

EMBEDDED SYSTEMS – CC01

LAB 3

ESP

ESP-L05

Source code:

```
main > C software_timer.c > ...
1 #include <stdio.h>
2 #include "sdkconfig.h"
3 #include "freertos/FreeRTOS.h"
4 #include "freertos/task.h"
5 #include "esp_system.h"
6 #include "esp_spi_flash.h"
7
8 #include "freertos/timers.h"
9
10 #define NUM_OF_TIMERS 2
11
12 TimerHandle_t mytimer[NUM_OF_TIMERS];
13
14 char *mytimer_name[NUM_OF_TIMERS] = {"ahihi", "ihaha"};
15 uint32_t mytimer_delay[NUM_OF_TIMERS] = {2000, 3000};
16 uint16_t mytimer_countToStop[NUM_OF_TIMERS] = {10, 5};
17
18 void mytimer_callback(TimerHandle_t xTimer)
19 {
20     const char *xTimer_name;
21     xTimer_name = pcTimerGetName(xTimer);
22
23     uint16_t xTimer_ID;
24     for (long x = 0; x < NUM_OF_TIMERS; x++)
25     {
26         if (xTimer_name == mytimer_name[x])
27             xTimer_ID = x;
28     }
29
30     const uint32_t ulMaxExpiryCountBeforeStopping = mytimer_countToStop[xTimer_ID];
31     uint32_t ulCount;
32     ulCount = (uint32_t)pvTimerGetTimerID(xTimer);
33     ulCount++;
34     if (ulCount > ulMaxExpiryCountBeforeStopping)
35     {
36         xTimerStop(xTimer, 0);
37         printf("Timer %s has stopped!\n", xTimer_name);
38     }
39     else
40     {
41         vTimerSetTimerID(xTimer, (void *)ulCount);
42         printf("Message: %s. Count: %d.\n", xTimer_name, ulCount);
43     }
44 }
45
46 void app_main(void)
47 {
48     printf("Timers start!\n");
49
50     vTimerSetTimerID(xTimer, (void *)ulCount);
51     printf("Message: %s. Count: %d.\n", xTimer_name, ulCount);
52 }
53
54 void app_main(void)
55 {
56     printf("Timers start!\n");
57     for (long x = 0; x < NUM_OF_TIMERS; x++)
58     {
59         mytimer[x] = xTimerCreate(mytimer_name[x], pdMS_TO_TICKS(mytimer_delay[x]), pdTRUE,
60             xTimerStart(mytimer[x], 0);
61     }
62
63     for (int i = 0; i <= 30; i++)
64     {
65         printf("%ds\n", i);
66         vTaskDelay(1000 / portTICK_PERIOD_MS);
67     }
68
69     printf("Done! Restarting now.\n");
70     fflush(stdout);
71     esp_restart();
72
73     /* Print chip information */
74     // esp_chip_info_t chip_info;
75     // esp_chip_info(&chip_info);
76     // printf("This is %s chip with %d CPU core(s), WiFi%s, ",
77     //     CONFIG_IDF_TARGET,
78     //     chip_info.cores,
79     //     (chip_info.features & CHIP_FEATURE_BT) ? "/BT" : "",
80     //     (chip_info.features & CHIP_FEATURE_BLE) ? "/BLE" : "");
81
82     // printf("silicon revision %d, ", chip_info.revision);
83
84     // printf("%dMB %s flash\n", spi_flash_get_chip_size() / (1024 * 1024),
85     //     (chip_info.features & CHIP_FEATURE_EMB_FLASH) ? "embedded" : "external");
86
87     // printf("Minimum free heap size: %d bytes\n", esp_get_minimum_free_heap_size());
88 }
```

Results:

Source code:

```
main > C wif_station.c > wif_init_sta(void)
1 #include <string.h>
2 #include "freertos/FreeRTOS.h"
3 #include "freertos/task.h"
4 #include "freertos/event_groups.h"
5 #include "esp_system.h"
6 #include "esp_wifi.h"
7 #include "esp_event.h"
8 #include "esp_log.h"
9 #include "nvs_flash.h"
10
11 #include "lwip/err.h"
12 #include "lwip/sys.h"
13
14 #define TARGET_ESP_WIFI_SSID CONFIG_ESP_WIFI_SSID
15 #define TARGET_ESP_WIFI_PASSWORD CONFIG_ESP_WIFI_PASSWORD
16 #define ESP_MAXIMUM_RETRY CONFIG_ESP_MAXIMUM_RETRY
17
18 #if CONFIG_ESP_WIFI_AUTH_OPEN
19 #define ESP_WIFI_SCAN_AUTH_MODE_THRESHOLD WIFI_AUTH_OPEN
20 #elif CONFIG_ESP_WIFI_AUTH_WEP
21 #define ESP_WIFI_SCAN_AUTH_MODE_THRESHOLD WIFI_AUTH_WEP
22 #elif CONFIG_ESP_WIFI_AUTH_WPA_PSK
23 #define ESP_WIFI_SCAN_AUTH_MODE_THRESHOLD WIFI_AUTH_WPA_PSK
24 #elif CONFIG_ESP_WIFI_AUTH_WPA2_PSK
25 #define ESP_WIFI_SCAN_AUTH_MODE_THRESHOLD WIFI_AUTH_WPA2_PSK
26 #elif CONFIG_ESP_WIFI_AUTH_WPA3_PSK
27 #define ESP_WIFI_SCAN_AUTH_MODE_THRESHOLD WIFI_AUTH_WPA3_PSK
28 #elif CONFIG_ESP_WIFI_AUTH_WPA2_WPA3_PSK
29 #define ESP_WIFI_SCAN_AUTH_MODE_THRESHOLD WIFI_AUTH_WPA2_WPA3_PSK
30 #elif CONFIG_ESP_WIFI_AUTH_WPA3_PSK
31 #define ESP_WIFI_SCAN_AUTH_MODE_THRESHOLD WIFI_AUTH_WPA3_PSK
32 #elif CONFIG_ESP_WIFI_AUTH_WAPI_PSK
33 #define ESP_WIFI_SCAN_AUTH_MODE_THRESHOLD WIFI_AUTH_WAPI_PSK
34 #endif
35
36 static EventGroupHandle_t s_wifi_event_group;
37 #define WIFI_CONNECTED_BIT BIT0
38 #define WIFI_FAIL_BIT BIT1
39
40 static const char *TAG = "wifi station";
41 static int s_retry_num = 0;
42
43 static void event_handler(void *arg, esp_event_base_t event_base,
44                          int32_t event_id, void *event_data)
45 {
46     if (event_base == WIFI_EVENT && event_id == WIFI_EVENT_STA_START)
47     {
48         esp_wifi_connect();
49     }
50 }
```

```
main > C wif_station.c > wif_init_sta(void)
47 {
48     esp_wifi_connect();
49 }
50 else if (event_base == WIFI_EVENT && event_id == WIFI_EVENT_STA_DISCONNECTED)
51 {
52     if (s_retry_num < ESP_MAXIMUM_RETRY)
53     {
54         esp_wifi_connect();
55         s_retry_num++;
56         ESP_LOGI(TAG, "retry to connect to the AP");
57     }
58     else
59     {
60         xEventGroupSetBits(s_wifi_event_group, WIFI_FAIL_BIT);
61     }
62     ESP_LOGI(TAG, "connect to the AP fail");
63 }
64 else if (event_base == IP_EVENT && event_id == IP_EVENT_STA_GOT_IP)
65 {
66     ip_event_got_ip_t *event = (ip_event_got_ip_t *)event_data;
67     ESP_LOGI(TAG, "got ip:" IPSTR, IP2STR(&event->ip_info.ip));
68     s_retry_num = 0;
69     xEventGroupSetBits(s_wifi_event_group, WIFI_CONNECTED_BIT);
70 }
71 }
72
73 void wif_init_sta(void)
74 {
75     s_wifi_event_group = xEventGroupCreate();
76     ESP_ERROR_CHECK(esp_netif_init());
77     ESP_ERROR_CHECK(esp_event_loop_create_default());
78     esp_netif_create_default_wifi_sta();
79     wifi_init_config_t cfg = WIFI_INIT_CONFIG_DEFAULT();
80     ESP_ERROR_CHECK(esp_wifi_init(&cfg));
81     esp_event_handler_instance_t instance_any_id;
82     esp_event_handler_instance_t instance_got_ip;
83     ESP_ERROR_CHECK(esp_event_handler_instance_register(WIFI_EVENT,
84                                                         ESP_EVENT_ANY_ID,
85                                                         &event_handler,
86                                                         NULL,
87                                                         &instance_any_id));
88     ESP_ERROR_CHECK(esp_event_handler_instance_register(IP_EVENT,
89                                                         IP_EVENT_STA_GOT_IP,
90                                                         &event_handler,
91                                                         NULL,
92                                                         &instance_got_ip));
93     wifi_config_t wifi_config = {
94         .sta = {
95             .ssid = TARGET_ESP_WIFI_SSID,
```

```
main > C wif_station.c > wif_init_sta(void)
94     .sta = {
95         .ssid = TARGET_ESP_WIFI_SSID,
96         .password = TARGET_ESP_WIFI_PASSWORD,
97         .threshold.authmode = ESP_WIFI_SCAN_AUTH_MODE_THRESHOLD,
98     },
99 };
100 ESP_ERROR_CHECK(esp_wifi_set_mode(WIFI_MODE_STA));
101 ESP_ERROR_CHECK(esp_wifi_set_config(WIFI_IF_STA, &wifi_config));
102 ESP_ERROR_CHECK(esp_wifi_start());
103 ESP_LOGI(TAG, "wifi_init_sta finished.");
104 EventBits_t bits = xEventGroupWaitBits(s_wifi_event_group,
105                                         WIFI_CONNECTED_BIT | WIFI_FAIL_BIT,
106                                         pdFALSE,
107                                         pdFALSE,
108                                         portMAX_DELAY);
109 if (bits & WIFI_CONNECTED_BIT)
110 {
111     ESP_LOGI(TAG, "Connected to ap SSID: %s",
112              TARGET_ESP_WIFI_SSID);
113 }
114 else if (bits & WIFI_FAIL_BIT)
115 {
116     ESP_LOGI(TAG, "Failed to connect to SSID: %s",
117              TARGET_ESP_WIFI_SSID);
118 }
119 else
120 {
121     ESP_LOGE(TAG, "UNEXPECTED EVENT");
122 }
123 }
124
125 void app_main(void)
126 {
127     esp_err_t ret = nvs_flash_init();
128     if (ret == ESP_ERR_NVS_NO_FREE_PAGES || ret == ESP_ERR_NVS_NEW_VERSION_FOUND)
129     {
130         ESP_ERROR_CHECK(nvs_flash_erase());
131         ret = nvs_flash_init();
132     }
133     ESP_ERROR_CHECK(ret);
134     ESP_LOGI(TAG, "ESP_WIFI_MODE_STA");
135     wif_init_sta();
136 }
137
```

Results:

The screenshot shows a development environment with a terminal window displaying the output of an ESP-IDF 4.4 command. The logs show the initialization of the WiFi subsystem, including setting up buffers, enabling the WiFi IRAM OP, and connecting to an access point. The code editor shows the implementation of the WiFi initialization function, which sets up the WiFi configuration, initializes the station, and connects to the access point.

```
ESP-IDF 4.4 CMD - "C:\Espressif\idf_cmd_init.bat" esp-idf-e91d384503485bb54f6ce3d11e841fe
I (605) wifi:Init dynamic tx buffer num: 32
I (615) wifi:Init static rx buffer size: 1600
I (615) wifi:Init static rx buffer num: 10
I (615) wifi:Init dynamic rx buffer num: 32
I (625) wifi_init: rx ba win: 6
I (625) wifi_init: tcpip mbox: 32
I (625) wifi_init: udp mbox: 6
I (635) wifi_init: tcp mbox: 6
I (635) wifi_init: tcp rx win: 5744
I (645) wifi_init: tcp rx win: 5744
I (645) wifi_init: tcp mss: 1440
I (645) wifi_init: WiFi IRAM OP enabled
I (655) wifi_init: WiFi RX IRAM OP enabled
I (665) phy_init: phy version 4670,710199f, Feb 18 2021,17:07:07
I (765) wifi:mode = sta (78:21:84:c6:1c:68)
I (765) wifi:enable tsf
I (765) wifi station: wifi init sta finished.
I (775) wifi:new<10,2>, old:<1,0>, ap:<255,255>, sta:<10,2>, prof:1
I (775) wifi:state: init -> auth (b0)
I (785) wifi:state: auth -> assoc (0)
I (785) wifi:state: assoc -> run (10)
I (825) wifi:connected with TuDuanV, aid = 2, channel 10, 400, bssid = c0:4a:00:d5:7b:34
I (825) wifi:security: WPA2-PSK, phy: bgn, rssi: -37
I (825) wifi:pm start, type: 1
I (915) wifi:AP's beacon interval = 102400 us, DTIM period = 1
I (1555) esp_netif_handlers: sta ip: 192.168.0.101, mask: 255.255.255.0, gw: 192.168.0.1
I (1555) wifi station: got ip:192.168.0.101
I (1555) wifi station: Connected to ap SSID: TuDuanV
```

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

```
126 {
127     esp_err_t ret = nvs_flash_init();
128     if (ret == ESP_ERR_NVS_NO_FREE_PAGES || ret == ESP_ERR_NVS_NEW_VERSION_FOUND)
129     {
130         ESP_ERROR_CHECK(nvs_flash_erase());
131         ret = nvs_flash_init();
132     }
133     ESP_ERROR_CHECK(ret);
134     ESP_LOGI(TAG, "ESP_WIFI_MODE_STA");
135     wifi_init_sta();
136 }
137
```

b) As an Access Point:

Source code:

```
main > C:\wifilab>...
1 #include <string.h>
2 #include "freertos/FreeRTOS.h"
3 #include "freertos/task.h"
4 #include "esp_system.h"
5 #include "esp_wifi.h"
6 #include "esp_event.h"
7 #include "esp_log.h"
8 #include "nvs_flash.h"
9 #include "lwip/err.h"
10 #include "lwip/sys.h"
11
12 #define EXAMPLE_ESP_WIFI_SSID "viettut"
13 #define EXAMPLE_ESP_WIFI_PASS "222222222"
14 #define EXAMPLE_ESP_WIFI_CHANNEL CONFIG_ESP_WIFI_CHANNEL
15 #define EXAMPLE_MAX_STA_CONN CONFIG_ESP_MAX_STA_CONN
16
17 static const char *TAG = "wifi AP";
18
19 static void wifi_event_handler(void *arg, esp_event_base_t event_base,
20                               int32_t event_id, void *event_data)
21 {
22     if (event_id == WIFI_EVENT_AP_STACONNECTED)
23     {
24         wifi_event_ap_staconnected_t *event = (wifi_event_ap_staconnected_t *)event_data;
25         ESP_LOGI(TAG, "station \"MACSTR\" join, AID=%d",
26                 MAC2STR(event->mac), event->aid);
27     }
28     else if (event_id == WIFI_EVENT_AP_STADISCONNECTED)
29     {
30         wifi_event_ap_stadisconnected_t *event = (wifi_event_ap_stadisconnected_t *)
31         event_data;
32         ESP_LOGI(TAG, "station \"MACSTR\" leave, AID=%d",
33                 MAC2STR(event->mac), event->aid);
34     }
35 }
36
37 void wifi_init_softap(void)
38 {
39     ESP_ERROR_CHECK(esp_netif_init());
40     ESP_ERROR_CHECK(esp_event_loop_create_default());
41     esp_netif_create_default_wifi_ap();
42     wifi_init_config_t cfg = WIFI_INIT_CONFIG_DEFAULT();
43     ESP_ERROR_CHECK(esp_wifi_init(&cfg));
44     ESP_ERROR_CHECK(esp_event_handler_instance_register(WIFI_EVENT,
45                                                         ESP_EVENT_ANY_ID,
46                                                         &wifi_event_handler,
47                                                         NULL,
48                                                         NULL));
49     wifi_config_t wifi_config = {
50         .ap = {
51             .ssid = EXAMPLE_ESP_WIFI_SSID,
52             .ssid_len = strlen(EXAMPLE_ESP_WIFI_SSID),
53             .channel = EXAMPLE_ESP_WIFI_CHANNEL,
54             .password = EXAMPLE_ESP_WIFI_PASS,
55             .max_connection = EXAMPLE_MAX_STA_CONN,
56             .authmode = WIFI_AUTH_WPA_WPA2_PSK,
57         }
58     };
59     if (strlen(EXAMPLE_ESP_WIFI_PASS) == 0)
60     {
61         wifi_config.ap.authmode = WIFI_AUTH_OPEN;
62     }
63     ESP_ERROR_CHECK(esp_wifi_set_mode(WIFI_MODE_AP));
64     ESP_ERROR_CHECK(esp_wifi_set_config(WIFI_IF_AP, &wifi_config));
65     ESP_ERROR_CHECK(esp_wifi_start());
66     ESP_LOGI(TAG, "wifi_init softap finished. SSID:%s password:%s channel:%d",
67             EXAMPLE_ESP_WIFI_SSID, EXAMPLE_ESP_WIFI_PASS, EXAMPLE_ESP_WIFI_CHANNEL);
68 }
69
```

```
main > C:\wifilab>...
40 esp_netif_create_default_wifi_ap();
41 wifi_init_config_t cfg = WIFI_INIT_CONFIG_DEFAULT();
42 ESP_ERROR_CHECK(esp_wifi_init(&cfg));
43 ESP_ERROR_CHECK(esp_event_handler_instance_register(WIFI_EVENT,
44                                                         ESP_EVENT_ANY_ID,
45                                                         &wifi_event_handler,
46                                                         NULL,
47                                                         NULL));
48
49 wifi_config_t wifi_config = {
50     .ap = {
51         .ssid = EXAMPLE_ESP_WIFI_SSID,
52         .ssid_len = strlen(EXAMPLE_ESP_WIFI_SSID),
53         .channel = EXAMPLE_ESP_WIFI_CHANNEL,
54         .password = EXAMPLE_ESP_WIFI_PASS,
55         .max_connection = EXAMPLE_MAX_STA_CONN,
56         .authmode = WIFI_AUTH_WPA_WPA2_PSK,
57     }
58 };
59 if (strlen(EXAMPLE_ESP_WIFI_PASS) == 0)
60 {
61     wifi_config.ap.authmode = WIFI_AUTH_OPEN;
62 }
63 ESP_ERROR_CHECK(esp_wifi_set_mode(WIFI_MODE_AP));
64 ESP_ERROR_CHECK(esp_wifi_set_config(WIFI_IF_AP, &wifi_config));
65 ESP_ERROR_CHECK(esp_wifi_start());
66 ESP_LOGI(TAG, "wifi_init softap finished. SSID:%s password:%s channel:%d",
67         EXAMPLE_ESP_WIFI_SSID, EXAMPLE_ESP_WIFI_PASS, EXAMPLE_ESP_WIFI_CHANNEL);
68 }
69
70 void app_main(void)
71 {
72     esp_err_t ret = nvs_flash_init();
73     if (ret == ESP_ERR_NVS_NO_FREE_PAGES || ret == ESP_ERR_NVS_NEW_VERSION_FOUND)
74     {
75         ESP_ERROR_CHECK(nvs_flash_erase());
76         ret = nvs_flash_init();
77     }
78     ESP_ERROR_CHECK(ret);
79     ESP_LOGI(TAG, "ESP_WIFI_MODE_AP");
80     wifi_init_softap();
81 }
```

Results:

The screenshot displays a VS Code editor with two main windows. The left window shows the output of the ESP-IDF 4.4 command, and the right window shows the C code for configuring the ESP32 WiFi subsystem.

ESP-IDF 4.4 CMD: "C:\Espressif\idf_cmd_init.bat" esp-idf-e91d384503485fbb54f6ce3d11e841fe

Output Log:

```
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 (627) wifi_init: tcp mbox: 6
I (627) wifi_init: tcp tx wins: 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 numbers: 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:zzzzzzzzzz channel:1
I (14477) wifi:new:cl,0, old:cl,1, ap:cl,1, sta:c255,255, 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 (15827) esp_wifi:ba: WPS 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 (22887) wifi:ba-add>idx:4 (ifx:1, 32:f5:c0:c1:31:4c), tid:5, ssn:0, winSize:64
```

C Code:

```
ap(void)
{
    esp_netif_init();
    esp_event_loop_create_default();
    default_wifi_ap();
    t.cfg = WIFI_INIT_CONFIG_DEFAULT();
    esp_wifi_init(&t.cfg);
    esp_event_handler_instance_register(WIFI_EVENT,
        ESP_EVENT_ANY_ID,
        &wifi_event_handler,
        NULL,
        NULL);

    t.cfi_config = {

        EXAMPLE_ESP_WIFI_SSID,
        strlen(EXAMPLE_ESP_WIFI_SSID),
        EXAMPLE_ESP_WIFI_CHANNEL,
        EXAMPLE_ESP_WIFI_PASS,
        EXAMPLE_MAX_STA_CONN,
        WIFI_AUTH_WPA2_PSK,
        !EXAMPLE_ESP_WIFI_PASS == 0);

    ap.authmode = WIFI_AUTH_OPEN;

    esp_wifi_set_mode(WIFI_MODE_AP);
    esp_wifi_set_config(WIFI_IF_AP, &t.cfi_config);
    esp_wifi_start();
    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();
}
```