   ESP_ERROR_CHECK(uart_param_config(MP3->DFP_UART,
&uart_config));
```

Gọi hàm uart_param_config() để cấu hình các thông số cho cổng UART với cấu hình đã định nghĩa ở trên.

4. **Thiết lập các chân UART:**

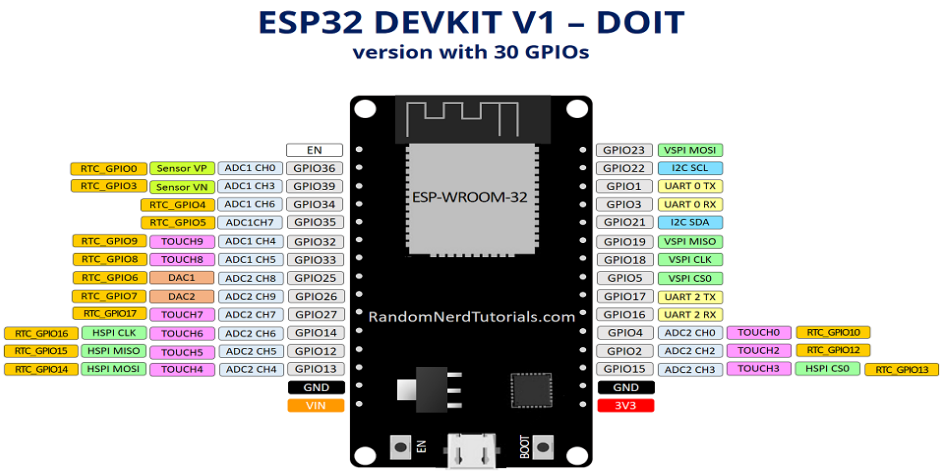
```
   ESP_ERROR_CHECK(uart_set_pin(MP3->DFP_UART, TXD_PIN,
RXD_PIN, UART_PIN_NO_CHANGE, UART_PIN_NO_CHANGE));
```

Gọi hàm uart_set_pin() để thiết lập các chân (pin) cho cổng UART:

- MP3->DFP_UART: Cổng UART được sử dụng cho DFPlayer.
- TXD_PIN: Chân truyền dữ liệu (TXD).
- RXD_PIN: Chân nhận dữ liệu (RXD).
- UART_PIN_NO_CHANGE: Không thay đổi chân RTS (Request to Send).
- UART_PIN_NO_CHANGE: Không thay đổi chân CTS (Clear to Send).

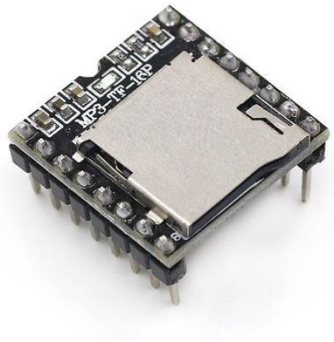
2.3. Hardware

- Vi điều khiển ESP32 – WROOM 32



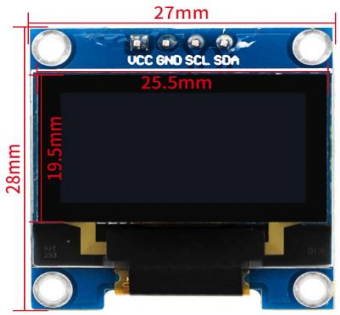
Hình 5 - ESP32 Devkit V1

- Mạch Phát Âm Thanh MP3 Kết Hợp Amply DFPlayer Mini : phát âm thanh MP3 qua thẻ nhớ MicroSD giao tiếp với vi điều khiển qua giao tiếp UART



Hình 6 - DFplayer Mini

- Màn hình LCD SSD1306: Hiển thị thông báo ra màn hình.



Hình 7 - Màn hình LCD SSD1306

- ***Loa :*** phát âm thanh ra bên ngoài.

Hình 8 - Loa

2.4. Software

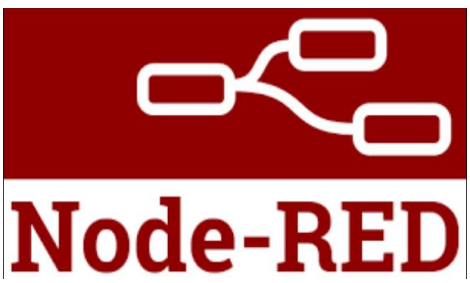
- ***Espressif-IDF:*** là framework hỗ trợ để lập trình trên ESP32 giúp tối ưu hóa chức năng hơn so với dùng Arduino.

Hình 9 - Espressif IDF

- ***Flespi:*** là MQQT Broker được lưu trữ dưới dạng cloud giúp người dùng có thể public hoặc subcribe dữ liệu bằng giao thức MQTT.

Hình 10 - Flespi

- ***Node-RED:*** là công cụ dùng để thực hiện tất cả chức năng từ việc subcribe hay public dữ liệu từ MQTT Broker và cập nhật dữ liệu đó lên database.



Hình 11 - Node-red

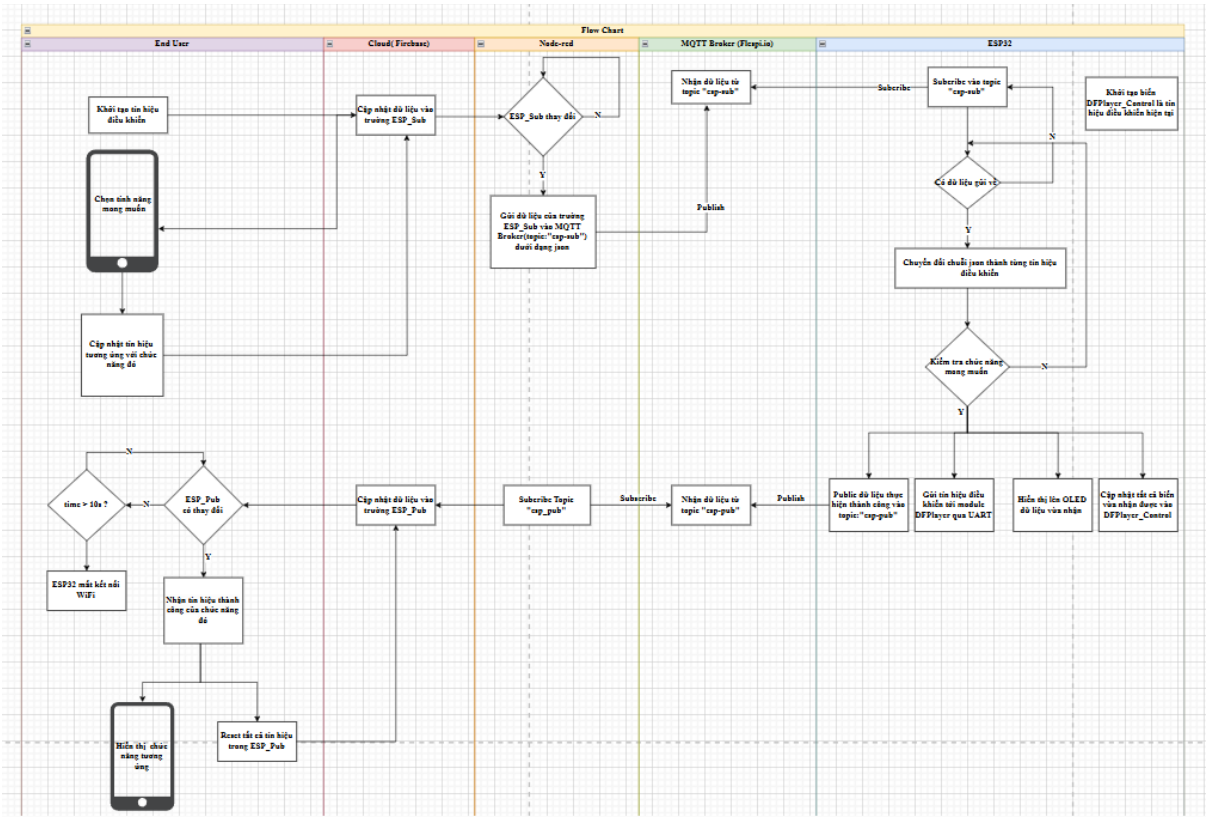
- ***Firebase:*** là 1 database lưu trữ dưới dạng cloud giúp người dùng có thể lưu trữ dữ liệu trên đó, có thể dùng để thao tác thêm, sửa, xóa dữ liệu.

Hình 12 - Firebase

- ***Flutter:*** là 1 nền tảng để phát triển phần mềm đa nền tảng giúp người dùng có thể trực quan hóa dữ liệu lên màn hình.

Hình 13 - Flutter

2.5. Sơ đồ thiết kế phần mềm:



Hình 14 - Sơ đồ thiết kế phần mềm

3. THỰC NGHIỆM VÀ ĐÁNH GIÁ:

3.1. Kịch bản kiểm thử và kết quả:

3.1.1. Chức năng kết nối WiFi:

Trường hợp 1:

- Input: ESP32 có kết nối WiFi, loa đang phát bài số 3, ứng dụng thực hiện thao tác Next.
- Output: Loa phát bài số 4, màn hình LCD, ứng dụng hiển thị trạng thái bài số 4, màn hình LCD hiển thị tình trạng đang kết nối WiFi.

Trường hợp 2:

- Input: Loa đang phát bài sô 4, ngắt kết nối WiFi kết nối tới ESP32, ứng dụng thực hiện thao tác Play.
- Output: màn hình LCD hiển thị trạng thái mất kết nối, sau 10s mất kết nối ứng dụng hiển thị thông báo "ESP32 WiFi Lost Connection" và quay trở về trang chủ.

3.1.2. Chức năng hẹn giờ:

Trường hợp 1: Hẹn giờ dừng phát nhạc trong 30 giây

- Input: Đặt thời gian dừng phát nhạc là 30 giây.
- Output: Sau 30 giây, loa dừng phát nhạc và ứng dụng trở về trang chủ.

Trường hợp 2: Hẹn giờ dừng phát nhạc trong 5 phút

- Input: Đặt thời gian dừng phát nhạc là 5 phút.
- Output: Sau 5 phút, loa dừng phát nhạc và ứng dụng trở về trang chủ

Trường hợp 3: Hủy hẹn giờ dừng phát nhạc

- Input: Đặt thời gian dừng phát nhạc là 30 giây, sau đó hủy thời gian dừng phát.
- Output: Loa tiếp tục phát nhạc bình thường.

Trường hợp 4: Thay đổi thời gian hẹn giờ dừng phát nhạc

- Input: Đặt thời gian dừng phát là 30 giây, sau đó đổi sang 5 phút.
- Output: Sau 5 phút, loa dừng phát nhạc và ứng dụng trở về trang chủ.

3.1.3. Chức năng chọn bài hát:

- Input: Chọn bài hát số 3 trong danh sách bài hát của ứng dụng.
- Output: Loa phát đúng bài, màn hình LCD, ứng dụng hiển thị trạng thái đang phát bài số 3.

3.1.4. Chức năng phát, dừng nhạc:

Trường hợp 1: Dừng phát nhạc

- Input: Bài nhạc số 3 đang phát, ấn nút Play trên ứng dụng.
- Ouptut: Loa ngừng phát, màn hình LCD, ứng dụng hiển thị trạng thái dừng phát bài số 3.

Trường hợp 2: Phát nhạc

- Input: Bài nhạc số 3 đang dừng phát, ấn nút Play trên ứng dụng.
- Output: Loa sẽ tiếp tục phát bài số 3, màn hình LCD, ứng dụng hiển thị trạng thái đang phát bài số 3.

3.1.5. Chức năng tăng, giảm âm lượng:

Trường hợp 1: Tăng âm lượng

- Input: Âm lượng đang ở mức 5, điều chỉnh âm lượng lên mức 7.
- Output: Loa phát nhạc với âm lượng mức 7, màn hình LCD, ứng dụng hiển thị âm lượng mức 7.

Trường hợp 2: Giảm âm lượng

- Input: Âm lượng đang ở mức 5, điều chỉnh âm lượng lên mức 3.
- Output: Loa phát nhạc với âm lượng mức 3, màn hình LCD, ứng dụng hiển thị âm lượng mức 3.

3.1.6. Chức năng lặp lại:

Trường hợp 1: Lặp lại bài hát

- Input: Bài số 3 đang phát, chọn nút lặp lại.
- Output: Sau khi phát hết bài số 3, loa phát lại bài số 3, màn hình LCD, ứng dụng hiển thị trạng thái đang phát của bài số 3, biểu tượng lặp lại chuyển sang màu tím.

Trường hợp 2: Tắt nút lặp lại

- Input: Bài số 3 đang phát, biểu tượng lặp lại đang tô đậm, chọn nút lặp lại.
- Output: Sau khi phát hết bài số 3, loa phát qua bài số 4, màn hình LCD, ứng dụng hiển thị trạng thái đang phát của bài số 4, biểu tượng lặp lại chuyển về lại màu xám.

3.1.7. Chức năng chuyển bài:

Trường hợp 1: Chuyển sang bài kế tiếp

- Input: Bài số 4 đang phát, chọn nút Next.
- Output: Loa phát qua bài số 5, màn hình LCD, ứng dụng hiển thị trạng thái đang phát của bài số 5.

Trường hợp 2: Quay về bài trước đó

- Input: Bài số 5 đang phát, chọn nút Prev.
- Output: Loa phát về lại bài số 4, màn hình LCD, ứng dụng hiển thị trạng thái đang phát của bài số 4.

Trường hợp 3: Chuyển sang bài kế tiếp khi nút ngẫu nhiên đang bật

- Input: Bài số 4 đang phát, chọn biểu tượng phát ngẫu nhiên sau đó chọn nút Next.

- Output: Loa phát ngẫu nhiên bài số 2, màn hình LCD, ứng dụng hiển thị trạng thái đang phát bài số 2, biểu tượng phát ngẫu nhiên chuyển sang màu tím.

Trường hợp 4: Quay về bài trước đó khi nút ngẫu nhiên đang bật

- Input : Bài số 2 đang phát, biểu tượng phát ngẫu nhiên vẫn màu tím, chọn nút Prev.
- Output: Loa phát ngẫu nhiên bài số 7, màn hình LCD, ứng dụng hiển thị trạng thái đang phát bài số 7.

Trường hợp 5: Chuyển sang bài kế tiếp khi tắt nút ngẫu nhiên

- Input: Bài số 7 đang phát, chọn biểu tượng phát ngẫu nhiên màu tím sau đó chọn nút Next.
- Output: Loa phát qua bài số 8, màn hình LCD, ứng dụng hiển thị trạng thái đang phát bài số 8, biểu tượng phát ngẫu nhiên chuyển về màu xám.

Trường hợp 6: Chuyển sang bài kế tiếp khi nút lặp lại đang bật

- Input: Bài số 4 đang phát, biểu tượng lặp lại đang bật (đang tô đậm), sau đó chọn nút Next.
- Output: Loa phát bài số 5, màn hình LCD, ứng dụng hiển thị trạng thái đang phát bài số 5, biểu tượng lặp lại vẫn đang tô đậm

3.2. Đánh giá:

3.2.1. Ưu điểm:

- Hệ thống đáp ứng được mục tiêu đề ra.
- Hoạt động ổn định, không xảy ra lỗi.
- Giao diện bắt mắt, thân thiện với người dùng.

3.2.2. Nhược điểm:

- Hệ thống vẫn còn độ trễ trung bình 1.8 giây.

- Chưa thực hiện tải nhạc bằng ứng dụng trên điện thoại.
- Chưa đóng gói sản phẩm để tiện lợi cho việc di chuyển.

3.3. Kết luận:

Vì sử dụng trong mục đích làm đồ án nên hệ thống vẫn chưa thể đáp ứng những tiêu chuẩn công nghiệp, phải thử nghiệm và nghiên cứu trong 1 thời gian dài để xem tính ổn định của hệ thống.

Hướng phát triển tương lai:

- Tối ưu hóa chương trình để độ trễ của hệ thống thấp nhất có thể.
- Đóng gói sản phẩm gọn gàng, nhẹ dễ mang theo.
- Thêm chức năng tải nhạc bằng ứng dụng trên điện thoại.
- Thêm chế độ Bluetooth vào sản phẩm để tăng độ tiện ích cho người dùng.
- Ứng dụng di động thêm nhiều chức năng mới như: tìm kiếm bài hát, tạo từng playlist riêng cho từng thể loại nhạc, tạo playlist yêu thích.

4. PHÂN CÔNG CÔNG VIỆC:

Thành viên	Công việc	Đánh giá
Trần Văn Cường – 21521909	Xây dựng và thực hiện mô hình IoT Speaker. Điều khiển các module phát nhạc, màn hình, kết nối WiFi, MQTT, thực hiện giao tiếp với Firebase, xử lý tín hiệu gửi về.	9.5
Vòng Chí Cường – 21521910	Xây dựng và thực hiện ứng dụng di động cho thống IoTspeaker. Xây dựng và phát triển giải thuật cho các chức năng.	9.5

5. LIÊN KẾT BỔ SUNG:

Link sơ đồ thiết kế phần mềm: FlowChart - Google Drive

Link github: CuongTranMCU/IoTSpeaker (github.com)

Link demo hiện thực đồ án: Video Demo - Google Drive

--- Hết ---