ĐẠI HỌC QUỐC GIA TP. HỒ CHÍ MINH

TRƯỜNG ĐẠI HỌC CÔNG NGHỆ THÔNG TIN

BÁO CÁO BÀI THỰC HÀNH LAB03

LỚP CE232.N21.1

Giảng viên hướng dẫn: TRẦN HOÀNG LỘC

Sinh viên thực hiện nhóm 7:

Trương Hữu Khang	20520211
Nguyễn Linh Anh Khoa	20520219
Hà Vĩnh Kiện	20520597
Phan Duy Thông	20520789

TP. Hồ Chí Minh, 2023

PHẦN ĐỀ:

1. Dựa trên project ble_compatibility_test, trả lời các câu hỏi sau:

a. Trình bày cách cài đặt Advertise Data, Scan Response Data và quá trình quảng bá dữ liệu? Phân tích và cho biết ý nghĩa các phần tử của mảng raw_adv_data[]?

b. Nêu cách cài đặt truyền/nhận dữ liệu giữa ESP32 và điện thoại (qua ứng dụng LightBlue® Explorer)?

2. Dựa trên test case được cung cấp trong file ble_compatibility_test_case.md:

Hãy viết báo cáo kết quả test (lưu ý thực hiện đủ số test lần yêu cầu).

3. Một nhà hàng muốn gửi khuyến mãi giảm giá 50% đến tất cả các khách hàng đứng trước cửa nhà hàng:

Sử dụng ESP32 để hiện thực mô hình BLE Broadcasting gửi khuyến mãi cho tất cả các điện thoại ở gần. Test bằng cách dùng nhiều điện thoại và quan sát trường dữ liệu tương ứng trên ứng dụng LightBlue.

4. Viết một chương trình thực hiện các công việc sau:

a. Quảng bá ESP32 với tên thiết bị là tên nhóm.

b. Thiết lập kết nối ESP32 và điện thoại.

c. Dùng ứng dụng LightBlue để đọc và gửi dữ liệu đến ESP32 theo thứ tự:

> ESP32: Please input ID of each team member:

> Mobile: <ID1>

> Mobile: <ID2>

> Mobile: <ID3>

d. Mỗi khi điện thoại gửi dữ liệu ID (mã số sinh viên) thành công, ESP32 hiển thị ID đó lên LCD.

PHẦN BÁO CÁO

1. Dựa trên project ble_compatibility_test, trả lời các câu hỏi sau:

a. Trình bày cách cài đặt Advertise Data, Scan Response Data và quá trình quảng bá dữ liệu? Phân tích và cho biết ý nghĩa các phần tử của mảng raw_adv_data[]?

Đầu tiên, ta sẽ cài đặt cấu trúc của gói tin adv_data bằng cách sử dụng hàm esp_err_t esp_ble_gap_config_adv_data(esp_ble_adv_data_t *adv_data). Hàm này nhận vào một con trỏ đến cấu trúc esp_ble_adv_data_t, mô tả dữ liệu quảng cáo cần được cấu hình. Các trường của của cấu trúc esp_ble_adv_data_t bao gồm:

- set_scan_rsp: Giá trị boolean chỉ ra liệu dữ liệu quảng cáo có phải là một phản hồi quét (scan response) hay không. Trong trường hợp này, nó được thiết lập thành false, cho biết dữ liệu quảng cáo không phải là một phản h quét.
- include_name: Giá trị boolean chỉ ra liệu tên thiết bị có nên được bao gồm trong dữ liệu quảng cáo hay không. Trong trường hợp này, nó được thiết lập thành true, cho biết tên thiết bị sẽ được bao gồm.
- include_txpower: Giá trị boolean chỉ ra liệu mức công suất truyền có nên được bao gồm trong dữ liệu quảng cáo hay không. Trong trường hợp này, nó được thiết lập thành true, cho biết mức công suất truyền sẽ được bao gồm.
- min_interval: Khoảng thời gian quảng cáo tối thiểu, tính bằng đơn vị 0.625 ms. Trong trường hợp này, nó được thiết lập thành 0x20, tương ứng với khoảng thời gian quảng cáo là 20 x 0.625 ms = 12.5 ms.

- max_interval: Khoảng thời gian quảng cáo tối đa, tính bằng đơn vị 0.625 ms. Trong trường hợp này, nó được thiết lập thành 0x40, tương ứng với khoảng thời gian quảng cáo là 40 x 0.625 ms = 25 ms.
- appearance: Hình dạng của thiết bị, được định nghĩa trong các thông số kỹ thuật Bluetooth SIG. Trong trường hợp này, nó được thiết lập thành 0x00, cho biết thiết bị có hình dạng khôngxác định được.
- manufacturer_len: Độ dài dữ liệu nhà sản xuất, tính bằng byte. Trong trường hợp này, nó được thiết lập thành 0, cho biết không có dữ liệu nhà sản xuất nào.
- p_manufacturer_data: Con trỏ đến dữ liệu nhà sản xuất sẽ được bao gồm trong dữ liệu quảng cáo. Trong trường hợp này, nó được thiết lập thành NULL, cho biết không có dữ liệu nhà sản xuất nào.
- service_data_len: Độ dài dữ liệu dịch vụ, tính bằng byte. Trong trường hợp này, nó được thiết lập thành 0, cho biết không có dữ liệu dịch vụ nào.
- p_service_data: Con trỏ đến dữ liệu dịch vụ sẽ được bao gồm trong dữ liệu quảng cáo. Trong trường hợp này, nó được thiết lập thành NULL, cho biết không có dữ liệu dịch vụ nào.
- service_uuid_len: Độ dài UUID dịch vụ,tính bằng byte. Trong trường hợp này, nó được thiết lập bằng kích thước của mảng service_uuid, được giả định chứa một UUID 16-bit duy nhất.
- p_service_uuid: Con trỏ đến UUID dịch vụ sẽ được bao gồm trong dữ liệu quảng cáo. Trong trường hợp này, nó được thiết lập bằng mảng service_uuid, chứa một UUID 16-bit duy nhất.
- flag: Một bit field chỉ ra các cờ liên quan đến dữ liệu quảng cáo. Trong trường hợp này, nó được thiết lập thành ESP_BLE_ADV_FLAG_GEN_DISC | ESP_BLE_ADV_FLAG_BREDR_NOT_SPT, cho biết thiết bị sẽ sử dụng chế độ phát hiện chung và không hỗ trợ kết nối Bluetooth classic (BR/EDR).

Ngoài ra, ta có thể chọn cách cấu hình đơn giản của raw_adv_data, Các phần tử trong mảng raw_adv_data[] :

- Flags (0x02, 0x01, 0x06): có độ dài là 3 byte chứa các thông tin về các tính năng được hỗ trợ bởi thiết bị Bluetooth. Ý nghĩa của từng byte:
    - 0x02: độ dài của dữ liệu là 2 byte
    - 0x01: dạng dữ liệu là flags
    - 0x06: giá trị của flags
- Tx Powers (0x02, 0x0a, 0xeb): có độ dài 3 byte chứa các thông tin về công suất phát của thiết bị Bluetooth. Trong trường hợp này giá trị của trường này là 0xeb cho biết công suất phát của thiết bị là -21dB
- Service UUID (0x03, 0x03, 0xFF, 0x00): có độ dài 4 byte chứa thông tin về UUID (Universally Unique Identifier) của dịch vụ BLE được cung cấp. Trong trường hợp này, giá trị của UUID là 0xFF00.
- Device Name (0x0E, 0x09, 'B', 'L', 'E', '', 'C', 'O','M', 'P', '', 'T','E', 'S', 'T'): có độ dài 15 byte mang thông tin về tên gọi của thiết bị, trong trường hợp này thiết bị có tên là BLE_COMP_TEST.

Sau khi quá trình cấu hình adv_data hoàn tất, chương trình sẽ bật cờ hiệu để báo hiệu việc hoàn thành, tiếp đó ta sẽ cấu hình cho scan_response_data bằng câu lệnh esp_err_t esp_ble_gap_config_scan_rsp_data(esp_ble_adv_data_t *scan_rsp_data). Hàm này nhận vào một con trỏ đến cấu trúc esp_ble_adv_data_t, mô tả dữ liệu phản hồi quét cần được cấu hình.

Quá trình quảng bá dữ liệu: khi thiết bị BLE được bật lên, nó sẽ gửi Advertise Data đến các thiết bị khác trong vùng phủ sóng của nó. Nếu một thiết bị khác quét được nó, nó sẽ gửi Scan Request đến thiết bị BLE đó và thiết bị BLE sẽ trả lời bằng cách gửi Scan Response Data.

b. Nêu cách cài đặt truyền/nhận dữ liệu giữa ESP32 và điện thoại (qua ứng dụng LightBlue® Explorer)?

1. Trên ESP32, cần phải cài đặt Bluetooth để có thể kết nối với điện thoại. Có thể sử dụng thư viện Bluetooth của ESP32 để thực hiện điều này.
2. Tạo một dịch vụ Bluetooth trên ESP32 để truyền/nhận dữ liệu. Có thể sử dụng thư viện BLE của ESP32 để thực hiện điều này. Dịch vụ Bluetooth là một tập hợp các thuộc tính (characteristics) mà điện thoại có thể truy cập để truyền/nhận dữ liệu.
3. Trong ứng dụng LightBlue® Explorer trên điện thoại, cần phải tìm kiếm và kết nối với ESP32. Để tìm kiếm ESP32, có thể sử dụng tính năng "Scan" trong ứng dụng LightBlue® Explorer.
4. Sau khi kết nối với ESP32, cần phải tìm và truy cập vào dịch vụ Bluetooth đã tạo trên ESP32 để truyền/nhận dữ liệu. Để làm điều này, bạn có thể sử dụng tính năng "Discover" trong ứng dụng LightBlue® Explorer.
5. Khi đã truy cập vào dịch vụ Bluetooth trên ESP32, bạn có thể truyền/nhận dữ liệu thông qua các thuộc tính (characteristics) trong dịch vụ đó. Bạn có thể sử dụng tính năng "Write" để gửi dữ liệu từ điện thoại đến ESP32 và tính năng "Read" để đọc dữ liệu từ ESP32.

2. Dựa trên test case được cung cấp trong file ble_compatibility_test_case.md

- **Test for ADV Performance**
  - Thiết bị ESP32: terminal hiển thị “advertising start successfully” ngay sau khi nạp code
  - Light Blue Explorer: cả 10 lần refresh đều quét được “BLE_COMP_TEST”
  - Kết quả: Đạt
- **Test for Pairing Performance**
  - Thiết bị ESP32: terminal hiển thị “ESP_GATTS_CONNECT_EVT” sau khi thiết bị gửi yêu cầu ghép đôi, đồng thời hiển thị mật khẩu là 123456 trên

terminal. Sau khi nhập mật khẩu trên thiết bị thì thông báo “pair status = success” xuất hiện trên màn hình

- Light Blue Explorer: ghép đôi với ESP32 thành công và hiển thị các thông tin quảng bá của ESP32
- Kết quả: Đạt

- **Test for Service Discovery Performance**
  - Light Blue Explorer: hiển thị đầy đủ 3 characteristics “Char_1_Short_WR”, “Char_2_Long_WR”, “Char_3_Short_Notify”
  - Kết quả: Đạt
- **Test for Read and Encrypt**
  - Thiết bị ESP32: terminal hiển thị “read char_1”
  - Light Blue Explorer: các giá trị “11 22 33 44” xuất hiện ngay dưới dòng “READ AGAIN”
  - Kết quả: Đạt
- **Test for Short Notify**
  - Thiết bị ESP32: terminal hiển thị “short write success”
  - Light Blue Explorer: giá trị “88 99” được hiển thị lên giao diện của ứng dụng
  - Kết quả: Đạt
- **Test for Long Read and Write**
  - Thiết bị ESP32: terminal hiển thị “ESP_GATTS_EXEC_WRITE_EVT, Length=256” và “long write success”.
  - Light Blue Explorer: giá trị chuỗi gồm 256 ký tự được hiển thị lên giao diện của ứng dụng
  - Kết quả: Đạt
- **Test for Short Notify**
  - Thiết bị ESP32: terminal hiển thị “send notify AA BB”
  - Light Blue Explorer: xuất hiện thông báo “AA BB” trên giao diện ứng dụng
  - Kết quả: Đạt

- **Test for Connection Success Rate (*)**
  - Thiết bị ESP32: xuất hiện dòng “ESP_GATTS_DISCONNECT_EVT, reason = 13” khi thiết bị di động disconnect và dòng “pair status = success” khi reconnect, đã thực hiện đủ 10 lần
  - Light Blue Explorer: hiển thị thông báo đã disconnect và đã reconnect sau khi thực hiện hành động
  - Kết quả: Đạt
- **Test for Long Connection Stability**
  - Trong suốt quá trình test, kết quả vẫn ổn định, không xảy ra tình trạng tự ngắt kết nối
  - Kết quả: Đạt

4. Viết một chương trình thực hiện các công việc sau:

Quảng bá ESP32 với tên thiết bị là tên nhóm.

```
#define PROFILE_NUM 1
#define PROFILE_APP_IDX 0
#define ESP_APP_ID 0x55
#define SAMPLE_DEVICE_NAME "NHOM_7"
#define SVC_INST_ID 0
```

```
static uint8_t raw_adv_data[] = {
    /* flags */
    0x02, 0x01, 0x06,
    /* tx power*/
    0x02, 0x0a, 0xeb,
    /* service uuid */
    0x03, 0x03, 0xFF, 0x00,
    /* device name */
    0x07, 0x09, 'N', 'h', 'o', 'm', '_', '7'};
```

```
char id[9];
static void gatts_profile_event_handler(esp_gatts_cb_event_t event, esp_gatt_if_t gatts_if, esp_ble_gatts_cb_param_t *param)
{
    switch (event)
    {
    case ESP_GATTS_REG_EVT:
    {
        esp_err_t set_dev_name_ret = esp_ble_gap_set_device_name(SAMPLE_DEVICE_NAME);
        if (set_dev_name_ret)
        {
            ESP_LOGE(EXAMPLE_TAG, "set device name failed, error code = %x", set_dev_name_ret);
        }
#ifdef CONFIG_SET_RAW_ADV_DATA
        esp_err_t raw_adv_ret = esp_ble_gap_config_adv_data_raw(raw_adv_data, sizeof(raw_adv_data));
        if (raw_adv_ret)
```

```
case ESP_GATTS_READ_EVT:
    ESP_LOGE(EXAMPLE_TAG, "ESP_GATTS_READ_EVT, handle=0x%d, offset=%d", param->read.handle, param->read.offset);
    if (gatt_db_handle_table[IDX_CHAR_VAL_A] == param->read.handle)
    {
        ESP_LOGE(EXAMPLE_TAG, "(2) ***** read char1 ***** \n");
    }
    // if (gatt_db_handle_table[IDX_CHAR_VAL_B] == param->read.handle)
    // {
    //     ESP_LOGE(EXAMPLE_TAG, "(5) ***** read char2 ***** \n");
    // }
    break;
```

```
case ESP_GATTS_WRITE_EVT:
    if (!param->write.is_prep)
    {
```

```
    ESP_LOGI(EXAMPLE_TAG, "(3)***** short write success ***** \n");
    ESP_LOG_BUFFER_HEX(EXAMPLE_TAG, param->write.value, param->write.len);
    for (int i = 0; i < param->write.len; i++)
    {
        sprintf(&id[i * 2], "%02x", param->write.value[i]);
    }
    ssd1306_display_text(id);
    ssd1306_display_text("\n");
    /* if (gatt_db_handle_table[IDX_CHAR_VAL_A] == param->write.handle && param->write.len == 8)
    {
        // uint8_t write_data[2] = {0x88, 0x99};
        // if (memcmp(write_data, param->write.value, param->write.len) == 0)
        // {
        // }
        ESP_LOGI(EXAMPLE_TAG, "(3)***** short write success ***** \n");
        ESP_LOG_BUFFER_HEX(EXAMPLE_TAG, param->write.value, param->write.len);
        printf("%s\n", param->write.value);
    } */

    /* send response when param->write.need_rsp is true*/
    if (param->write.need_rsp)
    {
        esp_ble_gatts_send_response(gatts_if, param->write.conn_id, param->write.trans_id, ESP_GATT_OK, NULL);
    }
}
```

Giải thích ý nghĩa cách sử dụng hàm dựa trên ble_compatibility_test.c

Hàm gatts_profile_event_handler() được sử dụng để xử lý các sự kiện của dịch vụ BLE trên ESP32. Các sự kiện được xử lý trong switch-case bao gồm:

1. ESP_GATTS_REG_EVT: Sự kiện này được gọi khi đăng ký dịch vụ BLE thành công trên ESP32. Trong đoạn code này, ESP32 sẽ thiết lập tên thiết bị và dữ liệu quảng bá để các thiết bị khác có thể tìm thấy và kết nối tới dịch vụ.

2. ESP_GATTS_READ_EVT: Sự kiện này được gọi khi có yêu cầu đọc dữ liệu từ thiết bị điện thoại. Trong đoạn code này, nếu yêu cầu đọc dữ liệu được gửi đến từ characteristic A, ESP32 sẽ in ra thông báo "read char1" trên console.

3. ESP_GATTS_WRITE_EVT: Sự kiện này được gọi khi có yêu cầu ghi dữ liệu từ thiết bị điện thoại. Trong đoạn code này, nếu dữ liệu được ghi vào từ characteristic A có độ dài là 8 byte, ESP32 sẽ in ra dữ liệu đó trên console. Nếu dữ liệu được ghi vào từ characteristic C_2 có độ dài là 2 byte và giá trị là 0x0001, ESP32 sẽ gửi thông báo (notify) với các giá trị AA BB đến thiết bị điện thoại. Nếu giá trị là 0x0002, ESP32 sẽ gửi thông báo kèm theo xác nhận (indicate) đến thiết bị điện

thoại. Nếu giá trị là 0x0000, ESP32 sẽ tắt chức năng thông báo/indicate. Nếu dữ liệu được ghi vào là các characteristic khác, ESP32 sẽ in ra thông báo "short write success" trên console và hiển thị dữ liệu được ghi vào lên màn hình OLED. Sau đó, nếu yêu cầu ghi dữ liệu cần phản hồi (param->write.need_rsp = true), ESP32 sẽ gửi phản hồi về cho thiết bị điện thoại.

Sử dụng điều kiện biên dịch để kiểm tra xem có sử dụng dữ liệu quảng bá (raw advertising data) hay không và tương ứng cấu hình dữ liệu quảng bá. Nếu không sử dụng dữ liệu quảng bá, ESP32 sẽ cấu hình dữ liệu quảng bá thông qua các biến adv_data và scan_rsp_data.

Hàm esp_ble_gatts_create_attr_tab() để tạo bảng thuộc tính cho dịch vụ BLE. Bảng này chứa các characteristic và descriptor của dịch vụ, được định nghĩa trong mảng gatt_db[] trước đó.

Ở bài tập này nhóm sử dụng lại source của **ble_compatibility_test** với một vài chỉnh sửa để có thể hoạt động đúng theo yêu cầu. Ở hàm gatts_profile_event_handler, nhóm đã chỉnh lại code ở Event ESP_GATTS_WRITE

```
    case ESP_GATTS_WRITE_EVT:
        if (!param->write.is_prep)
        {
            ESP_LOGI(EXAMPLE_TAG, "(3)***** short write success ***** \n");
            ESP_LOG_BUFFER_HEX(EXAMPLE_TAG, param->write.value, param->write.len);
            for (int i = 0; i < param->write.len; i++)
            {
                sprintf(&id[i * 2], "%02x", param->write.value[i]);
            }
            ssd1306_display_text(id);
            ssd1306_display_text("\n");

            /* send response when param->write.need_rsp is true*/
            if (param->write.need_rsp)
            {
                esp_ble_gatts_send_response(gatts_if, param->write.conn_id, param->write.trans_id, ESP_GATT_OK, NULL);
            }
        }
```

Sau khi nhận được message từ điện thoại, sẽ tiến hành lấy dữ liệu đã ghi để giải mã thành một chuỗi kí tự để in ra màn hình (dạng dữ liệu ở dạng hex).

Video câu 2:

https://drive.google.com/file/d/14ho18qNnG5UlEwzyAZIIs7kcSouKVHtb/view?fbclid=IwAR2YJP2dvcVlrcmd6dJM24NixDy_ZDmEJKX4wMNs3lOsq3buBUg6i5EO21I&pli=1

Video câu 4: https://youtu.be/cp93oTRacJM

Source code: https://github.com/otaros/CE232_Lab/tree/main/Lab3