ĐẠI HỌC QUỐC GIA TP.HỒ CHÍ MINH
TRƯỜNG ĐẠI HỌC BÁCH KHOA
KHOA ĐIỆN – ĐIỆN TỬ
BỘ MÔN ĐIỆN TỬ
----------o0o----------

ĐỒ ÁN MÔN HỌC

ỨNG DỤNG ESP32 VÀ BLUETOOTH TRONG TRUYỀN NHẬN DỮ LIỆU CẢM BIẾN THÔNG QUA MQTT

GVHD: Thầy Bùi Quốc Bảo
SVTH: Nguyễn Trường Sơn
MSSV: 2212930

TP. HỒ CHÍ MINH, THÁNG 6 NĂM 2025

LỜI CẢM ƠN

Trước tiên em xin cảm ơn đến giảng viên hướng dẫn đồ án 1 của em là Thầy Bùi Quốc Bảo thầy đã hỗ trợ góp ý cho em thực hiện được đồ án này. Em cũng xin cảm ơn các thầy cô trong khoa đã truyền đạt cho em những nền tảng kiến thức vững chắc trong suốt quá trình học tập tại trường. Nhờ đó, em mới có thể triển khai ý tưởng và hoàn thiện sản phẩm một cách cụ thể và có định hướng rõ ràng.

Em xin chân thành cảm ơn.

Tp. Hồ Chí Minh, ngày 27 tháng 5 năm 2025.

Sinh viên

Nguyễn Trường Sơn

TÓM TẮT ĐỒ ÁN

Đồ án này trình bày về việc ESP32 sẽ nhận dữ liệu từ những cảm biến như là: cảm biến ánh sáng, cảm biến nhiệt độ và độ ẩm,... Và nhận lệnh từ người dùng khi người dùng muốn gửi bất kì dữ liệu nào của bất kì cảm biến nào thông qua kết nối bluetooth với ESP32 đến máy tính thông qua giao thức MQTT.

MỤC LỤC

DANH SÁCH HÌNH MINH HỌA

Hình 8-10 MQTT kết nối thành công

1. GIỚI THIỆU

1.1 Tổng quan

Lĩnh vực truyền thông không dây đặc biệt là wifi và bluetooth đóng vài trò quan trọng trong việc phát triển các hệ thống IoT (Internet of Things) hiện đại. Sự kết hợp của công nghệ bluetooth và với giao thức mqtt giúp tối ưu việc truyền nhận dữ liệu và điều khiển thiết bị từ xa. Mục tiêu cần nghiên cứu là về cách truyền dẫn dữ liệu thông qua bluetooth và giao thức mqtt. Nhiệm vụ đặt ra là một mô hình hệ thống ESP32 truyện nhận dữ liệu bằng Bluetooth và MQTT.

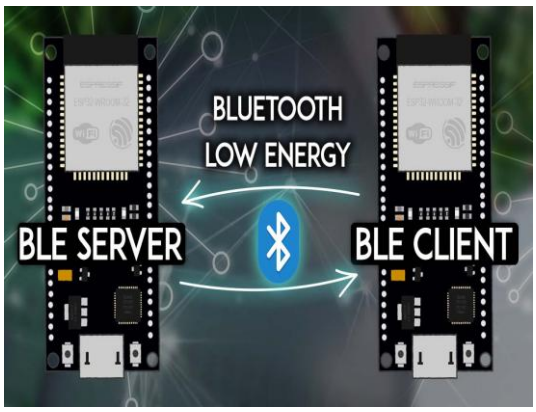
1.2 Nhiệm vụ đề tài

Sử dụng ESP32 đọc dữ liệu từ cảm biến nhiệt độ và độ ẩm DHT11, cảm biến ánh sáng có sử dụng LDR và cảm biến khí gas MQ2. Nhận lệnh từ người dùng thông qua kết nối BLE và gửi dữ liệu cần thiết đến với người dùng khác thông qua giao thức MQTT.

2. LÝ THUYẾT

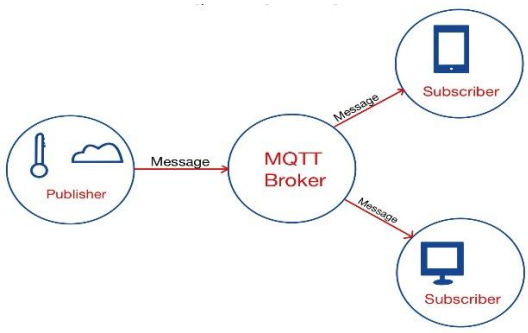
Bluetooth theo như đề tài sẽ sử dụng là BLE (Bluetooth Low Energy) và được sử dụng theo kiểu BLE client và BLE server. [1] “Bộ thu phát của Bluetooth Low Energy được thiết kế để hoạt động với mức tiêu thụ điện năng cực thấp.Bộ thu phát này truyền dữ liệu qua 40 kênh trong băng tần ISM 2.4GHz không cần giấy phép, mang đến cho các nhà phát triển sự linh hoạt to lớn để xây dựng các sản phẩm đáp ứng nhu cầu kết nối đặc thù của từng thị trường”. [2] “Attribute Protocol (ATT) định nghĩa cách một máy chủ công bố dữ liệu của mình cho máy khách (client) và cách mà dữ liệu này được cấu trúc. Generic Attribute Profile (GATT) định nghĩa định dạng của dữ liệu được thiết bị BLE công bố. Nó cũng định nghĩa

các thủ tục cần thiết để truy cập vào dữ liệu mà thiết bị công bố. Generic Access Profile (GAP) cung cấp một khung làm việc định nghĩa cách các thiết bị Bluetooth Low Energy tương tác với nhau. Điều này bao gồm: vai trò của thiết bị BLE, Quảng bá (phát quảng bá, khám phá thiết bị, các tham số quảng bá, dữ liệu quảng bá), thiết lập kết nối (khởi tạo kết nối, chấp nhận kết nối, các tham số kết nối và bảo mật". Kết nối BLE sẽ có các hoạt động như sau BLE server sẽ phát quảng bá và BLE client sẽ đồng ý kết nối khi thấy được tin quảng bá đó và cuối cùng là server và client có thể giao tiếp với nhau.



Hình 2-1 BLE server và BLE client

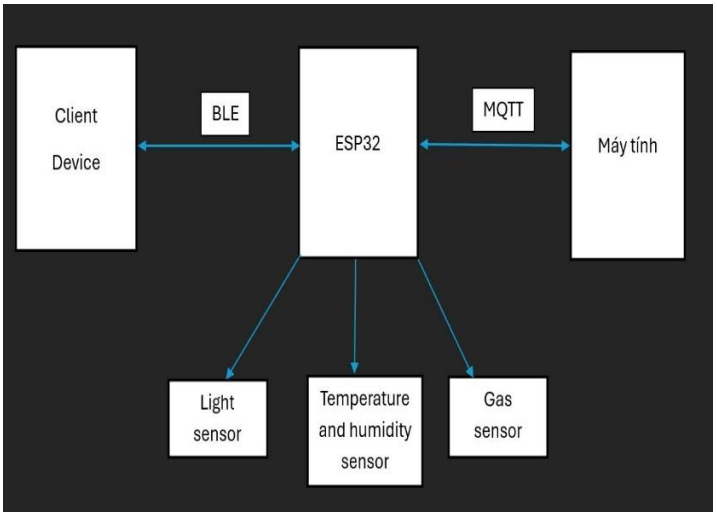
Giao thức truyền tin MQTT. [3] "MQTT là một giao thức nhắn tin tiêu chuẩn của OASIS dành cho IoT. Giao thức này được thiết kế cực kỳ nhẹ, theo mô hình publish/subscribe rất lý tưởng để kết nối các thiết bị từ xa với mã nguồn nhỏ gọn và băng thông mạng tối thiểu". Sử dụng MQTT broker là mosquitto [4] "hỗ trợ giao thức MQTT (v5.0, v3.1.1, v3.1)".



Hình 2-2 Giao thức MQTT

3. THIẾT KẾ VÀ THỰC HIỆN PHẦN CỨNG

Thiết kế 1 mạch bao gồm 1 ESP32 30pin, 1 cảm biến ánh sáng sử dụng LDR, 1 cảm biến nhiệt độ độ ẩm DHT11, 1 cảm biến khí gas MQ2, nguồn sử dụng Adapter 5V làm nguồn cung cấp chính cho mạch và sử dụng các jumper kiểu cái làm đế cho các linh kiện.



Hình 3-1 Block Diagram

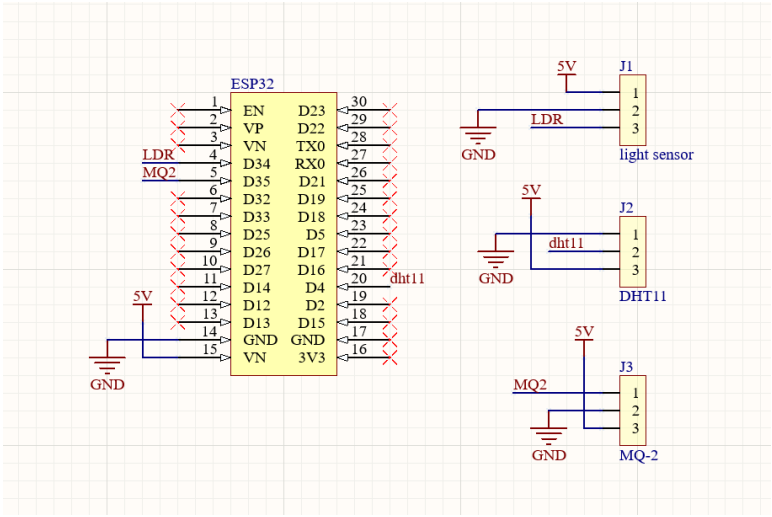
Client Device: Thiết bị của khác hàng kết nối BLE với ESP32 để có thể yêu cầu dữ liệu từ cảm biến.

ESP32 : đóng vai trò là BLE server đọc các giá trị cảm từ lệnh của client

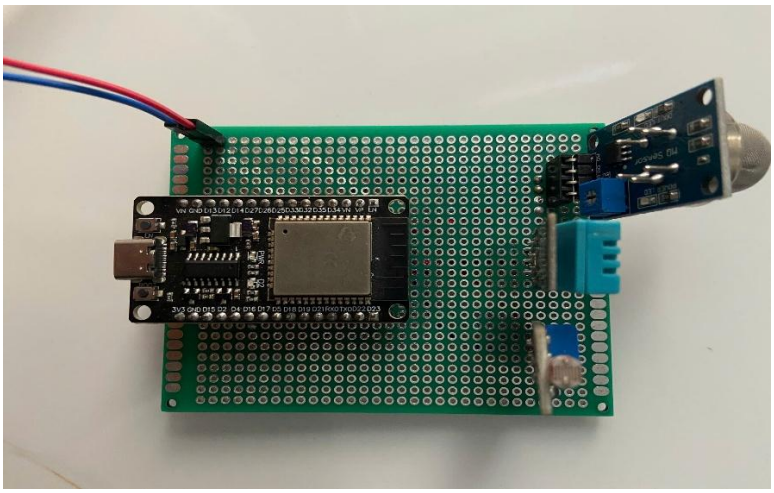
Light sensor: cảm biến đó cường độ ánh sáng có sử dụng LDR

Temperature and humidity sensor : cảm biến đo nhiệt độ và độ ẩm DHT11

Gas sensor : cảm biến đo lượng khí gas trong không khí, nồng độ và chất lượng của không khí MQ2

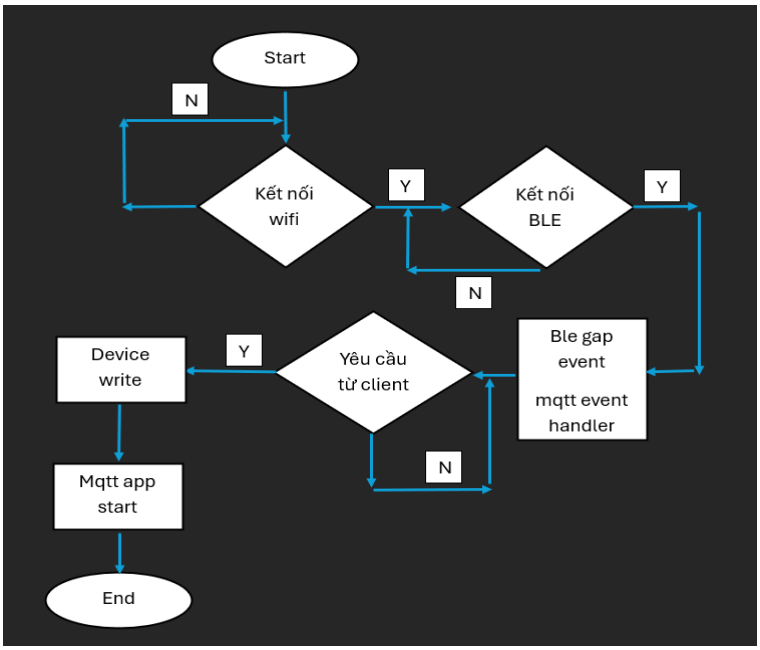


Hình 3-2 Schematic



Hình 3-3 Mạch thực tế

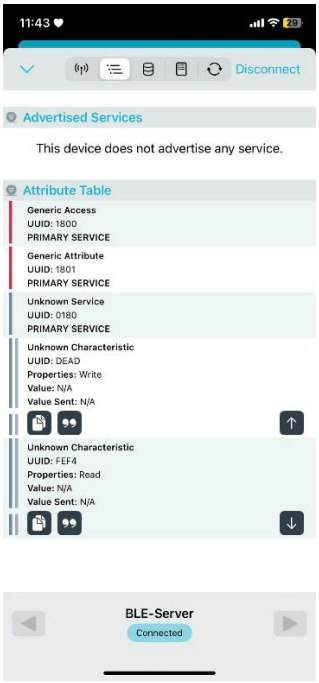
4. THIẾT KẾ VÀ THỰC HIỆN PHẦN MỀM



Hình 4-1 Lưu đồ giải thuật

5. KẾT QUẢ THỰC HIỆN

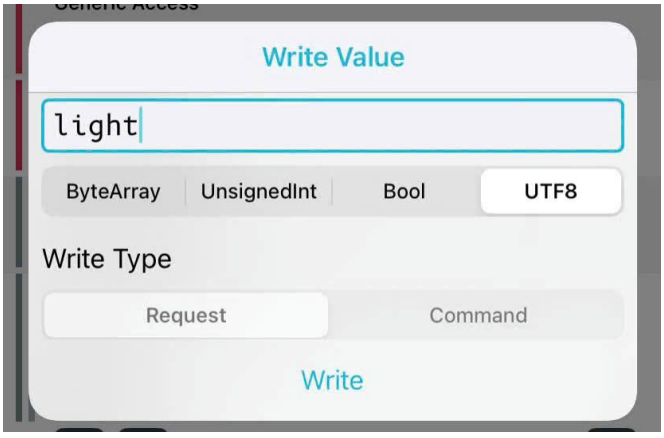
Sử dụng app nRF Connect for Mobile để thực hiện kết nối giữa điện thoại và ESP32.



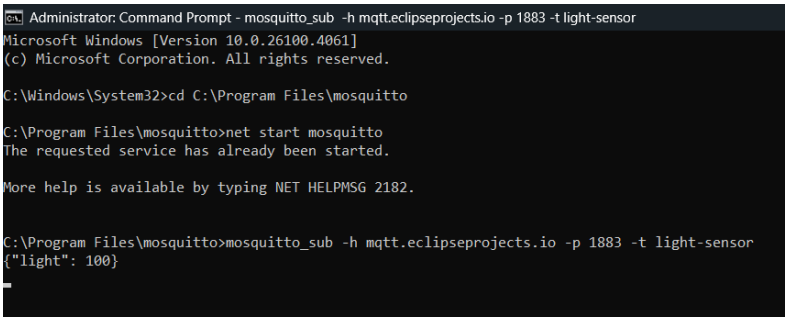
Hình 5-1 Sử dụng app nRF connect

Kết quả của mạch ESP32 gửi dữ liệu sensor thông qua kết nối BLE đến các máy khác qua thức truyền tin MQTT. Gồm các chức năng sau:

Đo cường độ ánh sáng thông qua cảm biến ánh sáng có sử dụng LDR

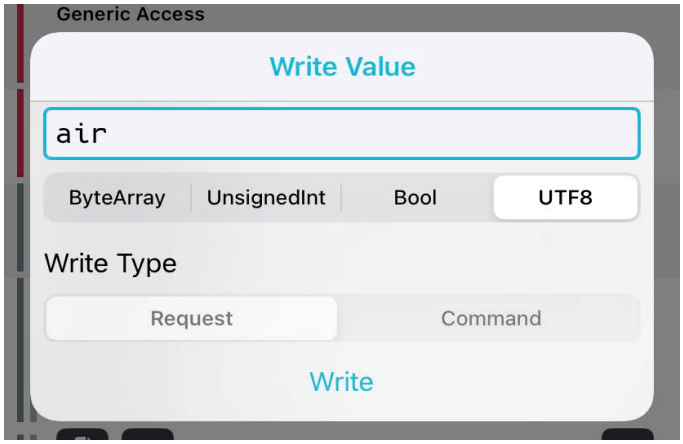


Hình 5-2 Nhận lệnh đo cường độ ánh sáng

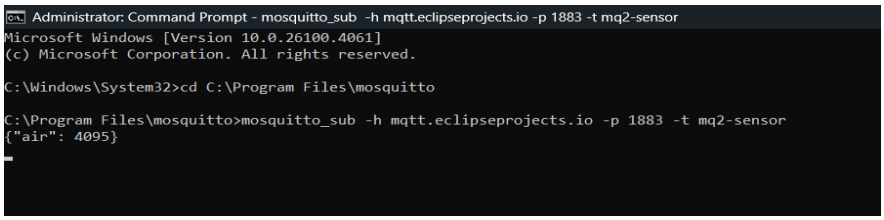


Hình 5-3 Gửi thành công dữ liệu cường độ ánh sáng

Đo nồng độ không khí thông qua cảm biến MQ2

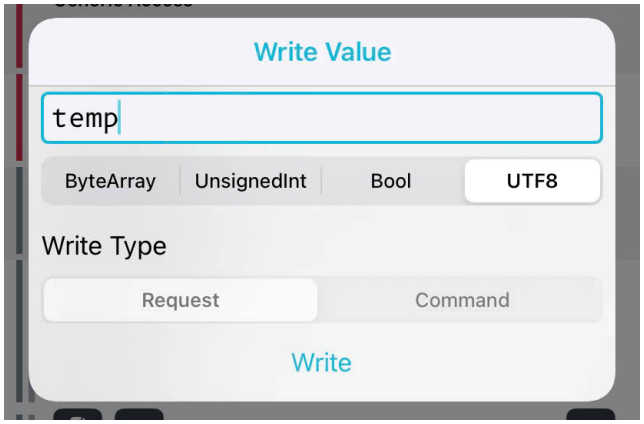


Hình 5-4 Nhận lệnh đo nồng độ không khí



Hình 5-5 Gửi thành công dữ liệu nồng độ không khí

Đo nhiệt độ và độ ẩm thông qua cảm biến DHT11



Hình 5-6 Nhận lệnh đo nhiệt độ và độ ẩm

Hình 5-7 Gửi thành công dữ liệu nhiệt độ và độ ẩm

Kết quả cho thấy lệnh từ client bằng kết nối BLE hoạt động tốt đã gửi được lệnh điều khiển tới ESP32, các cảm biến vẫn hoạt động bình thường và thành công gửi dữ liệu của cảm biến đến các máy khác thông qua MQTT.

6. KẾT LUẬN VÀ HƯỚNG PHÁT TRIỂN

6.1 Kết luận

Thông qua dự án giúp bản thêm em tìm hiểu và nghiên cứu thêm về các giao thức truyền thông không dây trong hệ thống IoT như Bluetooth mà trong dự án là Bluetooth Low Energy và Wifi thông qua cách sử dụng giao thức MQTT.

Với ưu điểm là truyền nhận dữ liệu không dây cho nên sẽ đỡ tốn chi phí mua dụng cụ cũng như làm gọn diện tích của sản phẩm làm ra. Có thể điều khiển thiết bị từ xa thông qua điện thoại của các khách hàng sử dụng sản phẩm như là có thể đi làm xa nhưng mà vẫn kiểm soát được các nhiệt độ ánh sáng trong căn nhà đảm bảo các con nhỏ được an toàn từ xa và

nếu có nguy hiểm thì sẽ có giải pháp ứng cứu kịp thời. Đó là 1 trong những ứng dụng thông của dự án này.

Về nhược điểm thì sản phẩm chỉ sử dụng Wifi ở một địa điểm cố định nếu đi đến địa điểm khác mà vẫn muốn sử dụng sản phẩm thì phải sửa lại phần Wifi SSID và Wifi password. Và sản phẩm vẫn còn ít cảm biến để có thể theo dõi các chức năng trong nhà từ xa được. Cũng như chưa có phần cảnh báo nếu như gặp mức độ nguy hiểm.

6.2 Hướng phát triển

Cần phải bổ sung trang web có thể đổi mật khẩu wifi thông qua điện thoại hoặc laptop mà không cần vào ESP-IDF. Cần bổ sung thêm chức năng cảnh báo khi các cảm biến đo được dữ liệu nằm ở vùng nguy hiểm và cần bổ sung thêm các cảm biến để bổ sung thêm chức năng cho dự án được đầy đủ và phù hợp khi sử dụng để giám sát trong nhà hơn.

7. TÀI LIỆU THAM KHẢO

[1] Bluetooth SIG, Inc, “Learn about Bluetooth”, www.bluetooth.com

[2] Mohammad Afaneh, “Bluetooth Low Energy (BLE): A Complete Guide”, novelbits.io

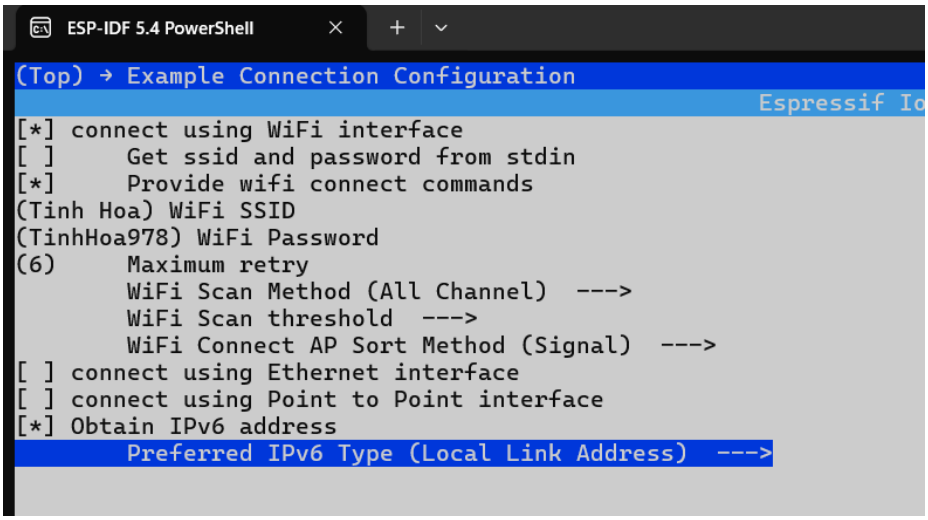
[3] OASIS (Organization for the Advancement of Structured Information Standards), “MQTT: The Standard for IoT Messaging”, mqtt.org

[4] Eclipse Foundation, “Eclipse Mosquitto”, mosquitto.org

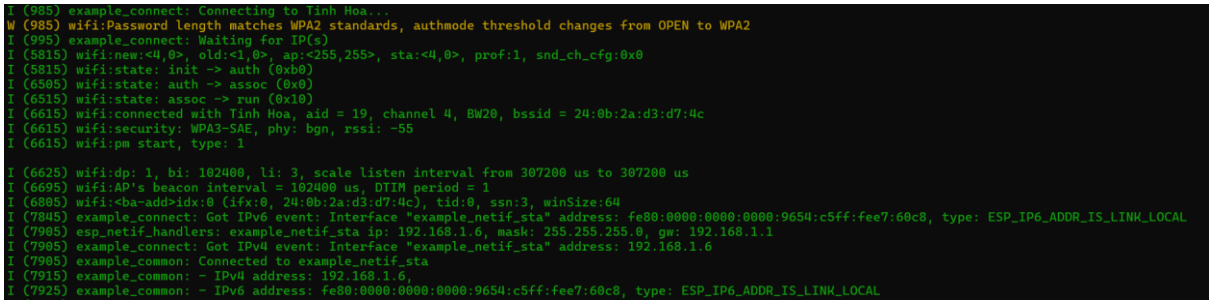
[5] Công ty THHH Công Nghệ Trung Sơn, “PPM là gì? Cách tính, hướng dẫn chuyển đổi, đơn vị tính PPM”, tschem.com.vn

8. PHỤ LỤC

Thay đổi Wifi cho project sử dụng lệnh idf.py menuconfig ở serial monitor sau khi đã build xong chương trình.



Hình 8-1 Thay đổi wifi cho project



Hình 8-2 Đã kết nối Wifi thành công

Khi đã kết nối xong Wifi tiếp đến là BLE thì ESP32 sẽ liên tục phát quảng bá đến các thiết bị xung quanh trong phạm vi của nó để được kết nối. Thông qua hàm ble_app_advertise.

```
9. void ble_app_advertise(void) {
10.     struct ble_hs_adv_fields fields;
11.     const char *device_name;
12.     memset(&fields, 0, sizeof(fields));
13.     device_name = ble_svc_gap_device_name();
14.     fields.name = (uint8_t *)device_name;
15.     fields.name_len = strlen(device_name);
16.     fields.name_is_complete = 1;
17.     ble_gap_adv_set_fields(&fields);
18.
19.     struct ble_gap_adv_params adv_params;
```

```
20.     memset(&adv_params, 0, sizeof(adv_params));
21.     adv_params.conn_mode = BLE_GAP_CONN_MODE_UND;
22.     adv_params.disc_mode = BLE_GAP_DISC_MODE_GEN;
23.     ble_gap_adv_start(ble_addr_type, NULL, BLE_HS_FOREVER, &adv_params,
    ble_gap_event, NULL);
24. }
```

Mảng dịch vụ GATT service trong BLE bằng NimBLE stack của ESP-IDF. Dùng để tạo 1 dịch vụ BLE với 2 characteristic: một cho write và một cho read.

```
25. static const struct ble_gatt_svc_def gatt_svcs[] = {
26.     {.type = BLE_GATT_SVC_TYPE_PRIMARY,
27.      .uuid = BLE_UUID16_DECLARE(0x0180),
```

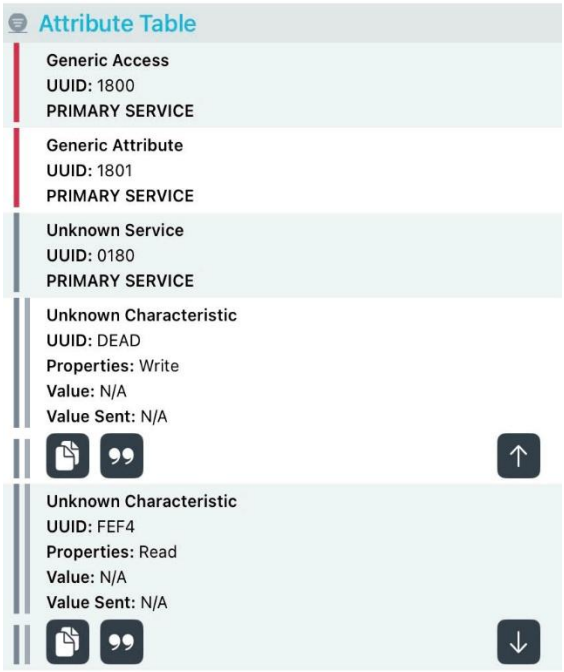
Đây là dịch vụ chính của BLE khi được kết nối .

```
28.      .characteristics = (struct ble_gatt_chr_def[]){
29.          {.uuid = BLE_UUID16_DECLARE(0xFEF4),
30.           .flags = BLE_GATT_CHR_F_READ,
31.           .access_cb = device_read},
```

Characteristic với UUID là 0xFEF4, cờ là READ thể hiện đọc dữ liệu từ BLE Server và hàm callback device_read để thực hiện hành động đọc giá trị từ BLE Server từ BLE Client.

```
32.          {.uuid = BLE_UUID16_DECLARE(0xDEAD),
33.           .flags = BLE_GATT_CHR_F_WRITE,
34.           .access_cb = device_write},
35.          {0}}},
36.
37.     {0}
38. };
```

Characteristic với UUID là 0xDEAD, cờ là WRITE thể hiện ghi dữ liệu vào BLE Server và hàm callback device_write để thực hiện xử lý dữ liệu được ghi vào BLE Server.



Hình 8-3 GATT Service

Hàm ble_gap_event quản lý các sự kiện như kết nối và ngắt kết nối của BLE.

```
39.static int ble_gap_event(struct ble_gap_event *event, void *arg) {
40.    switch (event->type) {
41.    case BLE_GAP_EVENT_CONNECT:
42.        ESP_LOGI("GAP", "BLE GAP EVENT CONNECT %s", event-
   >connect.status == 0 ? "OK!" : "FAILED!");
43.        if (event->connect.status != 0) {
44.            ble_app_advertise();
45.        }
46.        break;
47.    case BLE_GAP_EVENT_DISCONNECT:
48.        ESP_LOGI("GAP", "BLE GAP EVENT DISCONNECTED");
49.        break;
50.    case BLE_GAP_EVENT_ADV_COMPLETE:
51.        ESP_LOGI("GAP", "BLE GAP EVENT ADV COMPLETE");
52.        ble_app_advertise();
53.        break;
54.    default:
55.        break;
56.    }
57.    return 0;
58.}
59.
60.
```

Nếu BLE được kết nối thành công thì trên serial monitor sẽ xuất hiện dòng " GAP: BLE GAP EVENT CONNECT OK!". Nếu ngắt kết nối sẽ in ra "GAP: BLE GAP EVENT DÍCONNECTED". Còn nếu như khi quảng bá của BLE kết thức thì sẽ in ra serial monitor " GAP: BLE GAP EVENT ADV COMPLETE" và thực hiện quảng bá lại. Nếu dùng app nRF Connect bấm connect vô thiết bị tên là BLE-Server thì serial monitor sẽ xuất hiện bảng như sau:

```
I (7935) BLE-Server: Other event id:7
I (7935) main_task: Returned from app_main()
I (8335) wifi:<ba-add>idx:1 (ifx:0, 24:0b:2a:d3:d7:4c), tid:5, ssn:1, winSize:64
I (8945) BLE-Server: MQTT_EVENT_CONNECTED
I (8955) BLE-Server: sent publish successful, msg_id=14430
I (8955) BLE-Server: sent subscribe successful, msg_id=60695
I (9265) BLE-Server: Other event id:5
I (9665) BLE-Server: Other event id:3
I (40115) GAP: BLE GAP EVENT CONNECT OK!
```

Hình 8-4 Đã kết nối BLE thành công

```
61.int raw_value = 0;
62.     esp_err_t ret = adc_oneshot_read(adc1_handle, LIGHT_CHANNEL,
   &raw_value);
63.     int light_intensity = (int)((float)raw_value / 4095.0f * 100);
```

Đọc giá trị cảm biến ánh sáng ở chế độ oneshot tức là chỉ đọc 1 lần khi có yêu cầu. Lưu giá trị vào biến raw_value và tiến hành chuyển đổi từ giá trị đọc được từ giá trị số ADC(0-4095) sáng điện áp thực tế (0-3.3V) đó là giá trị của cường độ ánh sáng.

```
64.     int raw_value = 0;
65.     esp_err_t ret = adc_oneshot_read(adc1_handle, MQ2_CHANNEL,
   &raw_value);
```

Tương tự như ở cảm biến đo cường độ ánh sáng cảm biến khí gas MQ2 cũng dùng chế độ oneshot để đo giá trị cảm biến.

```
66.  float mq2_get_ppm(int raw_value) {
67.     float voltage = ((float)raw_value / 4095.0f) * 3.3f;
68.     float A = 1000.0f;
69.     float B = -1.5f;
70.     float ppm = A * powf(voltage, B);
71.
72.     ESP_LOGI(TAG_SENSOR, "Gas Concentration: %.2f ppm", ppm);
73.     return ppm;
74.}
```

Hàm mq2_get_ppm tính toán đưa giá trị đã lấy được từ cảm biến ở trên để tính theo công thức và cho kết quả là giá trị ppm tính theo công thức.[5] "PPM là đơn vị để đo mật độ đối với thể tích, khối lượng cực kỳ thấp. Đơn vị ppm này thường được sử dụng trong các phép tính toán đo lường hoặc phân tích vi lượng, nghĩa là cái nó dùng để đo cực kỳ nhỏ. Người ta sử dụng đơn vị ppm trong nhiều ngành,như hoá học, vật lý, toán học, điện tử, … và nhất là dùng ppm để đo nồng độ các loại khí thải, khí gây ô nhiễm, và tính trên thể tích một lít".

```
75.void dht11_test(void *pvParameters){
76.    DHT11_init(GPIO_NUM_4);
77.    while(1) {
78.        stDht11Reading = DHT11_read();
79.        if(DHT11_OK == stDht11Reading.status)    {
80.            ESP_LOGI(TAG_DHT11,"Temperature: %d °C\tHumidity: %d
   %%",stDht11Reading.temperature,stDht11Reading.humidity);
81.        char dht11_str[50];
82.        snprintf(dht11_str, sizeof(dht11_str), "{\"temperature\": %d,
   \"humidity\": %d}",stDht11Reading.temperature,
   stDht11Reading.humidity);
83.        esp_mqtt_client_publish(mqtt_client, "dht11-sensor", dht11_str,
   0, 1, 0);
84.            }else        {
85.            ESP_LOGW(TAG_DHT11,"Cannot read from sensor:
   %s",(DHT11_TIMEOUT_ERROR == stDht11Reading.status) ? "Timeout" : "Bad
   CRC");     }
86.        vTaskDelay(2500 / portTICK_PERIOD_MS);    }}
```

Hàm dht11_test dùng để đọc cảm biến DHT11 và gửi nó đến topic "dht11-sensor" bằng giao thức MQTT. Lấy chân GPIO 4 trên ESP32 làm chân tín hiệu nếu không có lỗi sẽ tiến hành đọc nhiệt độ và độ ẩm từ cảm biến và in ra trên serial monitor. Sau đó tạo 1 chuỗi JSON để gửi dữ liệu. Gửi dữ liệu đến topic đã subcribe nếu lỗi thì sẽ in lên serial monitor là "Cannot read from sensor : Bad CRC".

```
87.static int device_write(uint16_t conn_handle, uint16_t attr_handle,
   struct ble_gatt_access_ctxt *ctxt, void *arg)
88.{
89.    char *data = (char *)ctxt->om->om_data;
90.    int data_len = ctxt->om->om_len;
```

Hàm device_write thực hiện xử lí các lệnh nhận được thông qua kết nối BLE với client.

```
91.if (data_len > 0 && data[data_len - 1] != '\0') {
92.         data[data_len] = '\0';
93.     }
```

Câu lệnh đảm bảo nhận chuỗi đầu vào đúng ký tự và kết thúc bằng ký tự "/0".

```
94. if (strcmp(data, "temp") == 0)
95.     {
96.         printf("Received Data Length: %d\n", data_len);
97.         esp_mqtt_client_publish(mqtt_client, "ble_send_data", data,
    data_len, 1, 0);
98.         xTaskCreate(dht11_test, "dht11_test", configMINIMAL_STACK_SIZE
    * 3, NULL, 5, NULL);
99.     }
100.
```

Hàm if thực hiện lệnh điều khiển cho khi muốn đo cảm biến nhiệt độ. Nếu như khách hàng nhập vào app nRF Connect theo chuẩn UTF8 là "temp" thì ở serial monitor sẽ in ra những thông tin sau: kích thước chuỗi mà khách hàng đã nhập. Ví dụ "temp" in ra sẽ là 4 ký tự, Điều này nhằm đảm bảo rằng đã nhập đúng lệnh để thực hiện đúng chức năng của mạch.

```
Received Data Length: 4
I (57015) BLE-Server: Other event id:5
I (57705) DHT11: Temperature: 31 °C     Humidity: 71 %
I (58045) BLE-Server: Other event id:5
W (60225) DHT11: Cannot read from sensor: Bad CRC
W (62745) DHT11: Cannot read from sensor: Bad CRC
```

Hình 8-5 Data của DHT11 trong serial monitor

```
101.      else if (strcmp(data, "light") == 0)
102.          {
103.              printf("Received Data Length: %d\n", data_len);
104.              esp_mqtt_client_publish(mqtt_client, "ble_send_data",
    data, data_len, 1, 0);
105.
106.              sensor_manager_init();
107.
108.              int light_value = light_sensor_read();
109.              ESP_LOGI(TAG_LIGHT_SENSOR, "Light Intensity: %d%%",
    light_value);
110.
111.              sensor_manager_deinit();
```

```
112.
113.             char light_str[50];
114.             snprintf(light_str, sizeof(light_str), "{\"light\": %d}",
   light_value);
115.
116.             os_mbuf_append(ctxt->om, light_str, strlen(light_str));
117.             esp_mqtt_client_publish(mqtt_client, "light-sensor",
   light_str, 0, 1, 0);
118.         }
```

Hàm else if này dùng để thực hiện lệnh “light” từ client tức là lệnh đọc giá trị từ cảm biến ánh sáng. Trước tiên nó sẽ in ra số ký tự mà client đã nhập vào và gửi đến topic “ble_send_data”. Tiếp đến nó sẽ thực hiện việc khởi động sensor tức là dùng GPIO 34 là ADC1_6 để đọc giá trị từ cảm biến. Sau đó tương tự như hàm dht11_test cũng tạo 1 chuỗi JSON để gửi dữ liệu là biến đổi giá trị điện áp thành ký tự để gửi đi. Cuối cùng là gửi đến topic “light-sensor”.

```
Received Data Length: 5
I (115815) SENSOR_MANAGER: Sensor Manager Initialized
I (115815) SENSOR_MANAGER: Light Intensity: 100%
I (115815) LIGHT_SENSOR: Light Intensity: 100%
I (115825) SENSOR_MANAGER: ADC unit freed successfully
```

Hình 8-6 Data của light sensor trong serial monitor

```
119.     else if (strcmp(data, "air") == 0)
120.         {
121.             printf("Received Data Length: %d\n", data_len);
122.             esp_mqtt_client_publish(mqtt_client, "ble_send_data",
   data, data_len, 1, 0);
123.
124.             sensor_manager_init();
125.
126.             int air_value = mq2_sensor_read();
127.             ESP_LOGI(TAG_MQ2, "Air Quality: %d%%", air_value);
128.
129.             sensor_manager_deinit();
130.
131.             char air_str[50];
132.             snprintf(air_str, sizeof(air_str), "{\"air\": %d}",
   air_value);
133.
134.             os_mbuf_append(ctxt->om, air_str, strlen(air_str));
135.             esp_mqtt_client_publish(mqtt_client, "mq2-sensor",
   air_str, 0, 1, 0);
136.         }
```

Hàm else if này dùng để thực hiện lệnh "air" từ client tức là lệnh đọc giá trị từ cảm biến khí gas. Tương tự như cảm biến ánh trước tiên nó sẽ in ra số ký tự mà client đã nhập vào và gửi đến topic "ble_send_data". Tiếp đến nó sẽ thực hiện việc khởi động sensor tức là dùng GPIO 35 là ADC1_7 để đọc giá trị từ cảm biến. Sau đó tương tự như hàm dht11_test cũng tạo 1 chuỗi JSON để gửi dữ liệu là biến đổi giá trị điện áp thành ký tự để gửi đi. Cuối cùng là gửi đến topic "mq2-sensor".

```
Received Data Length: 3
I (39265) SENSOR_MANAGER: Sensor Manager Initialized
I (39265) SENSOR_MANAGER: MQ2 Raw ADC Value: 4095
I (39265) MQ2_SENSOR: Air Quality: 4095%
I (39265) SENSOR_MANAGER: ADC unit freed successfully
```

Hình 8-7 Data của MQ2 trong serial monitor

```
137.     else
138.         {
139.             printf("Received Data Length: %d\n", data_len);
140.             printf("Data from the client: %.*s\n", ctxt->om->om_len,
   ctxt->om->om_data);
141.             esp_mqtt_client_publish(mqtt_client, "ble_send_data",
   data, data_len, 1, 0);
142.         }
```

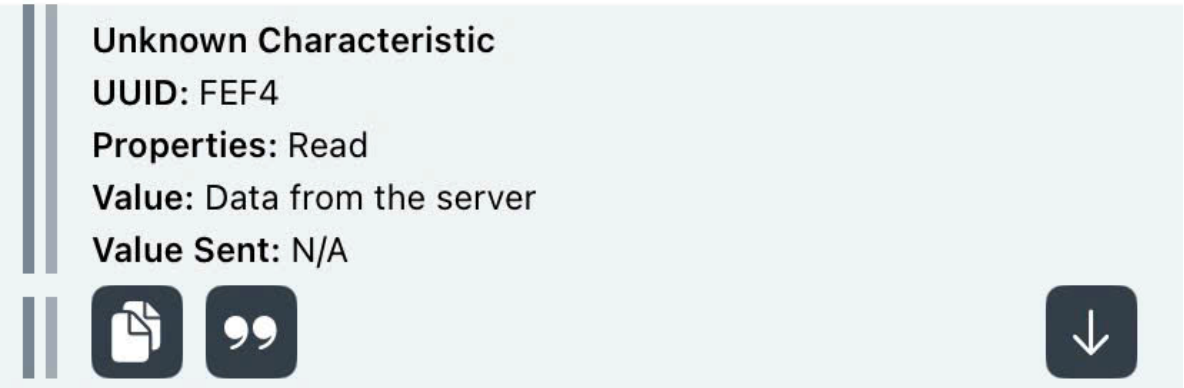
Hàm else này dùng để hiển thị các lệnh mà khách hàng đưa ra nhưng mà không nằm trong các lệnh thực hiện đo các cảm biến. Thì đầu tiên nó cũng như các lệnh khác cũng in ra độ dài của các ký tự, các ký tự đó và gửi nó đến topic "ble_send_data" bằng giao thức MQTT.

```
Received Data Length: 5
Data from the client: hello
I (875375) BLE-Server: Other event id:5
```

Hình 8-8 Data ngẫu nhiên không nằm trong lệnh đọc cảm biến

```
143.     static int device_read(uint16_t con_handle, uint16_t attr_handle,
   struct ble_gatt_access_ctxt *ctxt, void *arg) {
144.         os_mbuf_append(ctxt->om, "Data from the server", strlen("Data
   from the server"));
145.         return 0;
146.     }
```

Hàm device_read dùng để đọc giá trị từ cảm biến. Ở đây dùng app nRF Connect thì ta sẽ nhận được chuỗi “Data from the server” trên điện thoại của khách hàng.



Hình 8-9 Data khi dùng chức năng Read

```
147.      static void mqtt_event_handler(void *handler_args,
   esp_event_base_t base, int32_t event_id, void *event_data) {
148.          ESP_LOGD(TAG_BLE, "Event dispatched from event loop base=%s,
   event_id=%" PRIi32 "", base, event_id);
149.          esp_mqtt_event_handle_t event = event_data;
150.          esp_mqtt_client_handle_t client = event->client;
151.          int msg_id;
152.
153.          switch ((esp_mqtt_event_id_t)event_id) {
154.          case MQTT_EVENT_CONNECTED:
155.              ESP_LOGI(TAG_BLE, "MQTT_EVENT_CONNECTED");
156.              msg_id = esp_mqtt_client_publish(client, "esp32",
   "hello", 0, 1, 0);
157.              ESP_LOGI(TAG_BLE, "sent publish successful, msg_id=%d",
   msg_id);
158.              msg_id = esp_mqtt_client_subscribe(client, "mqttesp32",
   0);
159.              ESP_LOGI(TAG_BLE, "sent subscribe successful, msg_id=%d",
   msg_id);
160.              break;
161.          case MQTT_EVENT_DATA:
162.              ESP_LOGI(TAG_BLE, "MQTT_EVENT_DATA");
163.              printf("TOPIC=%.*s\r\n", event->topic_len, event->topic);
164.              printf("DATA=%.*s\r\n", event->data_len, event->data);
165.              break;
166.          default:
167.              ESP_LOGI(TAG_BLE, "Other event id:%d", event->event_id);
168.              break;
169.          }
170.      }
```

Hàm mqtt_event_handler dùng để quản lý các sự kiện khi kết nối MQTT. Chẳng hạn như khi MQTT kết nối thành công thì nó sẽ publish vào topic “esp32” với tin nhắn là “hello” và subcribe vào topic “mqttesp32”. Trên serial monitor nó sẽ xuất hiện các dòng như sau thể hiện MQTT đã được khởi động thành công.

```
I (18855) BLE-Server: MQTT_EVENT_CONNECTED
I (18855) BLE-Server: sent publish successful, msg_id=63662
I (18865) BLE-Server: sent subscribe successful, msg_id=33915
```

Hình 8-10 MQTT kết nối thành công

```
171.     static void mqtt_app_start(void) {
172.         esp_mqtt_client_config_t mqtt_cfg = {
173.             .broker.address.uri =
   "mqtt://mqtt.eclipseprojects.io:1883",
174.         };
175.
176.         mqtt_client = esp_mqtt_client_init(&mqtt_cfg);
177.         esp_mqtt_client_register_event(mqtt_client, ESP_EVENT_ANY_ID,
   mqtt_event_handler, NULL);
178.         esp_mqtt_client_start(mqtt_client);
179.     }
180.
```

Hàm mqtt_app_start() dùng để khởi tạo và kết nối các MQTT Client trong ESP IDF. Ở đây sử dụng MQTT Broker miễn phí với địa chỉ là mqtt.elipseproject.io và cổng port 1883.