



# AI FARM Robotics Factory



## Lab 01: Understanding WiFi\_Station

July 25 2024

---

NAME:	ID:
In Tara	IN-0008

---

### I. Introduction

The ESP32 is a popular microcontroller with integrated Wi-Fi and Bluetooth capabilities, making it suitable for IoT applications. ESP-IDF (Espressif IoT Development Framework) is the official development framework for the ESP32 chip. In the context of Wi-Fi, the ESP32 can operate in different modes, including Station (STA), Soft Access Point (AP), and both simultaneously.

### II. Understanding WIFI Station Mode

In Wi-Fi Station mode, the ESP32 connects to an existing Wi-Fi network as a client. This mode is used when you want your ESP32 to connect to a router or access point, enabling it to communicate with other devices on the network or access the internet.

### III. Key Components of ESP-IDF Wi-Fi Station Code

- Make a new project of Espressif and then select type of esp board ( esp32, esp32s3, esp32c3 , esp32c6.....)
- **Initialization and Configuration:** This involves initializing the TCP/IP stack and configuring the Wi-Fi settings.
- **Event Handling:** ESP-IDF uses an event-driven model where various Wi-Fi events (like connection, disconnection, IP acquisition) are handled through event handlers.
- **Connecting to the Wi-Fi Network:** This involves specifying the SSID and password of the target network and initiating the connection process.

My code:

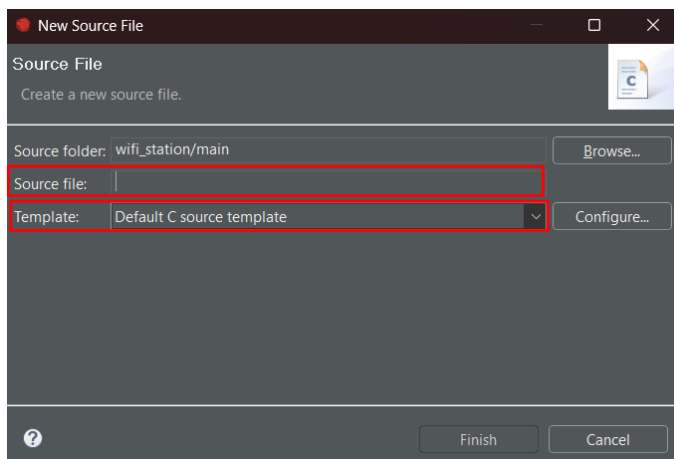
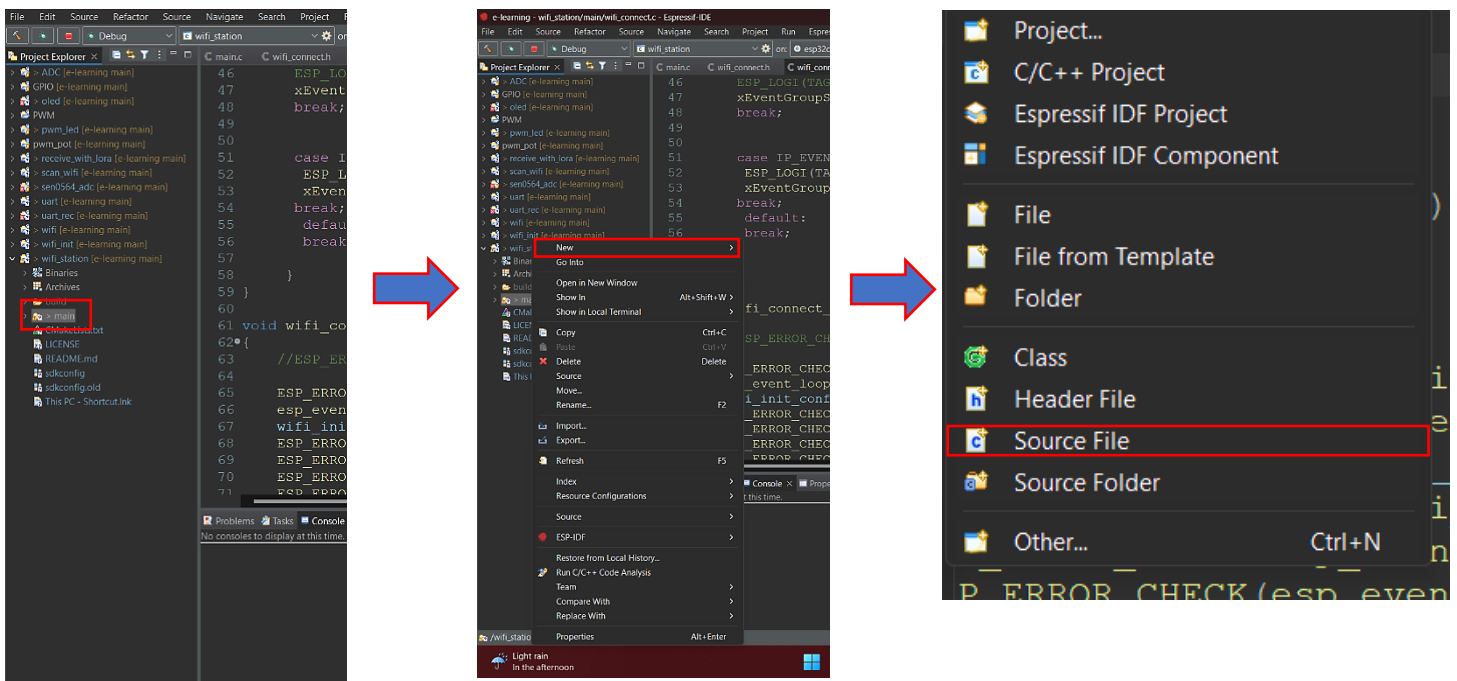
**Includes and Setup:** Necessary headers for Wi-Fi, FreeRTOS, logging, and NVS (Non-Volatile Storage) are included in file main.c .

```
#include <stdio.h>
#include "portmacro.h"
#include "nvs_flash.h"
#include "freertos/FreeRTOS.h"
#include "freertos/task.h"
```

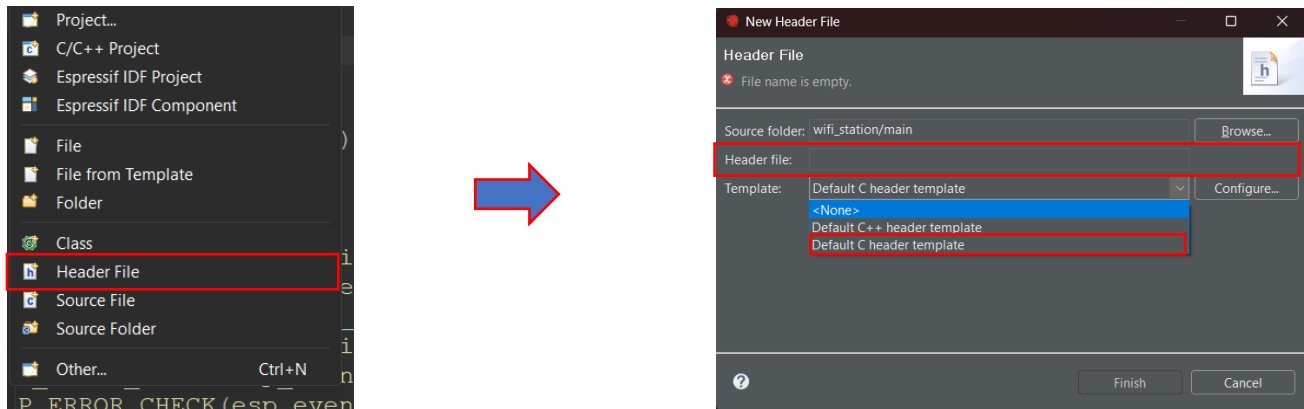
After include some library in main.c , we have to create new library your own.

For create new library we need to do like :

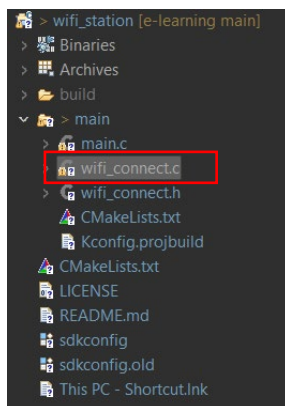
- Go to file that have name “main” -> right click on “ main ” -> click on “new” -> click on source file and then give name for this file , for this file name need to end by “.c” (example: wifi\_connect.c) -> at the last we select “ Default C Source template “ in Template and click finish .



- Do it again and we make header file after right click on “new “ -> click on header file  
-> give name for this (for recommendation should give the same name of source file)  
for header name need to end by “.h” (example: wifi\_connect.h ).



- Create new file (source file , and header file ) and give wifi\_connect.c for source file and wifi\_connect.h for header file . create source file for write code for make own library and header file for include to in file main.c .



we write code for library in file wifi\_connect.c:

### 1. Step1:

Include necessary library

```
#include <stdio.h>
#include "esp_err.h"
#include "esp_event.h"
#include "esp_log.h"
#include "esp_netif_types.h"
#include "esp_wifi.h"
#include "esp_netif.h"
#include "esp_wifi_default.h"
#include "esp_wifi_types.h"
#include "riscv/encoding.h"
```

```
#include "string.h"
#include "freertos/FreeRTOS.h"
#include "freertos/task.h"
#include "freertos/event_groups.h"
```

## 2. Step2:

Define some parameter

```
static char* TAG = "WIFI CONNECT" ;
static esp_netif_t *esp_netif ;
static EventGroupHandle_t wifi_events ;
static int CONNECTED = BIT0 ;
static int DISCONNECTED = BIT1 ;
```

## 3. Step3

The `wifi_event_handler` function handles various Wi-Fi events such as connection start, disconnection, and IP address acquisition.

```
void event_handler(void *event_handler_arg, esp_event_base_t
event_base, int32_t event_id, void *event_data)
{
    switch (event_id) {
        case WIFI_EVENT_STA_START:
            ESP_LOGI(TAG, "WIFI_EVENT_STA_START");
            esp_wifi_connect();
            break;
        case WIFI_EVENT_STA_CONNECTED:
            ESP_LOGI(TAG, "WIFI_EVENT_STA_CONNECTED");
            break;
        case WIFI_EVENT_STA_DISCONNECTED:
            /*{
                wifi_event_sta_disconnected_t *wifi_event_sta_disconnected
= event_data ;
                ESP_LOGI(TAG, "DISCONNECTED %d"wifi_event_sta_disconnected->
reason);*/
                xEventGroupSetBits(wifi_events, DISCONNECTED);
                break;

        case IP_EVENT_STA_GOT_IP:
            ESP_LOGI(TAG, "IP_EVENT_STA_GOT_IP");
            xEventGroupSetBits(wifi_events, CONNECTED);
            break;
        default:
            break;
    }
}
```

#### 4. Step4

The `wifi_init_sta` function initializes the TCP/IP stack, creates a default Wi-Fi station interface, and sets up the Wi-Fi configuration with SSID and password.

```
void wifi_connect_init(void)
{
    //ESP_ERROR_CHECK(x)

    ESP_ERROR_CHECK(esp_netif_init());
    esp_event_loop_create_default();
    wifi_init_config_t wifi_init_config = WIFI_INIT_CONFIG_DEFAULT();
    ESP_ERROR_CHECK(esp_wifi_init(&wifi_init_config));
    ESP_ERROR_CHECK(esp_event_handler_register(WIFI_EVENT, ESP_EVENT_ANY_ID, event_handler, NULL));
    ESP_ERROR_CHECK(esp_event_handler_register(IP_EVENT, IP_EVENT_STA_GOT_IP, event_handler, NULL));
    ESP_ERROR_CHECK(esp_wifi_set_storage(WIFI_STORAGE_RAM));
}
```

#### 5. Step5:

- This function **wifi\_connect\_sta** takes three parameters: **ssid** (the Wi-Fi SSID), **pass** (the Wi-Fi password), and **timeout** (the timeout duration for connecting).
- **wifi\_events** is created as an event group using **xEventGroupCreate()**. Event groups are used for task synchronization and signaling.
- **esp\_netif\_create\_default\_wifi\_sta()** start with the default Wi-Fi station (STA) interface.
- **esp\_wifi\_set\_mode(WIFI\_MODE\_STA)** sets the Wi-Fi mode to station mode. This allows the device to connect to an access point.
- A **wifi\_config\_t** structure named **wifi\_config** is created and start off. The SSID and password are copied into the **wifi\_config** structure using **strncpy()**. The -1 in **sizeof(wifi\_config.sta.ssid)-1** ensures there is space for the null terminator.
- **esp\_wifi\_set\_config()** applies the Wi-Fi configuration to the STA interface using the **wifi\_config** structure.
- **esp\_wifi\_start()** starts the Wi-Fi driver.
- **xEventGroupWaitBits()** waits for either the CONNECTED or DISCONNECTED event bits to be set in the **wifi\_events** event group.
- **true**: Clears the bits before returning.
- **false**: Do not wait for all bits to be set.
- **pdMS\_TO\_TICKS(timeout)** converts the timeout value from milliseconds to FreeRTOS ticks.
- 
- Condition **if**:

If the `CONNECTED` bit is set, it prints "connect...yey".

If the DISCONNECTED bit is set in type of loop , it prints ".....can not connect.....".

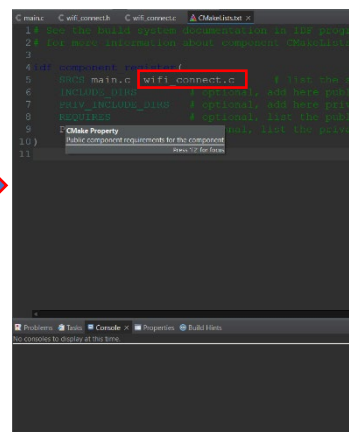
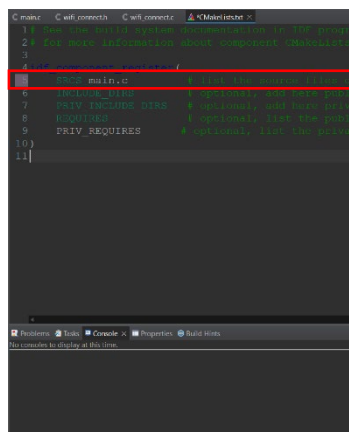
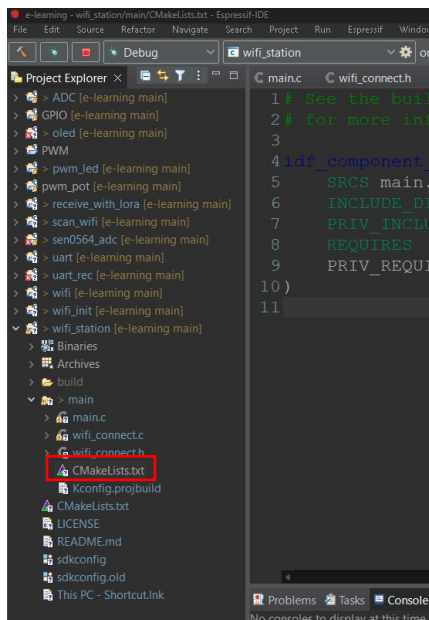
```
esp_err_t wifi_connect_sta(char* ssid, char* pass , int timeout){
    wifi_events = xEventGroupCreate();
    esp_netif = esp_netif_create_default_wifi_sta();
    esp_wifi_set_mode(WIFI_MODE_STA);
    wifi_config_t wifi_config = {};
    strncpy((char *)wifi_config.sta.ssid, ssid,
sizeof(wifi_config.sta.ssid)-1);
    strncpy((char *)wifi_config.sta.password, pass,
sizeof(wifi_config.sta.password)-1);
    esp_wifi_set_config(WIFI_IF_STA, &wifi_config);
    esp_wifi_start();

    EventBits_t result = xEventGroupWaitBits(wifi_events, CONNECTED
| DISCONNECTED ,
true, false, pdMS_TO_TICKS(timeout));

    if(result == CONNECTED) {
        printf("connect...yey  \n");
    }for(;;)
    if (result == DISCONNECTED) {
        printf(".....can not connect.....\n");
    }
    vTaskDelay(100/portTICK_PERIOD_MS);
}
```

- ii. After prepared code in source, we have to prepare code in header file, but before write code in header file we have to edit something in CMakeList.Txt . So we need to do like this:

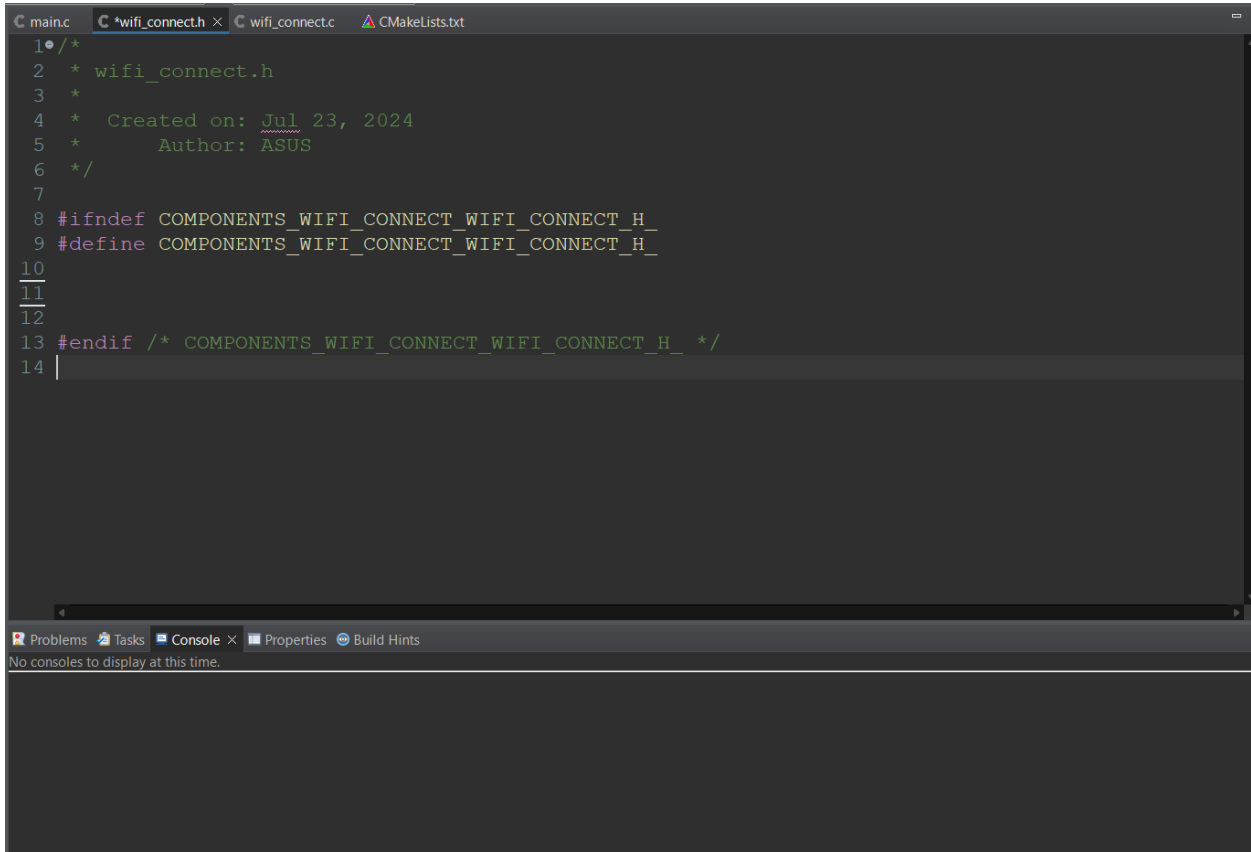
➤ Step to setup in CMakeList.txt



iii. Add wifi\_connect.c in CMakeList.txt and then we write code in header file .

1) Step1

Go to header file (wifi\_connect.h) and we will see like this .

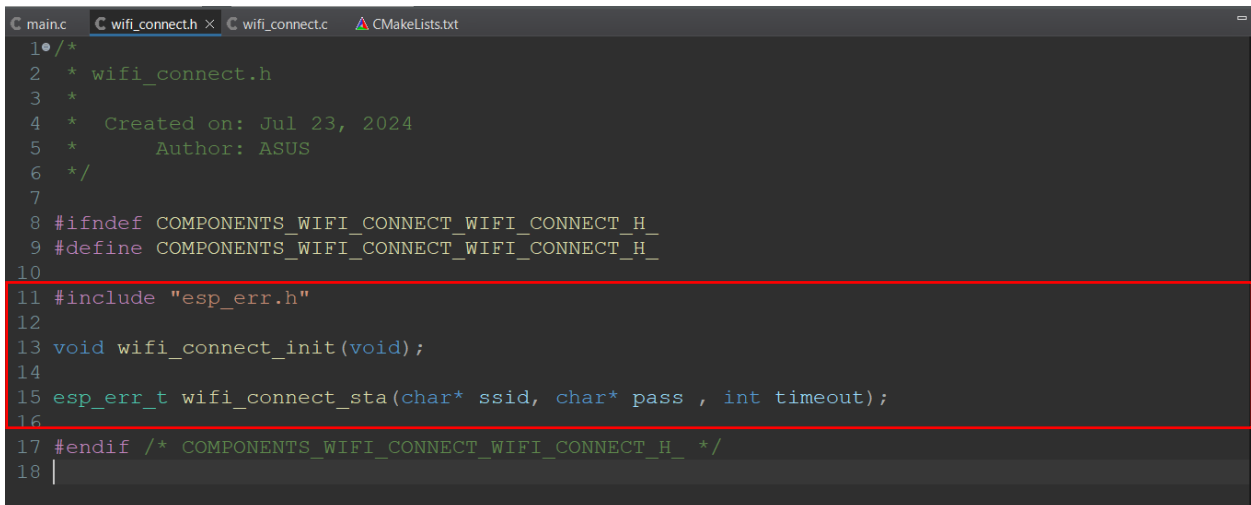
A screenshot of an IDE window showing the 'wifi\_connect.h' header file. The file contains a multi-line comment with creation date and author, followed by a guard macro. The code is as follows:

```
1 /*
2  * wifi_connect.h
3  *
4  * Created on: Jul 23, 2024
5  * Author: ASUS
6  */
7
8 #ifndef COMPONENTS_WIFI_CONNECT_WIFI_CONNECT_H_
9 #define COMPONENTS_WIFI_CONNECT_WIFI_CONNECT_H_
10
11
12
13 #endif /* COMPONENTS_WIFI_CONNECT_WIFI_CONNECT_H_ */
14 |
```

The IDE interface includes tabs for 'main.c', 'wifi\_connect.h', 'wifi\_connect.c', and 'CMakeLists.txt'. At the bottom, there is a status bar with 'Problems', 'Tasks', 'Console', 'Properties', and 'Build Hints' sections. The 'Console' section shows the message 'No consoles to display at this time.'

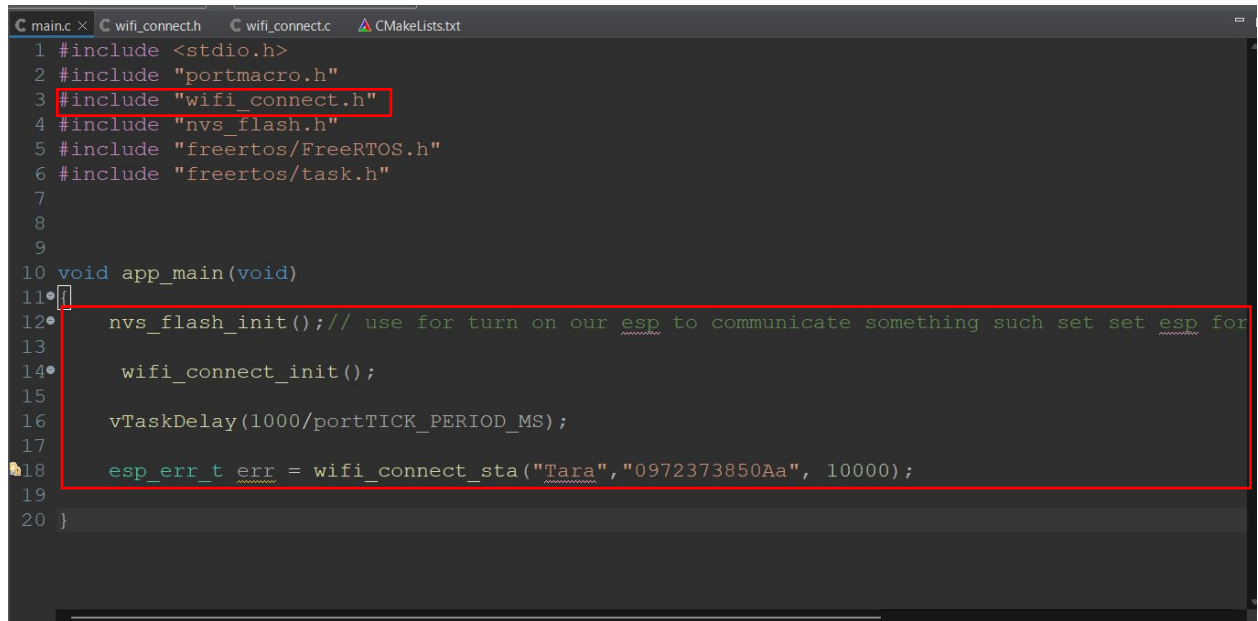
2) Step2

Include some library and call function that create in source file. At last click CTRL+S for save.

A screenshot of the same IDE window, now showing the 'wifi\_connect.h' file with additional function declarations. Lines 11 through 15 are highlighted with a red box. The code is as follows:

```
1 /*
2  * wifi_connect.h
3  *
4  * Created on: Jul 23, 2024
5  * Author: ASUS
6  */
7
8 #ifndef COMPONENTS_WIFI_CONNECT_WIFI_CONNECT_H_
9 #define COMPONENTS_WIFI_CONNECT_WIFI_CONNECT_H_
10
11 #include "esp_err.h"
12
13 void wifi_connect_init(void);
14
15 esp_err_t wifi_connect_sta(char* ssid, char* pass , int timeout);
16
17 #endif /* COMPONENTS_WIFI_CONNECT_WIFI_CONNECT_H_ */
18 |
```

After we prepare library your own already we go to file main.c to write code for esp32 let it work. We have to add code in void app\_main(void):



```
C main.c x C wifi_connect.h C wifi_connect.c CMakeLists.txt
1 #include <stdio.h>
2 #include "portmacro.h"
3 #include "wifi_connect.h"
4 #include "nvs_flash.h"
5 #include "freertos/FreeRTOS.h"
6 #include "freertos/task.h"
7
8
9
10 void app_main(void)
11 {
12     nvs_flash_init(); // use for turn on our esp to communicate something such set set esp for
13
14     wifi_connect_init();
15
16     vTaskDelay(1000/portTICK_PERIOD_MS);
17
18     esp_err_t err = wifi_connect_sta("Tara", "0972373850Aa", 10000);
19
20 }
```

Include library that we made into file main.c .

For code in link 18 `esp_err_t err = wifi_connect_sta("Tara", "0972373850Aa", 10000);` it is function that we created in wifi\_connect file:

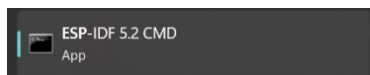
`esp_err_t wifi_connect_sta(char* ssid, char* pass , int timeout)`

- place of ssid: it write name wifi that we want to connect .
- place of pass: it write the password of wifi.
- Place of timeout : we choose 10000 .

#### iv. Running code

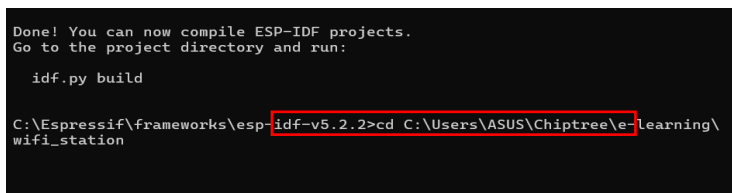
- Step1

Go find ESP-IDF CMD and open it .



- Step2

Use comment cd for change directory . write cd and add 1 space and past the directory of our project .



```
Done! You can now compile ESP-IDF projects.
Go to the project directory and run:

idf.py build

C:\Espressif\frameworks\esp-idf-v5.2.2>cd C:\Users\ASUS\Chiptree\learning\
wifi_station
```



- Step3

Use comment idf.py set-target esp32 (type of your board) to set type of esp32

```
Done! You can now compile ESP-IDF projects.
Go to the project directory and run:

idf.py build

C:\Espressif\frameworks\esp-idf-v5.2.2>cd C:\Users\ASUS\Chiptree\e-learning\
wifi_station
C:\Users\ASUS\Chiptree\e-learning\wifi_station>idf.py set-target esp32c3
```

- Step4

Use comment "idf.py build" for build your code in your project

```
/wpa_supplicant
-- Configuring done
-- Generating done
-- Build files have been written to: C:/Users/ASUS/Chiptree/e-learning/wifi_
station/build
C:\Users\ASUS\Chiptree\e-learning\wifi_station>idf.py build |
```

- Step5

Use comment "idf.py -p COM4 flash monitor for enter the code into esp and we will see IP address in monitor .

```
Generated C:/Users/ASUS/Chiptree/e-learning/wifi_station/build/app-template.
bin
[894/894] cmd.exe /C "cd /D C:\Users\ASUS\Chiptree\e-learning\wifi_station\build\app-template.bin"
app-template.bin binary size 0xc0a80 bytes. Smallest app partition is 0x1000
00 bytes. 0x3f580 bytes (25%) free.

Project build complete. To flash, run:
idf.py flash
or
idf.py -p PORT flash
or
python -m esptool --chip esp32c3 -b 460800 --before default_reset --after h
ard_reset write_flash --flash_mode dio --flash_size 2MB --flash_freq 80m 0x0
build\bootloader\bootloader.bin 0x8000 build\partition_table\partition-tabl
e.bin 0x10000 build\app-template.bin
or from the "C:\Users\ASUS\Chiptree\e-learning\wifi_station\build" directory
python -m esptool --chip esp32c3 -b 460800 --before default_reset --after h
ard_reset write_flash "@flash_args"
C:\Users\ASUS\Chiptree\e-learning\wifi_station>idf.py -p COM4 flash monitor
```

## IV. Conclusion

Implementing a Wi-Fi station on the ESP32 using ESP-IDF involves several steps, including initializing NVS, configuring the Wi-Fi settings, handling Wi-Fi events, and starting the Wi-Fi driver. By following these steps, you can connect your ESP32 to a Wi-Fi network and handle various network events efficiently.