Entregable 1 - wifi/getting_started/station

En la tarea se compila el modo station en los siguientes escenarios:

Conectar ESP32 con un punto de acceso existente

Para ello se actualizan los parámetros existente "WiFi SSID" y "WiFi Password" en el apartado de SDK Configuration Editor.

Tras compilar y flashear se observa la siguiente salida:

```
I (600) wifi station: ESP_WIFI_MODE_STA
I (610) wifi:wifi driver task: 3ffbf94c, prio:23, stack:6656, core=0
I (630) wifi:wifi firmware version: ce9244d
I (640) wifi:wifi certification version: v7.0
I (640) wifi:config NVS flash: enabled
I (640) wifi:config nano formating: disabled
I (640) wifi:Init data frame dynamic rx buffer num: 32
I (640) wifi:Init management frame dynamic rx buffer num: 32
I (650) wifi:Init management short buffer num: 32
I (650) wifi:Init dynamic tx buffer num: 32
I (660) wifi:Init static rx buffer size: 1600
I (660) wifi:Init static rx buffer num: 10
I (670) wifi:Init dynamic rx buffer num: 32
I (670) wifi_init: rx ba win: 6
I (670) wifi_init: tcpip mbox: 32
I (680) wifi_init: udp mbox: 6
I (680) wifi_init: tcp mbox: 6
I (680) wifi_init: tcp tx win: 5744
I (690) wifi_init: tcp rx win: 5744
I (690) wifi_init: tcp mss: 1440
I (700) wifi_init: WiFi IRAM OP enabled
I (700) wifi_init: WiFi RX IRAM OP enabled
I (710) phy_init: phy_version 4670,719f9f6,Feb 18 2021,17:07:07
I (820) wifi:mode : sta (8c:aa:b5:b8:bf:f4)
I (820) wifi:enable tsf
I (820) wifi station: wifi_init_sta finished.
I (950) wifi:new:<1,0>, old:<1,0>, ap:<255,255>, sta:<1,0>, prof:1
I (2110) wifi:state: init -> auth (b0)
I (2110) wifi:state: auth -> assoc (0)
I (2120) wifi:state: assoc -> run (10)
I (2150) wifi:connected with WIFIAP, aid = 1, channel 1, BW20, bssid =
b2:08:43:4c:40:bf
I (2150) wifi:security: WPA2-PSK, phy: bgn, rssi: -28
I (2170) wifi:pm start, type: 1

I (2170) wifi:<ba-add>idx:0 (ifx:0, b2:08:43:4c:40:bf), tid:0, ssn:0,
winSize:64
I (2240) wifi:AP's beacon interval = 102400 us, DTIM period = 2
I (3180) esp_netif_handlers: sta ip: 192.168.223.86, mask: 255.255.255.0,
```

```
gw: 192.168.223.152
I (3180) wifi station: got ip:192.168.223.86
I (3180) wifi station: connected to ap SSID:WIFIAP password:Wifiap123?
I (3190) main_task: Returned from app_main()
```

Conectar ESP32 con un punto de acceso inexistente

En este caso se modifica el parámetro "WiFi SSID" con un valor aleatorio observando la siguiente salida:

```
I (599) wifi station: ESP_WIFI_MODE_STA
I (609) wifi:wifi driver task: 3ffbf94c, prio:23, stack:6656, core=0
I (629) wifi:wifi firmware version: ce9244d
I (639) wifi:wifi certification version: v7.0
I (639) wifi:config NVS flash: enabled
I (639) wifi:config nano formating: disabled
I (639) wifi:Init data frame dynamic rx buffer num: 32
I (639) wifi:Init management frame dynamic rx buffer num: 32
I (649) wifi:Init management short buffer num: 32
I (649) wifi:Init dynamic tx buffer num: 32
I (659) wifi:Init static rx buffer size: 1600
I (659) wifi:Init static rx buffer num: 10
I (669) wifi:Init dynamic rx buffer num: 32
I (669) wifi_init: rx ba win: 6
I (669) wifi_init: tcpip mbox: 32
I (679) wifi_init: udp mbox: 6
I (679) wifi_init: tcp mbox: 6
I (679) wifi_init: tcp tx win: 5744
I (689) wifi_init: tcp rx win: 5744
I (689) wifi_init: tcp mss: 1440
I (699) wifi_init: WiFi IRAM OP enabled
I (699) wifi_init: WiFi RX IRAM OP enabled
I (709) phy_init: phy_version 4670,719f9f6,Feb 18 2021,17:07:07
I (819) wifi:mode : sta (8c:aa:b5:b8:bf:f4)
I (819) wifi:enable tsf
I (819) wifi station: wifi_init_sta finished.
I (3229) wifi station: retry to connect to the AP
I (3239) wifi station: connect to the AP fail
I (5649) wifi station: retry to connect to the AP
I (5649) wifi station: connect to the AP fail
I (8059) wifi station: retry to connect to the AP
I (8059) wifi station: connect to the AP fail
I (10469) wifi station: retry to connect to the AP
I (10469) wifi station: connect to the AP fail
I (12879) wifi station: retry to connect to the AP
I (12879) wifi station: connect to the AP fail
I (15289) wifi station: connect to the AP fail
I (15289) wifi station: Failed to connect to SSID:WIFIAPINEXISTENTE,
password:Wifiap123?
I (15299) main_task: Returned from app_main()
```

En este caso, no se puede conectar al SSID WIFIAPINEXISTENTE intentando la reconexión tal cual se establece en el manejador event_handler ante el evento WIFI_EVENT_STA_DISCONNECTED.

Apagar el punto de acceso mientras la ip está concendida

```
I (600) wifi station: ESP_WIFI_MODE_STA
I (610) wifi:wifi driver task: 3ffbf94c, prio:23, stack:6656, core=0
I (630) wifi:wifi firmware version: ce9244d
I (640) wifi:wifi certification version: v7.0
I (640) wifi:config NVS flash: enabled
I (640) wifi:config nano formating: disabled
I (640) wifi:Init data frame dynamic rx buffer num: 32
I (640) wifi:Init management frame dynamic rx buffer num: 32
I (650) wifi:Init management short buffer num: 32
I (650) wifi:Init dynamic tx buffer num: 32
I (660) wifi:Init static rx buffer size: 1600
I (660) wifi:Init static rx buffer num: 10
I (670) wifi:Init dynamic rx buffer num: 32
I (670) wifi_init: rx ba win: 6
I (670) wifi_init: tcpip mbox: 32
I (680) wifi_init: udp mbox: 6
I (680) wifi_init: tcp mbox: 6
I (680) wifi_init: tcp tx win: 5744
I (690) wifi_init: tcp rx win: 5744
I (690) wifi_init: tcp mss: 1440
I (700) wifi_init: WiFi IRAM OP enabled
I (700) wifi_init: WiFi RX IRAM OP enabled
I (710) phy_init: phy_version 4670,719f9f6,Feb 18 2021,17:07:07
I (820) wifi:mode : sta (8c:aa:b5:b8:bf:f4)
I (820) wifi:enable tsf
I (820) wifi station: wifi_init_sta finished.
I (830) wifi:new:<6,0>, old:<1,0>, ap:<255,255>, sta:<6,0>, prof:1
I (830) wifi:state: init -> auth (b0)
I (840) wifi:state: auth -> assoc (0)
I (850) wifi:state: assoc -> run (10)
I (870) wifi:connected with WIFIAP, aid = 1, channel 6, BW20, bssid =
b2:08:43:4c:40:bf
I (870) wifi:security: WPA2-PSK, phy: bgn, rssi: -32
I (870) wifi:pm start, type: 1

I (880) wifi:<ba-add>idx:0 (ifx:0, b2:08:43:4c:40:bf), tid:0, ssn:0,
winSize:64
I (940) wifi:AP's beacon interval = 102400 us, DTIM period = 2
I (1880) esp_netif_handlers: sta ip: 192.168.223.86, mask: 255.255.255.0,
gw: 192.168.223.152
I (1880) wifi station: got ip:192.168.223.86
I (1880) wifi station: connected to ap SSID:WIFIAP password:Wifiap123?
I (1890) main_task: Returned from app_main()
I (35040) wifi:state: run -> init (1c0)
I (35040) wifi:pm stop, total sleep time: 32217443 us / 34166625 us

I (35040) wifi:<ba-del>idx:0, tid:0
```

```
I (35040) wifi:new:<6,0>, old:<6,0>, ap:<255,255>, sta:<6,0>, prof:1
I (35050) wifi station: retry to connect to the AP
I (35050) wifi station: connect to the AP fail
I (35060) wifi:new:<6,0>, old:<6,0>, ap:<255,255>, sta:<6,0>, prof:1
I (35070) wifi:state: init -> auth (b0)
I (35070) wifi:state: auth -> init (3c0)
I (35070) wifi:new:<6,0>, old:<6,0>, ap:<255,255>, sta:<6,0>, prof:1
I (35080) wifi station: retry to connect to the AP
I (35080) wifi station: connect to the AP fail
I (37490) wifi station: retry to connect to the AP
I (37490) wifi station: connect to the AP fail
I (39900) wifi station: retry to connect to the AP
I (39900) wifi station: connect to the AP fail
I (42320) wifi station: retry to connect to the AP
I (42320) wifi station: connect to the AP fail
I (44720) wifi station: connect to the AP fail
```

Revisando el tratamiento de eventos del código del ejemplo observamos que los siguientes eventos se tratan dentro de la función event_handler:

- WIFI_EVENT_STA_START //Se llama a la función esp_wifi_connect();
- WIFI_EVENT_STA_DISCONNECTED //Se reintenta la conexión llamando la función esp_wifi_connect();
- IP_EVENT_STA_GOT_IP //Se muestra la ip obtenida por DHCP

Dentro del apartado "Modo Station" se especifican eventos adicionales dentro de las distintas fases de conexión wifi:

- WIFI_EVENT_STA_CONNECTED // Se muestra el nombre del AP al que se ha conectado correctamente

```
I (600) wifi station: ESP_WIFI_MODE_STA
I (610) wifi:wifi driver task: 3ffbf94c, prio:23, stack:6656, core=0
I (630) wifi:wifi firmware version: ce9244d
I (640) wifi:wifi certification version: v7.0
I (640) wifi:config NVS flash: enabled
I (640) wifi:config nano formating: disabled
I (640) wifi:Init data frame dynamic rx buffer num: 32
I (640) wifi:Init management frame dynamic rx buffer num: 32
I (650) wifi:Init management short buffer num: 32
I (650) wifi:Init dynamic tx buffer num: 32
I (660) wifi:Init static rx buffer size: 1600
I (660) wifi:Init static rx buffer num: 10
I (670) wifi:Init dynamic rx buffer num: 32
I (670) wifi_init: rx ba win: 6
I (670) wifi_init: tcpip mbox: 32
I (680) wifi_init: udp mbox: 6
I (680) wifi_init: tcp mbox: 6
I (690) wifi_init: tcp tx win: 5744
I (690) wifi_init: tcp rx win: 5744
I (690) wifi_init: tcp mss: 1440
```

```
I (700) wifi_init: WiFi IRAM OP enabled
I (700) wifi_init: WiFi RX IRAM OP enabled
I (710) phy_init: phy_version 4670,719f9f6,Feb 18 2021,17:07:07
I (810) wifi:mode : sta (8c:aa:b5:b8:bf:f4)
I (820) wifi:enable tsf
I (820) wifi station: wifi_init_sta finished.
I (830) wifi:new:<1,0>, old:<1,0>, ap:<255,255>, sta:<1,0>, prof:1
I (830) wifi:state: init -> auth (b0)
I (830) wifi:state: auth -> assoc (0)
I (850) wifi:state: assoc -> run (10)
I (870) wifi:state: run -> init (2c0)
I (870) wifi:new:<1,0>, old:<1,0>, ap:<255,255>, sta:<1,0>, prof:1
I (880) wifi station: retry to connect to the AP
I (880) wifi station: connect to the AP fail
I (3290) wifi station: retry to connect to the AP
I (3290) wifi station: connect to the AP fail
I (3300) wifi:new:<1,0>, old:<1,0>, ap:<255,255>, sta:<1,0>, prof:1
I (3300) wifi:state: init -> auth (b0)
I (3300) wifi:state: auth -> assoc (0)
I (3310) wifi:state: assoc -> run (10)
I (3450) wifi:connected with WIFIAP, aid = 1, channel 1, BW20, bssid =
b2:08:43:4c:40:bf
I (3450) wifi:security: WPA2-PSK, phy: bgn, rssi: -34
I (3460) wifi:pm start, type: 1

I (3460) wifi:<ba-add>idx:0 (ifx:0, b2:08:43:4c:40:bf), tid:0, ssn:0,
winSize:64
I (3470) wifi station: Connected successfully to AP: WIFIAP
I (3520) wifi:AP's beacon interval = 102400 us, DTIM period = 2
I (4470) esp_netif_handlers: sta ip: 192.168.223.86, mask: 255.255.255.0,
gw: 192.168.223.152
I (4470) wifi station: got ip:192.168.223.86
I (4470) wifi station: connected to ap SSID:WIFIAP password:Wifiap123?
I (4480) main_task: Returned from app_main()
```

Modo Punto de Acceso

En el presente entregable se trabajará con el uso de la biblioteca WIFI proporcionada para la placa STM32 en su modo para trabajar como Punto de Acceso.

Tarea - Ejemplo de conexión

> **Tarea**
>
> Analiza el ejemplo softAP, compílalo y flashealo. Estudia el tratamiento de eventos que realiza, y cómo estos son emitidos para casos reales. Para ello, conecta distintos clientes (stations), bien sean ESP32 o cualquier otro dispositivo, y analiza los eventos generados y su respuesta.

Para comprobar que la placa funciona correctamente como Punto de Acceso, antes de realizar la conexión con otro dispositivo, ejecutaremos el código entregado y examinaremos su salida.

A continuación podemos ver la salida obtenida y como hemos dividido el proceso de configuración wifi en tres partes:

- **Paso 1 - Creación de los elementos para operar con el WIFI:** En este paso realizamos las siguientes actividades:
    - Creamos la pila TCP/IP
    - Creamos el bucle para la gestión de los eventos
    - Creamos la interfáz de red para asociar el dispositivo a la pila TCP/IP
    - Inicializamos la tarea que gestionará la conexión WIFI
- **Paso 2: Configuramos los parámetros de la conexión:** Especificamos elementos como el SSID de la red, la contraseña utilizada y el modo de operación de la placa (en nuestro caso como AP). Podmeos ver que no obtenemos ninguna salida.
- **Paso 3: Ejecución del driver WIFI:** Podemos ver como este inicia y nos devuelve información acerca del mismo.

Para finalizar podemos ver como se muestra información util de cara al usuario sobre el punto de acceso obtenido, tal como la IP del dispositivo, el SSID de la red o la contraseña de la misma.

```
**CONFIGURACIÓN WIFI - INICIO
*

FASE 1..: CREAMOS E INICIALIZAMOS LOS ELEMENTOS NECESARIOS PARA PODER
OPERAR CON EL WIFI
I (629) wifi:wifi driver task: 3ffbf918, prio:23, stack:6656, core=0
I (659) wifi:wifi firmware version: e03c1ca
I (659) wifi:wifi certification version: v7.0
I (659) wifi:config NVS flash: enabled
I (659) wifi:config nano formating: disabled
I (659) wifi:Init data frame dynamic rx buffer num: 32
I (669) wifi:Init management frame dynamic rx buffer num: 32
I (669) wifi:Init management short buffer num: 32
I (679) wifi:Init dynamic tx buffer num: 32
I (679) wifi:Init static rx buffer size: 1600
I (679) wifi:Init static rx buffer num: 10
I (689) wifi:Init dynamic rx buffer num: 32
I (689) wifi_init: rx ba win: 6
I (689) wifi_init: tcpip mbox: 32
I (699) wifi_init: udp mbox: 6
I (699) wifi_init: tcp mbox: 6
I (709) wifi_init: tcp tx win: 5744
I (709) wifi_init: tcp rx win: 5744
I (709) wifi_init: tcp mss: 1440
I (719) wifi_init: WiFi IRAM OP enabled
I (719) wifi_init: WiFi RX IRAM OP enabled

FASE 2..: CONFIGURAMOS EL TIPO Y LOS PARAMETROS DE LA CONEXIÓN WIFI A
```

```
REALIZAR

FASE 3..: INICIAMOS LA EJECUCIÓN DEL FRIVER WIFI
I (749) phy_init: phy_version 4771,450c73b,Aug 16 2023,11:03:10
I (829) wifi:mode : softAP (24:0a:c4:ea:36:b5)
I (949) wifi:Total power save buffer number: 16
I (949) wifi:Init max length of beacon: 752/752
I (949) wifi:Init max length of beacon: 752/752

*
**CONFIGURACIÓN WIFI - FIN

I (949) esp_netif_lwip: DHCP server started on interface WIFI_AP_DEF with
IP: 192.168.4.1
I (969) wifi softAP: wifi_init_softap finished. SSID:RPI1_test
password:test1234 channel:1
I (969) main_task: Returned from app_main()
```

Una vez que se ha llevado a cabo la inicialización del sistema como Punto de acceso y hemos comprobado la salida, pasaremos a conectar un dispositivo al mismo y observar lo que este nos devuelve. Tener en cuenta que esta salida se realiza dentro de la manejadora que trata el evento **WIFI_EVENT_AP_STACONNECTED**.

En este caso podemos observar como se ha solicitado la conexión desde una nueva estación a nuestro Punto de Acceso con dirección MAC DE:23:9C:51:10:93. Por otra parte vemos como el sistema ha asignado a esta nueva estación la dirección IP 192.168.4.2

```
I (20319) wifi:new:<1,1>, old:<1,1>, ap:<1,1>, sta:<255,255>, prof:1
I (20319) wifi:station: f8:94:c2:b9:6b:08 join, AID=1, bgn, 40U
I (20359) wifi softAP: station f8:94:c2:b9:6b:08 join, AID=1
I (20459) esp_netif_lwip: DHCP server assigned IP to a client, IP is:
192.168.4.2
I (20679) wifi:<ba-add>idx:2 (ifx:1, f8:94:c2:b9:6b:08), tid:0, ssn:15,
winSize:64
```

Para finalizar vamos a llevar a cabo la desconexión de la estación para obtener la siguiente salida, donde vemos que esta ha sido detectada por el Punto de Acceso. Tener en cuenta que esta salida se realiza dentro de la manejadora que trata el evento **WIFI_EVENT_AP_STADISCONNECTED**.

```
I (54499) wifi:station: f8:94:c2:b9:6b:08 leave, AID = 1, bss_flags is
8391747, bss:0x3ffb91f0
I (54499) wifi:new:<1,0>, old:<1,1>, ap:<1,1>, sta:<255,255>, prof:1
I (54509) wifi:<ba-del>idx:2, tid:0
I (54509) wifi softAP: station f8:94:c2:b9:6b:08 leave, AID=1
```

Entregable 2 - Configuración de eventos

> Entregable 2
>
> Revisa el tratamiento de eventos del código anterior, añade el tratamiento de los eventos que falten por tratar. Añade en tu código un comentario explicando el código añadido.

En el ejemplo del cual parte el entregable se realiza el manejo únicamente de dos eventos:

- WIFI_EVENT_AP_STACONNECTED: Lanzado cuando una estación lleva a cabo una conexión con el Punto de Acceso.
- WIFI_EVENT_AP_STADISCONNECTED: Lanzado cuando una estación se desconecta del Punto de acceso.

Ya hemos visto la salida que producen ambos eventos en el apartado anterior, donde hemos llevado a cabo la conexión y desconexión de una estación, por lo que no volveremos a visualizarla ahora.

Sin embargo, examinando todos los posibles eventos lanzados por el sistema WIFI configurado como Punto de Acceso y filtrando aquellos que sean relevantes, observamos que WIFI_EVENT_AP_START y WIFI_EVENT_AP_STOP no se encuentran tratados correctamente. Debido a esto, pasaremos a implementar manejadores espedificas para cada uno de ellos.

Tratando el evento WIFI_EVENT_AP_START

Este evento es enviado atomáticamente al ejecutar la funcion esp_wifi_start(), y para controlarlo vamos a enlazar una función manejadora que imprimirá en pantalla que se ha llevado correctamente el arranque del driver Wifi.

En el siguiente cuadro podemos ver la función manejadora en cuestión:

```
static void wifi_event_handler_stop(void* arg, esp_event_base_t event_base,
int32_t event_id, void* event_data)
{
    wifi_event_ap_staconnected_t* event = (wifi_event_ap_staconnected_t*)
event_data;
    ESP_ERROR_CHECK(esp_wifi_deinit());
    ESP_LOGI(TAG, "Detectada parada del driver WIFI. Cierre ordenado
realizado con exito. | "
                MACSTR" join, AID=%d",MAC2STR(event->mac), event->aid);
}
```

En el siguiente cuadro podemos ver el enlace de la función manejadora:

```
ESP_ERROR_CHECK(esp_event_handler_instance_register(WIFI_EVENT,
                                                    WIFI_EVENT_AP_STOP,

&wifi_event_handler_start,
                                                    NULL,
                                                    NULL));
```

En el siguiente cuadro tenemos la salida obtenida tras ejecutar el programa. Podmeos ver como después de terminar la configuración obtenemos el mensaje enviado desde la manejadora enlazada.

```
*
**CONFIGURACIÓN WIFI - FIN

I (940) esp_netif_lwip: DHCP server started on interface WIFI_AP_DEF with
IP: 192.168.4.1
I (950) wifi softAP: Driver WIFI iniciado correctamente.
I (960) wifi softAP: wifi_init_softap finished. SSID:RPI1_test
password:test1234 channel:1
I (970) main_task: Returned from app_main()
```

Tratando el evento WIFI_EVENT_AP_STOP

Este evento es enviado atomáticamente al ejecutar la funcion esp_wifi_stop() o en el caso de que se produzca un cierre repentino del driver WIFI. Igual que ne el caso anterior, vamos a enlazar una función manejadora que, en este caso, lleve a cabo un cierre ordenado de los recursos asignados al driver WIFI.

Antes de nada, para poder hacer esto utilizaremos un temporizador que tras pasados un número determinado de segundos cierre el driver WIFI. Esta será definida como una variable de confuguración accesible desde 'menuconfig', cuya definición dentro del fichero **Kconfig.projbuild** es:

```
    config ESP_TIME_CLOSE_WIFI
        int "Time to close the WIFI Access Point"
        range 10 99
        default 30
```

Por otra parte, el contador de segundos y la llamada a la función esp_wofo_stop se llevan a cabo al final de la función app_main(). En nuestro caso, como la placa STM32 se encuentra tabajando en modo Punto de Acceso, no necesitará aplicar la función **esp_wifi_disconnect()**, además de realziar la limpieza de recursos asignados mediante al función **esp_wifi_deinit()** una vez ha finalziado la manejadora del evento **ESP_EVENT_AP_STOP**.

En el siguiente cuadro podemos ver el fracmento de código indicado:

```
ESP_LOGI(TAG, "**CIERRE DEL PUNTO DE ACCESO EN %d SEGUNDOS",
CONFIG_ESP_TIME_CLOSE_WIFI) ;
for (int i = CONFIG_ESP_TIME_CLOSE_WIFI; i >= 0; i--)
{
    vTaskDelay(1000 / portTICK_PERIOD_MS);
}
ESP_LOGI(TAG, "**TIEMPO FINALIZADO, CERRANDO DRIVER WIFI") ;
ESP_ERROR_CHECK(esp_wifi_stop());
ESP_ERROR_CHECK(esp_wifi_deinit());
```

Por otra parte, en el siguiente cuadro podemos ver la función manejadora en cuestión:

```
static void wifi_event_handler_stop(void* arg, esp_event_base_t event_base,
int32_t event_id, void* event_data)
{
    ESP_LOGI(TAG, "Detectada parada del driver WIFI. Driver WIFI cerrado
con exito.");
}
```

En el siguiente cuadro podemos ver el enlace de la función manejadora:

```
ESP_ERROR_CHECK(esp_event_handler_instance_register(WIFI_EVENT,
                                                    WIFI_EVENT_AP_STOP,

&wifi_event_handler_stop,
                                                    NULL,
                                                    NULL));
```

Una vez implementado todo lo anterior, procedemos a la ajecución del dentregable y esperamos el tiempo necesario para que se lleve a cabo el cierre del driver WIFI. Obtenemos la siguiente salida

```
I (960) wifi softAP: wifi_init_softap finished. SSID:RPI1_test
password:test1234 channel:1
I (970) wifi softAP: **CIERRE DEL PUNTO DE ACCESO EN 30 SEGUNDOS
I (31980) wifi softAP: **TIEMPO FINALIZADO, CERRANDO DRIVER WIFI
E (31980) wifi:NAN WiFi stop
I (31980) wifi:flush txq
I (31980) wifi:stop sw txq
I (31980) wifi:lmac stop hw txq
I (31980) wifi softAP: Detectada parada del driver WIFI. Driver WIFI
cerrado con exito.
I (32000) wifi:Deinit lldesc rx mblock:10
I (32010) main_task: Returned from app_main()
```

MODO COMBINADO ESTACION / PUNTO DE ACCESO

> Entregable 3
>
> Modifica el ejemplo station para que el ESP32 se comporte a la vez como estación y como punto de acceso. Añade las opciones de configuración necesarias para que todos los parámetros se puedan modificar vía menuconfig . Comprueba que el ESP32 efectivamente se conecta al punto de acceso y que a la vez es posible conectar otro dispositivo al mismo (por ejemplo, tu teléfono móvil). Entrega el código.

En el presente apartado vamos a configurar la placa STM32 para que funcione como Punto de Acceso WIFI y al mismo tiempo pueda conectarse a otra red indicada como Estación. Para llevar esto a cabo vamos a partir del ejemplo **Station** y fusionaremos el código proporcionado con el desarrollado en el entregable 2.

El primer paso será definir nos menús de configuración, uno con los parámetros utilziados en modo Estacion y otro para los empleados cuando operamos como Punto de Acceso. Modificaremos levemente el nombre de las variables de configuración empleadas definidas en la configuración de AP con el objetivo de que no entren conflicto unas con otras.

En el siguiente cuadro podemos ver la definición de variables en el fichero **Kconfig.projbuild** utilizadas para la conexión con una red WIFI en el modo Estación:

```
menu "Example Configuration - Station Mode"

    config ESP_WIFI_SSID
        string "WiFi SSID"
        default "RPI1_test"
        help
            SSID (network name) for the example to connect to.

    config ESP_WIFI_PASSWORD
        string "WiFi Password"
        default "test1234"
        help
            WiFi password (WPA or WPA2) for the example to use.
```

En el siguiente cuadro podemos ver la definición de variables en el fichero **Kconfig.projbuild** utilizadas para la establecer la red WIFI en el modo Punto de Acceso:

```
menu "Example Configuration - Access Point Mode"

    config ESP_WIFI_AP_SSID
        string "WiFi SSID"
        default "RPI1_wifi"
        help
            SSID (network name) for the example to connect to.

    config ESP_WIFI_AP_PASSWORD
        string "WiFi Password"
        default "wifi1234"
        help
            WiFi password (WPA or WPA2) for the example to use.
```

Una vez establecidas las variables de configuración necesarias, deberemos enlazar la pila TCP/IP tanto para el uso como Punto de Acceso como Estación mediante el uso de las funciones **esp_netif_create_default_wifi_ap()** y **esp_netif_create_default_wifi_sta()** respectivamente. En el siguiente cuadro podremos ver la configuración de elementos previos, donde se puede apreciar la configuración de ambos elementos con las pilas:

```
//PASO 1..: Creamos e inicializamos los elementos necesarios para operar el
WIFI.
SP_ERROR_CHECK(esp_netif_init());
ESP_ERROR_CHECK(esp_event_loop_create_default());
esp_netif_create_default_wifi_sta();        //Creamos la interfaz con la
pila TCP/IP para el modo Estacion
esp_netif_create_default_wifi_ap();         //Creamos la interfaz con la
pila TCP/IP para el modo Punto de Acceso
```

Una vez establecida la configruación de los elementos previos, deberemos llevar a cabo la especificación de los parámetros a utilizar para cada conexión, siendo en nuestro caso dos, una para el modo Punto de Acceso y otra para el modo Estacion. Esta también nos llevará a indicar que el modo de operación para el driver WIFI será el modo convinado.

En el siguiente cuadro podemos ver la configuración tanto para el modo de operación combinado como para la conexión a realizar para ambos modos de operación:

```
    //PASO 2..: Configuramos la conexión a realizar y especificamos los
parámetros
    ESP_ERROR_CHECK(esp_wifi_set_mode(WIFI_MODE_APSTA) );       //Indicamos
el modo de conexión combinado

    wifi_config_t wifiConfig_STAmode = {
        .sta = {
            .ssid = EXAMPLE_ESP_WIFI_SSID,
            .password = EXAMPLE_ESP_WIFI_PASS,
            .threshold.authmode = ESP_WIFI_SCAN_AUTH_MODE_THRESHOLD,
            .sae_pwe_h2e = ESP_WIFI_SAE_MODE,
            .sae_h2e_identifier = EXAMPLE_H2E_IDENTIFIER,
        },
    };
    ESP_ERROR_CHECK(esp_wifi_set_config(WIFI_IF_STA, &wifiConfig_STAmode)
);     //Configuramos el modo Punto de Acceso


    wifi_config_t wifiConfig_APMode = {
        .ap = {
            .ssid = EXAMPLE_ESP_WIFI_AP_SSID,
            .ssid_len = strlen(EXAMPLE_ESP_WIFI_AP_SSID),
            .channel = EXAMPLE_ESP_WIFI_AP_CHANNEL,
            .password = EXAMPLE_ESP_WIFI_AP_PASS,
            .max_connection = EXAMPLE_ESP_WIFI_AP_MAX_CONN,
            #ifdef CONFIG_ESP_WIFI_SOFTAP_SAE_SUPPORT
                .authmode = WIFI_AUTH_WPA3_PSK,
                .sae_pwe_h2e = WPA3_SAE_PWE_BOTH,
            #else /* CONFIG_ESP_WIFI_SOFTAP_SAE_SUPPORT */
                .authmode = WIFI_AUTH_WPA2_PSK,
            #endif
            .pmf_cfg = {
                    .required = true,
```

```
            },
        },
    };
    if (strlen(EXAMPLE_ESP_WIFI_AP_PASS) == 0) {
        wifiConfig_APMode.ap.authmode = WIFI_AUTH_OPEN;
    }
    ESP_ERROR_CHECK(esp_wifi_set_config(WIFI_IF_AP, &wifiConfig_APMode) );
//Configuramos el modo Estación
```

Una vez realizados los pasos anteriores ya habremos configurado correctamenteel driver WIFI nuestra placa STM32 para funcionar en modo combinado, por lo que únicamente falta probarlo. Para esto primero conectaremos automáticamente nuestra placa a la red IP **RIP1_test**, y una ver realizado estableceremos la conexión de un segundo dispositivo a la red WIFI a la cual da soporte nyestra placa y a la que hemos llamado **RPI1_wifi**

En el siguiente cuadro tenemos la salida obtenida al llevar a cabo la conexión de nuestra placa a la red WIFI indicada en los parámetros de configuración:

```
I (6160) esp_netif_handlers: sta ip: 192.168.43.198, mask: 255.255.255.0,
gw: 192.168.43.1
I (6160) wifi station: got ip:192.168.43.198
I (6160) wifi station: connected to ap SSID:RPI1_test password:test1234
I (6170) main_task: Returned from app_main()
I (83240) wifi:new:<10,2>, old:<10,0>, ap:<10,2>, sta:<10,0>, prof:1
I (83240) wifi:station: f8:94:c2:b9:6b:08 join, AID=1, bgn, 40D
```

En el siguiente cuadro tenemos la salida obtenida al llevar a cabo la conexión de un tercer dispositivo a la red WIFI a la cual da soporte nuestra placa según indican los parámetros de configuración:

```
I (83260) wifi station: station f8:94:c2:b9:6b:08 join, AID=1
I (83360) esp_netif_lwip: DHCP server assigned IP to a client, IP is:
192.168.4.2
I (83450) wifi:<ba-add>idx:2 (ifx:1, f8:94:c2:b9:6b:08), tid:0, ssn:18,
winSize:64
```

ESCANEADO DE REDES WIFI

Analizando el ejemplo scan

> Tarea
>
> Compila, flashea y ejecuta el ejemplo de escaneado. Observa si los resultados son los esperados en el laboratorio o en un entorno doméstico. Modifica el código para conseguir distintos tipos de escaneado, asegurándote, por ejemplo, de que si fijas un canal específico en el que tu punto de

> acceso está trabajando, éste es detectado corretamente. Estudia y modifica los tiempos de espera en el escaneado y observa su efecto en el tiempo total de escaneado.

Para comenzar vamos a utilizar el código de ejemplo **scan** proporcioando por la plataforma, y tras un tiempo analizando, colocando y reorganizando las salidas para que este sea más visible, estamos listos para ejecutarlo y asi ver la salida proporcionada por el mismo.

En lo que corresponde a la configuración de los elementos previos, podemos ver como hemos incluido la especificación del país en el cual nos encontramos, con el objetivo de realizar un mejor escaneo de redes. A continuación podemos ver el fragmento de codigo mencionado y la salida obtenida de su ejecución:

```
//PASO 1..: REALIZACIÓN DEL ESCANEO
ESP_LOGI(TAG, "PASO 1..: CONFIGURACIÓN DEL ESCANEO");
esp_wifi_set_country_code("ES", true);
ESP_ERROR_CHECK(esp_wifi_start());
```

```
I (719) scan: PASO 1..: CONFIGURACIÓN DEL ESCANEO
I (719) wifi:set country: cc=ES schan=1 nchan=13 policy=0
I (729) phy_init: phy_version 4771,450c73b,Aug 16 2023,11:03:10
I (809) wifi:mode : sta (24:0a:c4:ea:36:b4)
I (809) wifi:enable tsf
```

A continuación, podemos ver como la función **esp_wifi_scan_start()**, encargada de realizar el escaneo de redes WIFI, no recibe ningúna estructura de configuración en su primer argumento, lo que le lleva a emplear la configuración por defecto, la cual especifica las siguientes opciones de escaneo:

- **show_hidden:false..:** Se ignoran los SSID ocultos.
- **scan_type:active..:** En escaneo se realiza de forma activa, es decir, re llevan a cabo el envío de paquetes **probe**.
- **scan_time.active.min:0..:** Tiempo mínimo durante el cual se llevará a cabo el escaneado activo.
- **scan_time.active.max:120 miliseconds..:** Tiempo máximo durante el cual se llevará a cabo el escaneado activo.
- **scan_time.passive:360 miliseconds..:** Tiempo durante el cual se llevará a cabo el escaneado pasivo.

A continuación podemos ver el fragmento de codigo mencionado y la salida obtenida de su ejecución, la cual esta completamente vacía, ya que no se ha producido ningún error y la función de escaneo no imprime nada por defecto:

```
//PASO 2..: REALIZACIÓN DEL ESCANEO
ESP_LOGI(TAG, "PASO 2..: REALIZACIÓN DEL ESCANEO");
esp_wifi_scan_start(NULL, true);
```

```
I (809) scan: PASO 2..: REALIZACIÓN DEL ESCANEO
```

Una vez especificado todo esto, vamos a llevar a cabo el escaneo de redes y observar los resultados obtenidos. Tener en cuenta que hemos activado la red WIFI **RPI1__test**, la cual estamos utilizando como prueba para analizar mejor los resultados. En el siguiente cuadro podemos ver el fragmento de codigo correspondiente a la obtención e impresión de las redes detectadas:

```
//PASO 3..: OBTENCIÓN DE RESULTADOS
ESP_LOGI(TAG, "PASO 3..: OBTENCIÓN DE RESULTADOS");
ESP_ERROR_CHECK(esp_wifi_scan_get_ap_records(&number, ap_info));
ESP_ERROR_CHECK(esp_wifi_scan_get_ap_num(&ap_count));

ESP_LOGI(TAG, "Total APs scanned = %u", ap_count);
for (int i = 0; (i < DEFAULT_SCAN_LIST_SIZE) && (i < ap_count); i++) {
    if (ap_info[i].authmode != WIFI_AUTH_WEP) {
        ESP_LOGI(TAG, "SSID: %s    RSSI: %d    %s    %s    CHANNEL: %d",
                ap_info[i].ssid,
                ap_info[i].rssi,
                print_auth_mode(ap_info[i].authmode),
                print_cipher_type(ap_info[i].pairwise_cipher,
ap_info[i].group_cipher),
                ap_info[i].primary);
    }else{
        ESP_LOGI(TAG, "SSID: %s    RSSI: %d    %s    CHANNEL: %d",
                ap_info[i].ssid,
                ap_info[i].rssi,
                print_auth_mode(ap_info[i].authmode),
                ap_info[i].primary);
    }
}
```

```
I (3319) scan: PASO 3..: OBTENCIÓN DE RESULTADOS
I (3319) scan: Total APs scanned = 12
I (3319) scan: SSID: RPI1_test    RSSI: -46    Authmode: WIFI_AUTH_WPA2_PSK
Pairwise Cipher: WIFI_CIPHER_TYPE_CCMP    CHANNEL: 10
I (3329) scan: SSID: UCM-CONGRESO    RSSI: -64    Authmode:
WIFI_AUTH_WPA2_PSK    Pairwise Cipher: WIFI_CIPHER_TYPE_CCMP    CHANNEL: 13
I (3339) scan: SSID: UCMOT    RSSI: -64    Authmode: WIFI_AUTH_WPA2_PSK
Pairwise Cipher: WIFI_CIPHER_TYPE_CCMP    CHANNEL: 13
I (3349) scan: SSID: eduroam    RSSI: -64    Authmode:
WIFI_AUTH_WPA2_ENTERPRISE    Pairwise Cipher: WIFI_CIPHER_TYPE_CCMP
CHANNEL: 13
I (3359) scan: SSID: UCM    RSSI: -64    Authmode: WIFI_AUTH_OPEN
Pairwise Cipher: WIFI_CIPHER_TYPE_NONE    CHANNEL: 13
I (3379) scan: SSID: AndroidAP477B    RSSI: -71    Authmode:
WIFI_AUTH_WPA2_PSK    Pairwise Cipher: WIFI_CIPHER_TYPE_CCMP    CHANNEL: 6
I (3389) scan: SSID: UCM-CONGRESO    RSSI: -76    Authmode:
WIFI_AUTH_WPA2_PSK    Pairwise Cipher: WIFI_CIPHER_TYPE_CCMP    CHANNEL: 9
I (3399) scan: SSID: UCMOT    RSSI: -77    Authmode: WIFI_AUTH_WPA2_PSK
Pairwise Cipher: WIFI_CIPHER_TYPE_CCMP    CHANNEL: 9
I (3409) scan: SSID: UCM    RSSI: -77    Authmode: WIFI_AUTH_OPEN
Pairwise Cipher: WIFI_CIPHER_TYPE_NONE    CHANNEL: 9
```

```
I (3429) scan: SSID: eduroam    RSSI: -78    Authmode:
WIFI_AUTH_WPA2_ENTERPRISE    Pairwise Cipher: WIFI_CIPHER_TYPE_CCMP
CHANNEL: 9
I (3439) main_task: Returned from app_main()
```

Podemos ver como hemos detectado una gran cantidad de redes WIFI, entre las que se encuentran tres redes relevantes que nos indican que el proceso se ha llevado a cabo correctamente:

- La red de pruebas:
  - SSID: RPI1_test RSSI: -46 Authmode: WIFI_AUTH_WPA2_PSK Pairwise Cipher: WIFI_CIPHER_TYPE_CCMP CHANNEL: 10.
- La red de la universidad:
  - SSID: UCM RSSI: -64 Authmode: WIFI_AUTH_OPEN Pairwise Cipher: WIFI_CIPHER_TYPE_NONE CHANNEL: 13.
- La red de la red Eduroan:
  - SSID: eduroam RSSI: -64 Authmode: WIFI_AUTH_WPA2_ENTERPRISE Pairwise Cipher: WIFI_CIPHER_TYPE_CCMP CHANNEL: 13.

A continuación vamos a observar los resultados obtenidos cuando llevamos a cabo algunas modificaciones en la configuración especificada, por ejemplo, escanear unicamente el canal donde esta trabajando al red **RPI1_test** (siendo en nuetsro caso el canal 10).

Para esto necesitaremos crear una estructura del tipo **wifi_scan_config_t** y pasarsela a la función **esp_wifi_scan_start()**. En los siguientes cuadros tenemos la configuración realizada y el resultado obtenido por pantalla:

```
ESP_LOGI(TAG, "PASO 2..: REALIZACIÓN DEL ESCANEO");
wifi_scan_config_t scan_config = {
    .show_hidden = false,
    .channel = 10,
    .scan_type = WIFI_SCAN_TYPE_ACTIVE,
    .scan_time.active.min = 1200,
    .scan_time.active.max = 2400,
    .scan_time.passive = 360
};
ESP_ERROR_CHECK(esp_wifi_scan_start(&scan_config, true));
```

```
I (2419) scan: PASO 3..: OBTENCIÓN DE RESULTADOS
I (2419) scan: Total APs scanned = 1
I (2419) scan: SSID: RPI1_test    RSSI: -35    Authmode: WIFI_AUTH_WPA2_PSK
Pairwise Cipher: WIFI_CIPHER_TYPE_CCMP    CHANNEL: 10
I (2429) main_task: Returned from app_main()
```

Como podemos ver, nuestra placa ha sido capaz de escanear la red WIFI de prueba que estamos utilizando, por lo que podemos deducir que el escaneo funciona correctamente. Además, también hemos aumentado

drasticamente el tiempo de escaneo en modo activo, lo que repercute en que la placa tarde más en finalziar el escaneo de cada uno de los canales.

Implementar la conexión a redes conocidas

> Entregable 4
>
> Diseña un firmware de nodo que realice un escaneado de las redes disponibles. Si el nodo detecta la presencia de una o más de las redes conocidas, se conectará en modo STA a la red de mayor prioridad entre las conocidas. Probadlo usando como redes conocidas la del laboratorio, vuestro móvil y vuestro domicilio.

> Entregable 5
>
> Codificar el código de la tarea anterior para que la lista de redes conocidas y la prioridad relativa se puedan configurar con menuconfig.

En este apartado se van a realizar conjuntamente los entregable 4 y 5, puesto que se piensa desarrollar el uso del menu de configuración a medida que se implementa la funcionalidad indicada. De este modo se han decidido crear dos secciones diferenciadas en el menú de configuración, siendo estas las siguientes:

- Example Configuration - Scan config: Opciones de configuración relacionadas al escaneo de redes WIFI.
- Example Configuration - Known networks: Opciones de configuración relacionadas con las redes conocidas.

En el siguiente cuadro podemos ver las entradas del menú de configuración dedicado al escaneo de redes WIFI, cuyas opciones coinciden con las proporcionadas en el ejemplo, además de haber agregado algunas como la visualización de redes ocultas o la modificación de los tiempos de escaneo.

```
menu "Example Configuration - Scan config"

    config EXAMPLE_SCAN_LIST_SIZE
        int "Número maximo de redes escaneadas"
        range 0 20
        default 10
        help
            Tamaño del sarray que será utilizado apra guardar la lista de
puntos de acceso encontrados.

    choice WIFICONF_SHOW_HIDDEN
        prompt "Escaneo de redes ocultas"
        default WIFICONF_SHOW_HIDDEN_DISABLE
        help
            Indicar si se quiere obtener información también sobre las redes
ocultas
        config WIFICONF_SHOW_HIDDEN_ENABLE
            bool "Habilitado"
        config WIFICONF_SHOW_HIDDEN_DISABLE
            bool "Desabilitado"
    endchoice
```

```
    config WIFICONF_CHANNEL
        int "Canal de escaneo"
        default 0
        help
            Indique el canal en el cual quiere realziar el escaneo (0 para
indicar todos los canales)-

    choice WIFICONF_SCAN_MODE
        prompt "Tipo de escaneo"
        default WIFICONF_SCAN_MODE_ACTIVE
        help
            indique el tipo de escaneo a realizar: Activo(se enviarán
mensajes de aviso) o Pasivo (únicamente se esperará a la recepción de
mensajes periódicos)
        config WIFICONF_SCAN_MODE_ACTIVE
            bool "Activo"
        config WIFICONF_SCAN_MODE_PASIVE
            bool "Pasivo"
    endchoice

    config WIFICONF_SCANTIME_ACTIVE_MIN
        int  "Tiempo mínimo de escaneo en el modo activo"
        range 20 60
        default 40

    config WIFICONF_SCANTIME_ACTIVE_MAX
        int  "Tiempo máximo de escaneo en el modo activo"
        range 200 20000
        default 200

    config WIFICONF_SCANTIME_PASSIVE
        int  "Tiempo de escaneo en el modo pasivo"
        range 200 20000
        default 200

endmenu
```

En el siguiente cuadro nso encontramos las opciones de configuración relacionadas con la expecificación y conexión con aquellas redes conocidas. Dentro de este podemos especificar datos como el número maximo de redes a indicar, el la cantidad de reintentos a la hora de llevar a cabo una conexión.

Dentro de todas estas, la opción más importante es la denominada **KNOWN_NETWORKS_LIST**, la cual os permite especificar la información referente a als redes que queremos indicar como conocidas. Estas redes deben especificarse en orden de relevancia y siguiendo el formato **[SSID:CONTRASEÑA] |_;** e indicando tantas como las que se indican en la variable **KNOWN_NETWORKS_LIST_SIZE**.

```
menu "Example Configuration - Known networks"

    config KNOWN_NETWORKS_LIST_SIZE
```

```
        int  "Número máximo de redes conocidas"
        range 1 20
        default 10

    config KNOWN_NETWORKS_LIST
        string "Lista de redes conocidas"
        default "RPI1_test:test1234 RPI1_2test:2test1234;"
        help
            Las redes se indcan en orden de prioridad descendente y
especificadas en el siguiente formato: "SSID_1:PASS_1 SSID_2:PASS_2..."

    config ESP_MAXIMUM_RETRY
        int "Maximum retry"
        default 5
        help
            Número maximo de intentos apra conectarse con el Punto de
Acceso conocido

    config ESP_WIFI_PW_ID
        string "PASSWORD IDENTIFIER"
        depends on  ESP_WPA3_SAE_PWE_HASH_TO_ELEMENT||
ESP_WPA3_SAE_PWE_BOTH
        default ""
        help
            password identifier for SAE H2E

    choice ESP_WIFI_SAE_MODE
        prompt "WPA3 SAE mode selection"
        default ESP_WPA3_SAE_PWE_BOTH
        help
            Select mode for SAE as Hunt and Peck, H2E or both.
        config ESP_WPA3_SAE_PWE_HUNT_AND_PECK
            bool "HUNT AND PECK"
        config ESP_WPA3_SAE_PWE_HASH_TO_ELEMENT
            bool "H2E"
        config ESP_WPA3_SAE_PWE_BOTH
            bool "BOTH"
    endchoice

    choice ESP_WIFI_SCAN_AUTH_MODE_THRESHOLD
        prompt "WiFi Scan auth mode threshold"
        default ESP_WIFI_AUTH_WPA2_PSK
        help
            The weakest authmode to accept in the scan mode.This value
defaults to ESP_WIFI_AUTH_WPA2_PSK incase password is present and
ESP_WIFI_AUTH_OPEN is used. Please select
ESP_WIFI_AUTH_WEP/ESP_WIFI_AUTH_WPA_PSK incase AP is operating in WEP/WPA
mode.

        config ESP_WIFI_AUTH_OPEN
            bool "OPEN"
        config ESP_WIFI_AUTH_WEP
            bool "WEP"
        config ESP_WIFI_AUTH_WPA_PSK
```

```
            bool "WPA PSK"
        config ESP_WIFI_AUTH_WPA2_PSK
            bool "WPA2 PSK"
        config ESP_WIFI_AUTH_WPA_WPA2_PSK
            bool "WPA/WPA2 PSK"
        config ESP_WIFI_AUTH_WPA3_PSK
            bool "WPA3 PSK"
        config ESP_WIFI_AUTH_WPA2_WPA3_PSK
            bool "WPA2/WPA3 PSK"
        config ESP_WIFI_AUTH_WAPI_PSK
            bool "WAPI PSK"
    endchoice
endmenu
```

Una vez hemos introducidos los datos relativos a las redes conocidas, estos serán extraidos y almacenados en la estructura **knownNetwork** mediante la función **splitKnownNetworks()**. La estructura indicada tiene un total de tres campos, cuya función es:

- uint8_t *sentence: Almacena la expreción indicada por el usuario en el menú de configuración sobre la red bíen conocida.
- uint8_t *SSID: Contiene el identificador de al red bien conocida.
- uint8_t *pass: Contiene la contraseña de acceso de la red bien conocida.

En los siguientes cuadros podemos ver tanto la estructura en cuestión como la salida obtenida al ejecutar el paso de **Configuración del escaneo**, el cual contiene la ejecución de la función indicada, entre otras.

```
struct knownNetwork {
    uint8_t *sentence;
    uint8_t *SSID;
    uint8_t *pass;
}knownNetwork;
```

```
I (721) scan: PASO 1..: CONFIGURACIÓN DEL ESCANEO
I (721) wifi:set country: cc=ES schan=1 nchan=13 policy=0
I (731) phy_init: phy_version 4771,450c73b,Aug 16 2023,11:03:10
I (811) wifi:mode : sta (24:0a:c4:ea:36:b4)
I (811) wifi:enable tsf
I (811) scan: EXTRAYENDO REDES CONOCIDAS: RPI1_test:test1234
RPI1_2test:2test1234;
I (821) scan: Redes conocidas..:
I (821) scan:   Prioridad 1)    (RPI1_test:test1234) SSID: RPI1_test
Password: test1234
I (831) scan:   Prioridad 2)    (RPI1_2test:2test1234) SSID: RPI1_2test
Password: 2test1234
```

Después de obtener y almacenar las redes bién conocidas, procedemos a realziar el escaneo de la misma manera que en el ejemeplo **scan** y podemos ver como obtenemos los siguientes resultados:

```
I (841) scan: PASO 2..: REALIZACIÓN DEL ESCANEO

I (3741) scan: PASO 3..: OBTENCIÓN DE RESULTADOS
I (3741) scan: Total APs scanned = 16
I (3741) scan: SSID: RPI1_test    RSSI: -38    Authmode: WIFI_AUTH_WPA2_PSK
Pairwise Cipher: WIFI_CIPHER_TYPE_CCMP    CHANNEL: 12
I (3751) scan: SSID: Oppo    RSSI: -49    Authmode: WIFI_AUTH_WPA2_PSK
Pairwise Cipher: WIFI_CIPHER_TYPE_CCMP    CHANNEL: 10
I (3761) scan: SSID: UCM-CONGRESO    RSSI: -55    Authmode:
WIFI_AUTH_WPA2_PSK    Pairwise Cipher: WIFI_CIPHER_TYPE_CCMP    CHANNEL: 1
I (3768) scan: SSID: RPI1_2test    RSSI: -38    Authmode:
WIFI_AUTH_WPA2_PSK    Pairwise Cipher: WIFI_CIPHER_TYPE_CCMP    CHANNEL: 14
I (3771) scan: SSID: eduroam    RSSI: -55    Authmode:
WIFI_AUTH_WPA2_ENTERPRISE    Pairwise Cipher: WIFI_CIPHER_TYPE_CCMP
CHANNEL: 1
I (3781) scan: SSID: UCM    RSSI: -55    Authmode: WIFI_AUTH_OPEN
Pairwise Cipher: WIFI_CIPHER_TYPE_NONE    CHANNEL: 1
I (3801) scan: SSID: UCM-CONGRESO    RSSI: -73    Authmode:
WIFI_AUTH_WPA2_PSK    Pairwise Cipher: WIFI_CIPHER_TYPE_CCMP    CHANNEL: 13
I (3811) scan: SSID: UCM-CONGRESO    RSSI: -74    Authmode:
WIFI_AUTH_WPA2_PSK    Pairwise Cipher: WIFI_CIPHER_TYPE_CCMP    CHANNEL: 13
I (3821) scan: SSID: eduroam    RSSI: -74    Authmode:
WIFI_AUTH_WPA2_ENTERPRISE    Pairwise Cipher: WIFI_CIPHER_TYPE_CCMP
CHANNEL: 13
I (3831) scan: SSID: UCM    RSSI: -74    Authmode: WIFI_AUTH_OPEN
Pairwise Cipher: WIFI_CIPHER_TYPE_NONE    CHANNEL: 13
I (3851) scan: SSID: UCMOT    RSSI: -74    Authmode: WIFI_AUTH_WPA2_PSK
Pairwise Cipher: WIFI_CIPHER_TYPE_CCMP    CHANNEL: 13
```

Para finalzar, comprobamos que entre las redes escaneadas se encuentra al menos una de las indicadas como bien conocidas, y si esto sucede, procedemos a ensamblar la estructura de configuración **wifi_config_t** mediante los datos indicados por el usuario y llevamos a cabo la conexión llamando a la función **esp_wifi_connect()**.

En los siguientes cuadros se puede ver el fragmento de código correspondiente a lo indicado y la salida obtenida por pantalla, en la cual observamos como se llevado automáticamente a cabo la conexión con la red **RPI1_test**.

```
wifi_config_t wifiConfig = {
    .sta = {
        .ssid = " ",
        .password = " ",
        .threshold.authmode = ESP_WIFI_SCAN_AUTH_MODE_THRESHOLD,
        .sae_pwe_h2e = ESP_WIFI_SAE_MODE,
        .sae_h2e_identifier = EXAMPLE_H2E_IDENTIFIER
    },
};
if(posToConnect==-1){
    ESP_LOGI(TAG, "NO SE HA DETECTADO NINGUNA RED CONOCIDA");
}else{
```

```
    for(int i=0 ; i<KNOWN_NETWORKS_SSID_SIZE ; i++){
        wifiConfig.sta.ssid[i] = knownNetworksList[posToConnect].SSID[i];
    }
    for(int i=0 ; i<KNOWN_NETWORKS_PASS_SIZE ; i++){
        wifiConfig.sta.password[i] =
knownNetworksList[posToConnect].pass[i];
    }

    ESP_LOGI(TAG, "INTENTANDO CONECTAR A: %s (%s)", wifiConfig.sta.ssid,
wifiConfig.sta.password);
    ESP_ERROR_CHECK(esp_wifi_set_config(WIFI_IF_STA, &wifiConfig));
    esp_wifi_connect();
}
```

```
I (3861) scan: PASO 4..: CONEXIÓN A RED CONOCIDA
I (3861) scan: INTENTANDO CONECTAR A: RPI1_test (test1234)
I (4731) wifi:new:<12,0>, old:<1,0>, ap:<255,255>, sta:<12,0>, prof:1
I (4731) wifi:state: init -> auth (b0)
I (4781) wifi:state: auth -> assoc (0)
I (4791) wifi:state: assoc -> run (10)
I (4851) wifi:connected with RPI1_test, aid = 1, channel 12, BW20, bssid =
ca:df:8d:9a:30:e2
I (4851) wifi:security: WPA2-PSK, phy: bgn, rssi: -40
I (4861) wifi:pm start, type: 1

I (4881) wifi:<ba-add>idx:0 (ifx:0, ca:df:8d:9a:30:e2), tid:0, ssn:0,
winSize:64
I (4931) wifi:AP's beacon interval = 102400 us, DTIM period = 2
I (5871) esp_netif_handlers: sta ip: 192.168.43.198, mask: 255.255.255.0,
gw: 192.168.43.1
```

Entregable 6 - wifi/wifi_enterprise

Para compilar y volcar el ejemplo en la placa se ha descargado el certificado desde este enlace y reemplazado el contenido en el fichero ca.pem dentro del directorio main.

También se han configurado los parámetros adecuados en el proyecto:

- SSID: eduroam
- Validate server: activo
- EAP method: TTLS
- Phase2 method for TTLS: PAP
- EAP ID: anonymous@ucm.es
- EAP USERNAME: (tu correo UCM)
- EAP PASSWORD: (tu contraseña UCM)

```
I (575) main_task: Calling app_main()
I (635) wifi:wifi driver task: 3ffbfb3c, prio:23, stack:6656, core=0
I (665) wifi:wifi firmware version: ce9244d
I (665) wifi:wifi certification version: v7.0
I (665) wifi:config NVS flash: enabled
I (665) wifi:config nano formating: disabled
I (665) wifi:Init data frame dynamic rx buffer num: 32
I (665) wifi:Init management frame dynamic rx buffer num: 32
I (675) wifi:Init management short buffer num: 32
I (675) wifi:Init dynamic tx buffer num: 32
I (685) wifi:Init static rx buffer size: 1600
I (685) wifi:Init static rx buffer num: 10
I (695) wifi:Init dynamic rx buffer num: 32
I (695) wifi_init: rx ba win: 6
I (695) wifi_init: tcpip mbox: 32
I (705) wifi_init: udp mbox: 6
I (705) wifi_init: tcp mbox: 6
I (715) wifi_init: tcp tx win: 5744
I (715) wifi_init: tcp rx win: 5744
I (715) wifi_init: tcp mss: 1440
I (725) wifi_init: WiFi IRAM OP enabled
I (725) wifi_init: WiFi RX IRAM OP enabled
I (735) example: Setting WiFi configuration SSID eduroam...
I (735) phy_init: phy_version 4670,719f9f6,Feb 18 2021,17:07:07
I (845) wifi:mode : sta (8c:aa:b5:b8:bf:f4)
I (845) wifi:enable tsf
I (855) main_task: Returned from app_main()
I (2645) wifi:new:<13,0>, old:<1,0>, ap:<255,255>, sta:<13,0>, prof:1
I (3805) wifi:state: init -> auth (b0)
I (3805) wifi:state: auth -> assoc (0)
I (3815) wifi:state: assoc -> run (10)
I (4525) wifi:connected with eduroam, aid = 2, channel 13, BW20, bssid =
00:dc:b2:51:14:22
I (4525) wifi:security: WPA2-ENT, phy: bgn, rssi: -52
I (4525) wifi:pm start, type: 1

I (4545) wifi:<ba-add>idx:0 (ifx:0, 00:dc:b2:51:14:22), tid:0, ssn:0,
winSize:64
I (4585) wifi:AP's beacon interval = 102400 us, DTIM period = 1
I (4855) example: ~~~~~~~~~~~
I (4855) example: IP:0.0.0.0
I (4855) example: MASK:0.0.0.0
I (4855) example: GW:0.0.0.0
I (4855) example: ~~~~~~~~~~~
I (5635) wifi:<ba-add>idx:1 (ifx:0, 00:dc:b2:51:14:22), tid:7, ssn:0,
winSize:64
I (6535) esp_netif_handlers: sta ip: 10.8.73.105, mask: 255.255.128.0, gw:
10.8.0.1
I (6535) example: Recibida la IP...
I (6855) example: ~~~~~~~~~~~
I (6855) example: IP:10.8.73.105
I (6855) example: MASK:255.255.128.0
```

```
I (6855) example: GW:10.8.0.1
I (6855) example: ~~~~~~~~~~~
```