به نام خدا

دانشگاه صنعتی شریف

دانشکده مهندسی برق

استاد درس: دکتر احمدی

دانشجو: محمد غفوریان | شماره دانشجویی: ۹۹۱۰۶۴۹۳

در کد زیر با گرفتن متن ON و LED OFF که داخل برد تعبیه شده(به پین دو وصل هست) را روشن و خاموش میکنیم

```
static int device_write(uint16_t conn_handle, uint16_t attr_handle, struct ble_gatt_access_ctxt *ctxt, void *arg)
{
    // printf("Data from the client: %.*s\n", ctxt->om->om_len, ctxt->om->om_data);

    char *data = (char *)ctxt->om->om_data;
    //printf("%d %s\n", strcmp(data, (char *)"ON") == 0, data);
    if (strcmp(data, (char *)"ON") == 0)
    {
        printf("ON1\n");
        gpio_set_level(GPIO_NUM_2, 1);
    }
    else if (strcmp(data, (char *)"OFF") == 0)
    {
        printf("OFF0\n");
        gpio_set_level(GPIO_NUM_2, 0);
    }

    memset(data, 0, strlen(data));


    return 0;
}
```

این یک دستگاه عملکردی استاتیک را تعریف می کند که چهار آرگومان را می گیرد uint16_t con_handle :، uint16_t attr_handle، struktur ble_gatt_access_ctxt *ctxt، و void *arg. این تابع رشته " DATA FROM SERVER" را به قسمت OM آرگومان CTXT اضافه می کند و ۰ را برمی گرداند.

این کد همچنین مجموعه ای از نشانگرها را به سایر تعاریف خدمات به نام GATT_SVCS تعریف می کند. آرایه شامل یک عنصر از نوع ساختار ble_gatt_svc_def است. این عنصر دارای سه قسمت است: نوع ، UUID و خصوصیات. قسمت نوع روی ble_gatt_svc_type_primary تنظیم شده است. قسمت UUID روی (0x180) ble_uuid16_declare تنظیم شده است. قسمت ویژگی ها مجموعه ای از دو عنصر از نوع ساختار ble_gatt_chr_def است. عنصر اول دارای سه قسمت است: UUID، FLAGS و .ACCESS_CB قسمت UUID روی (0xfef4) ble_uuid16_declare تنظیم شده است. قسمت Flags روی BLE_GATT_CHR_F_READ تنظیم شده است. قسمت Access_CB روی دستگاه تنظیم شده است. عنصر دوم دارای همان زمینه ها است ، به جز اینکه قسمت UUID روی (0xDead) BLE_UUID16_DECLARE تنظیم شده است و قسمت Flags روی BLE_GATT_CHR_F_WRITE تنظیم شده است. آرایه توسط یک عنصر تهی خاتمه می یابد.

```c
// Read data from ESP32 defined as server
static int device_read(uint16_t con_handle, uint16_t attr_handle, struct ble_gatt_access_
{
    os_mbuf_append(ctxt->om, "Data from the server", strlen("Data from the server"));
    return 0;
}

// Array of pointers to other service definitions
// UUID - Universal Unique Identifier
static const struct ble_gatt_svc_def gatt_svcs[] = {
    {.type = BLE_GATT_SVC_TYPE_PRIMARY,
     .uuid = BLE_UUID16_DECLARE(0x180),                 // Define UUID for device type
     .characteristics = (struct ble_gatt_chr_def[]){
         {.uuid = BLE_UUID16_DECLARE(0xFEF4),           // Define UUID for reading
          .flags = BLE_GATT_CHR_F_READ,
          .access_cb = device_read},
         {.uuid = BLE_UUID16_DECLARE(0xDEAD),           // Define UUID for writing
          .flags = BLE_GATT_CHR_F_WRITE,
          .access_cb = device_write},
         {0}}},
    {0}};
```

این یک تابع استاتیک BLE_GAP_EVENT را تعریف می کند که دو آرگومان را می گیرد: ساختار ble_gap_event *event and void *arg این تابع حاوی یک عبارت سوئیچ است که انواع مختلفی از رویدادها را کنترل می کند. اگر نوع رویداد BLE_GAP_EVENT_CONNECT باشد ، این عملکرد پیام را به کنسول وارد می کند که نشان می دهد اتصال موفقیت آمیز بوده است یا خیر. اگر اتصال ناموفق بود ، لاگ () BLE_APP_ADVERTISE را صدا می کند. اگر نوع رویداد BLE_GAP_EVENT_DISCONNECT باشد ، این لاگ پیام را به کنسول وارد می کند. اگر نوع رویداد BLE_GAP_EVENT_ADV_COMPLETE باشد ، لاگ پیام را به کنسول وارد می کند و () ble_app_advertise را صدا می کند.

```c
// BLE event handling
static int ble_gap_event(struct ble_gap_event *event, void *arg)
{
    switch (event->type)
    {
    // Advertise if connected
    case BLE_GAP_EVENT_CONNECT:
        ESP_LOGI("GAP", "BLE GAP EVENT CONNECT %s", event->connect.status == 0 ? "OK!" : "FAILED!");
        if (event->connect.status != 0)
        {
            ble_app_advertise();
        }
        break;
    // Advertise again after completion of the event
    case BLE_GAP_EVENT_DISCONNECT:
        ESP_LOGI("GAP", "BLE GAP EVENT DISCONNECTED");
        break;
    case BLE_GAP_EVENT_ADV_COMPLETE:
        ESP_LOGI("GAP", "BLE GAP EVENT");
        ble_app_advertise();
        break;
    default:
        break;
    }
    return 0;
}
```

این یک تابع BLE_APP_ADVERTISE را تعریف می کند که هیچ آرگومانی را نمی گیرد و هیچ ارزشی را برنگرداند. این تابع با تعریف زمینه های ساختار از نوع ساختار ble_hs_adv_fields شروع می شود. سپس ساختار زمینه ها را با استفاده از MEMSET به همه صفرها آغاز می کند. این تابع نام دستگاه BLE را با استفاده از ble_svc_gap_device_name خوانده و آن را به متغیر دستگاه_Name اختصاص می دهد. زمینه نام ساختار فیلدها بر روی متغیر دستگاه uint8_t بر روی یک متغیر

uint8_t تنظیم شده است. قسمت name_len روی طول رشته دستگاه_name تنظیم شده است و قسمت name_is_complete روی ۱ تنظیم شده است.

سپس عملکرد یک ساختار adv_params از نوع ساختار ble_gap_adv_params را تعریف می کند. این ساختار ADV_PARAMS را با استفاده از MEMSET به همه صفرها آغاز می کند. قسمت Conn_Mode از ساختار ADV_PARAMS روی BLE_GAP_CONN_MODE_UND تنظیم شده است ، به این معنی که دستگاه می تواند قابل اتصال باشد یا غیر قابل اتصال باشد. قسمت DISC_MODE از ساختار ADV_PARAMS روی BLE_GAP_DISC_MODE_GEN تنظیم شده است ، به این معنی که دستگاه قابل کشف است.

در آخر ، این عملکرد برای شروع ادورتایز دستگاه BLE_GAP_ADV_START تماس می گیرد. این تابع از متغیر ble_addr_type عبور می کند ، که نوع آدرس استفاده شده برای ADVERTIZING را مشخص می کند ، و همچنین برای خود آدرس خالی می کند. این عملکرد همچنین برای مدت زمان ADVERTIZING از ble_hs_forever عبور می کند ، به این معنی که دستگاه به طور نامحدود ADVERTIZE می کند. این تابع یک نشانگر را به ساختار ADV_PARAMS منتقل می کند ، که پارامترهای ADVERTIZING را مشخص می کند ، و همچنین یک اشارگر به عملکرد BLE_GAP_EVENT ، که هنگام وقوع یک رویداد توسط پشته Ble خوانده می شود. این تابع همچنین برای پارامتر ARG NULL عبور می کند ، که توسط عملکرد BLE_GAP_EVENT استفاده نمی شود.

```c
// Define the BLE connection
void ble_app_advertise(void)
{
    // GAP - device name definition
    struct ble_hs_adv_fields fields;
    const char *device_name;
    memset(&fields, 0, sizeof(fields));
    device_name = ble_svc_gap_device_name(); // Read the BLE device name
    fields.name = (uint8_t *)device_name;
    fields.name_len = strlen(device_name);
    fields.name_is_complete = 1;
    ble_gap_adv_set_fields(&fields);

    // GAP - device connectivity definition
    struct ble_gap_adv_params adv_params;
    memset(&adv_params, 0, sizeof(adv_params));
    adv_params.conn_mode = BLE_GAP_CONN_MODE_UND; // connectable or non-connectable
    adv_params.disc_mode = BLE_GAP_DISC_MODE_GEN; // discoverable or non-discoverable
    ble_gap_adv_start(ble_addr_type, NULL, BLE_HS_FOREVER, &adv_params, ble_gap_event, NULL);
}
```

در این کد لاگ با تماس با ble_hs_id_infer_auto شروع می شود تا بهترین نوع آدرس را به صورت خودکار تعیین کنید. سپس برای تعریف اتصال BLE با ble_app_advertise تماس می گیرد.

کد همچنین یک تابع host_task را تعریف می کند که یک آرگومان VOID * را به خود اختصاص می دهد و هیچ مقداری را برنمی گرداند. این تابع ، تابع nimble_port_run را صدا می کند. این تابع فقط در صورت اجرای nimble_port_stop مقدار برمی گرداند.

در آخر لاگ با تنظیم جهت GPIO PIN 2 برای خروجی شروع می شود. سپس Flash NVS را با استفاده از NVS_FLASH_INIT آغاز می کند. این کنترلر ESP را با استفاده از ESP_NIMBLE_HCI_AND_CONTROLLER_INIT شروع می کند. پشته میزبان را با استفاده از nimble_port_init آغاز می کند. این نام سرور BLE را با استفاده از ble_svc_gap_device_name_set تنظیم می کند. سرویس GAP را با استفاده از BLE_SVC_GAP_INIT شروع می کند. ان سرویس GATT را با استفاده از BLE_SVC_GATT_INIT آغاز می کند. این سرویس های GATT را با استفاده از ble_gatts_count_cfg و ble_gatts_add_svcs پیکربندی می کند. این لاگ پاسخگویی SYNC را با استفاده از ble_hs_cfg.sync_cb تنظیم می کند. سرانجام ، این موضوع را با استفاده از nimble_port_freertos_init اجرا می کند.

```c
// The application
void ble_app_on_sync(void)
{
    ble_hs_id_infer_auto(0, &ble_addr_type); // Determines the best address type automatically
    ble_app_advertise();                     // Define the BLE connection
}

// The infinite task
void host_task(void *param)
{
    nimble_port_run(); // This function will return only when nimble_port_stop() is executed
}

void app_main()
{
    gpio_set_direction(GPIO_NUM_2, GPIO_MODE_OUTPUT);
    nvs_flash_init();                          // 1 - Initialize NVS flash using
    esp_nimble_hci_and_controller_init();      // 2 - Initialize ESP controller
    nimble_port_init();                        // 3 - Initialize the host stack
    ble_svc_gap_device_name_set("BLE-Server"); // 4 - Initialize NimBLE configuration - server name
    ble_svc_gap_init();                        // 4 - Initialize NimBLE configuration - gap service
    ble_svc_gatt_init();                       // 4 - Initialize NimBLE configuration - gatt service
    ble_gatts_count_cfg(gatt_svcs);            // 4 - Initialize NimBLE configuration - config gatt services
    ble_gatts_add_svcs(gatt_svcs);             // 4 - Initialize NimBLE configuration - queues gatt services.
    ble_hs_cfg.sync_cb = ble_app_on_sync;      // 5 - Initialize application
    nimble_port_freertos_init(host_task);      // 6 - Run the thread
}
```