Doan Viet Tu – 1952521

EMBEDDED SYSTEMS – CC01
LAB 3
ESP

ESP-L05

Source code:

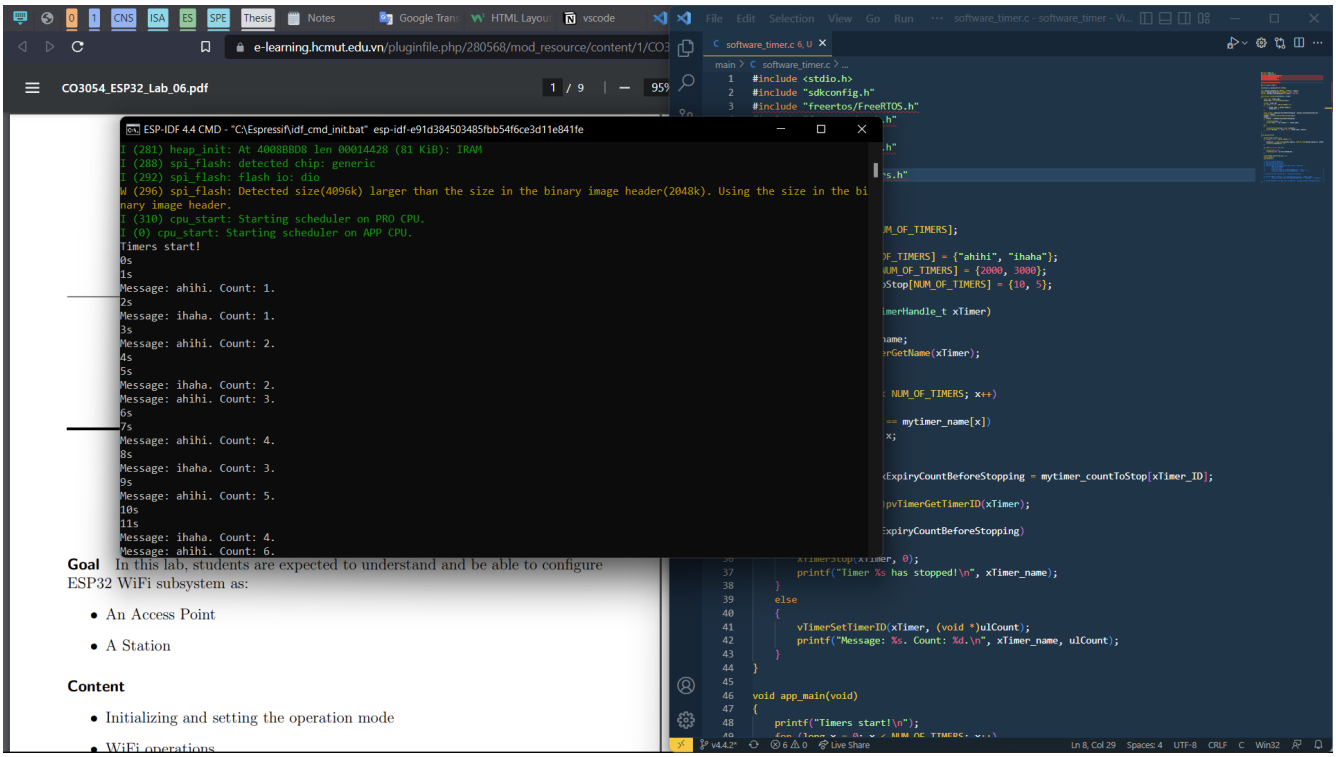
main > software_timer.c > ...

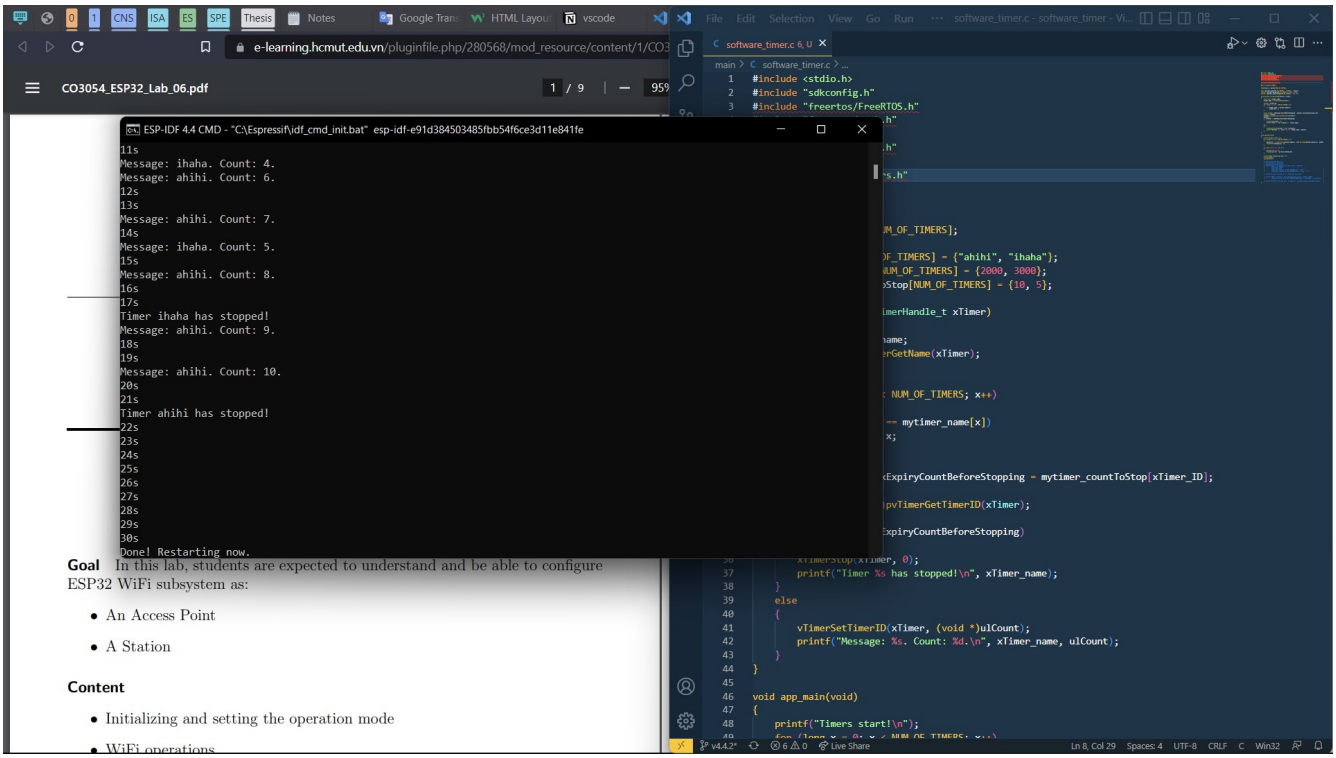
```c
#include <stdio.h>
#include "sdkconfig.h"
#include "freertos/FreeRTOS.h"
#include "freertos/task.h"
#include "esp_system.h"
#include "esp_spi_flash.h"

#include "freertos/timers.h"

#define NUM_OF_TIMERS 2

TimerHandle_t mytimer[NUM_OF_TIMERS];

char *mytimer_name[NUM_OF_TIMERS] = {"ahihi", "ihaha"};
uint32_t mytimer_delay[NUM_OF_TIMERS] = {2000, 3000};
uint16_t mytimer_countToStop[NUM_OF_TIMERS] = {10, 5};

void mytimer_callback(TimerHandle_t xTimer)
{
    const char *xTimer_name;
    xTimer_name = pcTimerGetName(xTimer);

    uint16_t xTimer_ID;
    for (long x = 0; x < NUM_OF_TIMERS; x++)
    {
        if (xTimer_name == mytimer_name[x])
            xTimer_ID = x;
    }

    const uint32_t ulMaxExpiryCountBeforeStopping = mytimer_countToStop[xTimer_ID];
    uint32_t ulCount;
    ulCount = (uint32_t)pvTimerGetTimerID(xTimer);
    ulCount++;
    if (ulCount > ulMaxExpiryCountBeforeStopping)
    {
        xTimerStop(xTimer, 0);
        printf("Timer %s has stopped!\n", xTimer_name);
    }
    else
    {
        vTimerSetTimerID(xTimer, (void *)ulCount);
        printf("Message: %s. Count: %d.\n", xTimer_name, ulCount);
    }
}

void app_main(void)
{
    printf("Timers start!\n");
```

main > software_timer.c > ...

```c
        vTimerSetTimerID(xTimer, (void *)ulCount);
        printf("Message: %s. Count: %d.\n", xTimer_name, ulCount);
    }
}

void app_main(void)
{
    printf("Timers start!\n");
    for (long x = 0; x < NUM_OF_TIMERS; x++)
    {
        mytimer[x] = xTimerCreate(mytimer_name[x], pdMS_TO_TICKS(mytimer_delay[x]), pdTRUE,
        xTimerStart(mytimer[x], 0);
    }

    for (int i = 0; i <= 30; i++)
    {
        printf("%ds\n", i);
        vTaskDelay(1000 / portTICK_PERIOD_MS);
    }

    printf("Done! Restarting now.\n");
    fflush(stdout);
    esp_restart();

    /* Print chip information */
    // esp_chip_info_t chip_info;
    // esp_chip_info(&chip_info);
    // printf("This is %s chip with %d CPU core(s), WiFi%s%s, ",
    //         CONFIG_IDF_TARGET,
    //         chip_info.cores,
    //         (chip_info.features & CHIP_FEATURE_BT) ? "/BT" : "",
    //         (chip_info.features & CHIP_FEATURE_BLE) ? "/BLE" : "");

    // printf("silicon revision %d, ", chip_info.revision);

    // printf("%dMB %s flash\n", spi_flash_get_chip_size() / (1024 * 1024),
    //         (chip_info.features & CHIP_FEATURE_EMB_FLASH) ? "embedded" : "external");

    // printf("Minimum free heap size: %d bytes\n", esp_get_minimum_free_heap_size());
}
```

Results:





ESP-L06

a) As a Station:

Source code:

main > C wifi_station.c > wifi_init_sta(void)

```c
#include <string.h>
#include "freertos/FreeRTOS.h"
#include "freertos/task.h"
#include "freertos/event_groups.h"
#include "esp_system.h"
#include "esp_wifi.h"
#include "esp_event.h"
#include "esp_log.h"
#include "nvs_flash.h"

#include "lwip/err.h"
#include "lwip/sys.h"

#define TARGET_ESP_WIFI_SSID CONFIG_ESP_WIFI_SSID
#define TARGET_ESP_WIFI_PASSWORD CONFIG_ESP_WIFI_PASSWORD
#define ESP_MAXIMUM_ENTRY CONFIG_ESP_MAXIMUM_RETRY

#if CONFIG_ESP_WIFI_AUTH_OPEN
#define ESP_WIFI_SCAN_AUTH_MODE_THRESHOLD WIFI_AUTH_OPEN
#elif CONFIG_ESP_WIFI_AUTH_WEP
#define ESP_WIFI_SCAN_AUTH_MODE_THRESHOLD WIFI_AUTH_WEP
#elif CONFIG_ESP_WIFI_AUTH_WPA_PSK
#define ESP_WIFI_SCAN_AUTH_MODE_THRESHOLD WIFI_AUTH_WPA_PSK
#elif CONFIG_ESP_WIFI_AUTH_WPA2_PSK
#define ESP_WIFI_SCAN_AUTH_MODE_THRESHOLD WIFI_AUTH_WPA2_PSK
#elif CONFIG_ESP_WIFI_AUTH_WPA_WPA2_PSK
#define ESP_WIFI_SCAN_AUTH_MODE_THRESHOLD WIFI_AUTH_WPA_WPA2_PSK
#elif CONFIG_ESP_WIFI_AUTH_WPA3_PSK
#define ESP_WIFI_SCAN_AUTH_MODE_THRESHOLD WIFI_AUTH_WPA3_PSK
#elif CONFIG_ESP_WIFI_AUTH_WPA2_WPA3_PSK
#define ESP_WIFI_SCAN_AUTH_MODE_THRESHOLD WIFI_AUTH_WPA2_WPA3_PSK
#elif CONFIG_ESP_WIFI_AUTH_WAPI_PSK
#define ESP_WIFI_SCAN_AUTH_MODE_THRESHOLD WIFI_AUTH_WAPI_PSK
#endif

static EventGroupHandle_t s_wifi_event_group;
#define WIFI_CONNECTED_BIT BIT0
#define WIFI_FAIL_BIT BIT1

static const char *TAG = "wifi station";
static int s_retry_num = 0;

static void event_handler(void *arg, esp_event_base_t event_base,
                          int32_t event_id, void *event_data)
{
    if (event_base == WIFI_EVENT && event_id == WIFI_EVENT_STA_START)
    {
        esp_wifi_connect();
    }
```

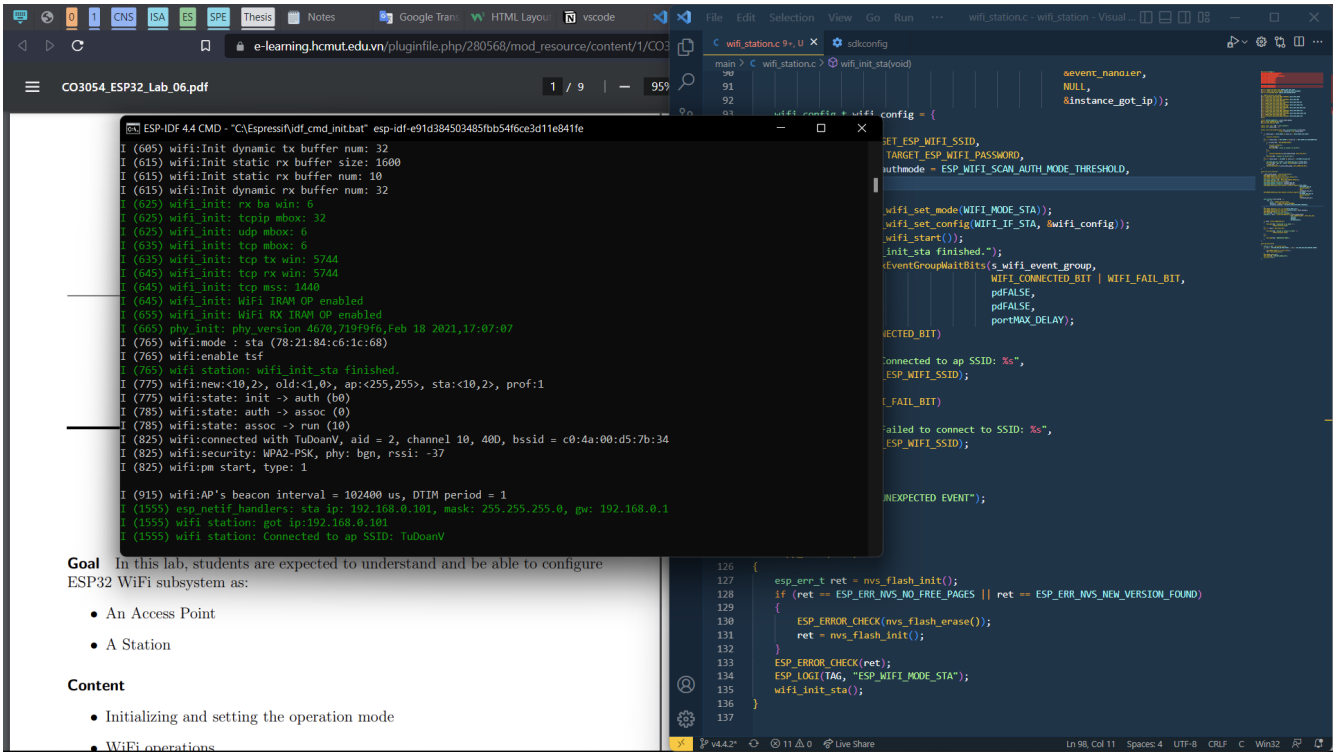
main > C wifi_station.c > wifi_init_sta(void)

```c
        esp_wifi_connect();
    }
    else if (event_base == WIFI_EVENT && event_id == WIFI_EVENT_STA_DISCONNECTED)
    {
        if (s_retry_num < ESP_MAXIMUM_ENTRY)
        {
            esp_wifi_connect();
            s_retry_num++;
            ESP_LOGI(TAG, "retry to connect to the AP");
        }
        else
        {
            xEventGroupSetBits(s_wifi_event_group, WIFI_FAIL_BIT);
        }
        ESP_LOGI(TAG, "connect to the AP fail");
    }
    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);
    }
}

void wifi_init_sta(void)
{
    s_wifi_event_group = xEventGroupCreate();
    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));
    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,
                                                        &event_handler,
                                                        NULL,
                                                        &instance_any_id));
    ESP_ERROR_CHECK(esp_event_handler_instance_register(IP_EVENT,
                                                        IP_EVENT_STA_GOT_IP,
                                                        &event_handler,
                                                        NULL,
                                                        &instance_got_ip));
    wifi_config_t wifi_config = {
        .sta = {
            .ssid = TARGET_ESP_WIFI_SSID,
```

main > C wifi_station.c > wifi_init_sta(void)

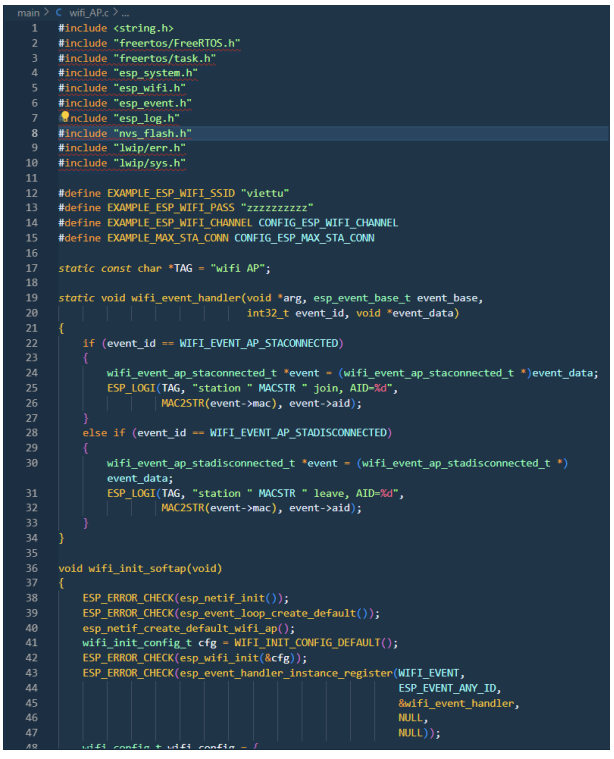
```c
            .sta = {
                .ssid = TARGET_ESP_WIFI_SSID,
                .password = TARGET_ESP_WIFI_PASSWORD,
                .threshold.authmode = ESP_WIFI_SCAN_AUTH_MODE_THRESHOLD,
            },
        };
    ESP_ERROR_CHECK(esp_wifi_set_mode(WIFI_MODE_STA));
    ESP_ERROR_CHECK(esp_wifi_set_config(WIFI_IF_STA, &wifi_config));
    ESP_ERROR_CHECK(esp_wifi_start());
    ESP_LOGI(TAG, "wifi_init_sta finished.");
    EventBits_t bits = xEventGroupWaitBits(s_wifi_event_group,
                                           WIFI_CONNECTED_BIT | WIFI_FAIL_BIT,
                                           pdFALSE,
                                           pdFALSE,
                                           portMAX_DELAY);
    if (bits & WIFI_CONNECTED_BIT)
    {
        ESP_LOGI(TAG, "Connected to ap SSID: %s",
                 TARGET_ESP_WIFI_SSID);
    }
    else if (bits & WIFI_FAIL_BIT)
    {
        ESP_LOGI(TAG, "Failed to connect to SSID: %s",
                 TARGET_ESP_WIFI_SSID);
    }
    else
    {
        ESP_LOGE(TAG, "UNEXPECTED EVENT");
    }
}

void app_main(void)
{
    esp_err_t ret = nvs_flash_init();
    if (ret == ESP_ERR_NVS_NO_FREE_PAGES || ret == ESP_ERR_NVS_NEW_VERSION_FOUND)
    {
        ESP_ERROR_CHECK(nvs_flash_erase());
        ret = nvs_flash_init();
    }
    ESP_ERROR_CHECK(ret);
    ESP_LOGI(TAG, "ESP_WIFI_MODE_STA");
    wifi_init_sta();
}
```

Results:



b) As an Access Point:

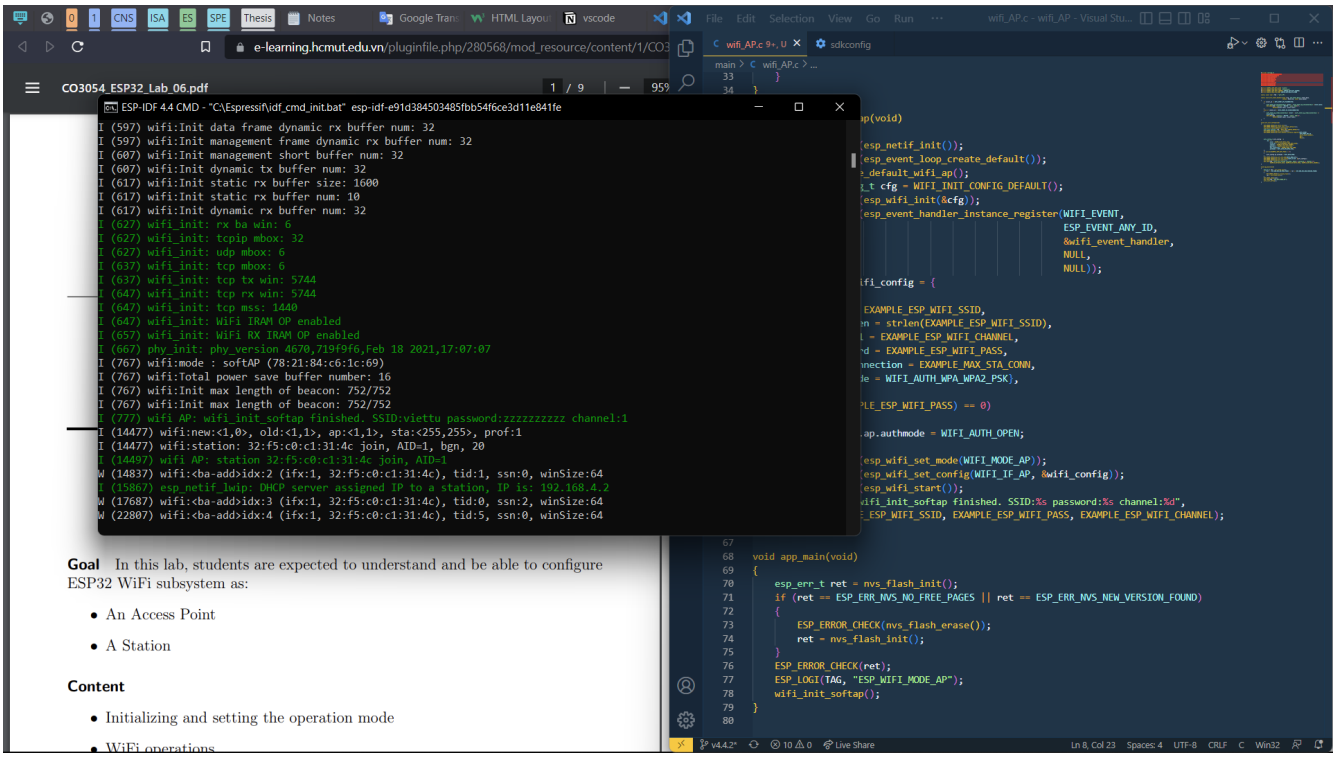
Source code:



```c
#include <string.h>
#include "freertos/FreeRTOS.h"
#include "freertos/task.h"
#include "esp_system.h"
#include "esp_wifi.h"
#include "esp_event.h"
#include "esp_log.h"
#include "nvs_flash.h"
#include "lwip/err.h"
#include "lwip/sys.h"

#define EXAMPLE_ESP_WIFI_SSID "viettu"
#define EXAMPLE_ESP_WIFI_PASS "zzzzzzzzzz"
#define EXAMPLE_ESP_WIFI_CHANNEL CONFIG_ESP_WIFI_CHANNEL
#define EXAMPLE_MAX_STA_CONN CONFIG_ESP_MAX_STA_CONN

static const char *TAG = "wifi AP";

static void wifi_event_handler(void *arg, esp_event_base_t event_base,
                               int32_t event_id, void *event_data)
{
    if (event_id == WIFI_EVENT_AP_STACONNECTED)
    {
        wifi_event_ap_staconnected_t *event = (wifi_event_ap_staconnected_t *)event_data;
        ESP_LOGI(TAG, "station " MACSTR " join, AID=%d",
                 MAC2STR(event->mac), event->aid);
    }
    else if (event_id == WIFI_EVENT_AP_STADISCONNECTED)
    {
        wifi_event_ap_stadisconnected_t *event = (wifi_event_ap_stadisconnected_t *)
        event_data;
        ESP_LOGI(TAG, "station " MACSTR " leave, AID=%d",
                 MAC2STR(event->mac), event->aid);
    }
}

void wifi_init_softap(void)
{
    ESP_ERROR_CHECK(esp_netif_init());
    ESP_ERROR_CHECK(esp_event_loop_create_default());
    esp_netif_create_default_wifi_ap();
    wifi_init_config_t cfg = WIFI_INIT_CONFIG_DEFAULT();
    ESP_ERROR_CHECK(esp_wifi_init(&cfg));
    ESP_ERROR_CHECK(esp_event_handler_instance_register(WIFI_EVENT,
                                                        ESP_EVENT_ANY_ID,
                                                        &wifi_event_handler,
                                                        NULL,
                                                        NULL));
```

```c
    esp_netif_create_default_wifi_ap();
    wifi_init_config_t cfg = WIFI_INIT_CONFIG_DEFAULT();
    ESP_ERROR_CHECK(esp_wifi_init(&cfg));
    ESP_ERROR_CHECK(esp_event_handler_instance_register(WIFI_EVENT,
                                                        ESP_EVENT_ANY_ID,
                                                        &wifi_event_handler,
                                                        NULL,
                                                        NULL));
    wifi_config_t wifi_config = {
        .ap = {
            .ssid = EXAMPLE_ESP_WIFI_SSID,
            .ssid_len = strlen(EXAMPLE_ESP_WIFI_SSID),
            .channel = EXAMPLE_ESP_WIFI_CHANNEL,
            .password = EXAMPLE_ESP_WIFI_PASS,
            .max_connection = EXAMPLE_MAX_STA_CONN,
            .authmode = WIFI_AUTH_WPA_WPA2_PSK},
    };
    if (strlen(EXAMPLE_ESP_WIFI_PASS) == 0)
    {
        wifi_config.ap.authmode = WIFI_AUTH_OPEN;
    }
    ESP_ERROR_CHECK(esp_wifi_set_mode(WIFI_MODE_AP));
    ESP_ERROR_CHECK(esp_wifi_set_config(WIFI_IF_AP, &wifi_config));
    ESP_ERROR_CHECK(esp_wifi_start());
    ESP_LOGI(TAG, "wifi_init_softap finished. SSID:%s password:%s channel:%d",
             EXAMPLE_ESP_WIFI_SSID, EXAMPLE_ESP_WIFI_PASS, EXAMPLE_ESP_WIFI_CHANNEL);
}

void app_main(void)
{
    esp_err_t ret = nvs_flash_init();
    if (ret == ESP_ERR_NVS_NO_FREE_PAGES || ret == ESP_ERR_NVS_NEW_VERSION_FOUND)
    {
        ESP_ERROR_CHECK(nvs_flash_erase());
        ret = nvs_flash_init();
    }
    ESP_ERROR_CHECK(ret);
    ESP_LOGI(TAG, "ESP_WIFI_MODE_AP");
    wifi_init_softap();
}
```

Results:


ESP-IDF 4.4 CMD - "C:\Espressif\idf_cmd_init.bat" esp-idf-e91d384503485fbb54f6ce3d11e841fe
I (597) wifi:Init data frame dynamic rx buffer num: 32
I (597) wifi:Init management frame dynamic rx buffer num: 32
I (607) wifi:Init management short buffer num: 32
I (607) wifi:Init dynamic tx buffer num: 32
I (617) wifi:Init static rx buffer size: 1600
I (617) wifi:Init static rx buffer num: 10
I (617) wifi:Init dynamic rx buffer num: 32
I (627) wifi_init: rx ba win: 6
I (627) wifi_init: tcpip mbox: 32
I (627) wifi_init: udp mbox: 6
I (637) wifi_init: tcp mbox: 6
I (637) wifi_init: tcp tx win: 5744
I (647) wifi_init: tcp rx win: 5744
I (647) wifi_init: tcp mss: 1440
I (647) wifi_init: WiFi IRAM OP enabled
I (657) wifi_init: WiFi RX IRAM OP enabled
I (667) phy_init: phy_version 4670,719f9f6,Feb 18 2021,17:07:07
I (767) wifi:mode : softAP (78:21:84:c6:1c:69)
I (767) wifi:Total power save buffer number: 16
I (767) wifi:Init max length of beacon: 752/752
I (767) wifi:Init max length of beacon: 752/752
I (777) wifi AP: wifi_init_softap finished. SSID:viettu password:zzzzzzzzz channel:1
I (14477) wifi:new:<1,0>, old:<1,1>, ap:<1,1>, sta:<255,255>, prof:1
I (14477) wifi:station: 32:f5:c0:c1:31:4c join, AID=1, bgn, 20
I (14497) wifi AP: station 32:f5:c0:c1:31:4c join, AID=1
W (14837) wifi:<ba-add>idx:2 (ifx:1, 32:f5:c0:c1:31:4c), tid:1, ssn:0, winSize:64
I (15867) esp_netif_lwip: DHCP server assigned IP to a station, IP is: 192.168.4.2
W (17687) wifi:<ba-add>idx:3 (ifx:1, 32:f5:c0:c1:31:4c), tid:0, ssn:2, winSize:64
W (22807) wifi:<ba-add>idx:4 (ifx:1, 32:f5:c0:c1:31:4c), tid:5, ssn:0, winSize:64
CO3054_ESP32_Lab_06.pdf
Goal In this lab, students are expected to understand and be able to configure ESP32 WiFi subsystem as:
An Access Point
A Station
Content
Initializing and setting the operation mode
WiFi operations
void app_main(void)
{
esp_err_t ret = nvs_flash_init();
if (ret == ESP_ERR_NVS_NO_FREE_PAGES || ret == ESP_ERR_NVS_NEW_VERSION_FOUND)
{
ESP_ERROR_CHECK(nvs_flash_erase());
ret = nvs_flash_init();
}
ESP_ERROR_CHECK(ret);
ESP_LOGI(TAG, "ESP_WIFI_MODE_AP");
wifi_init_softap();
}