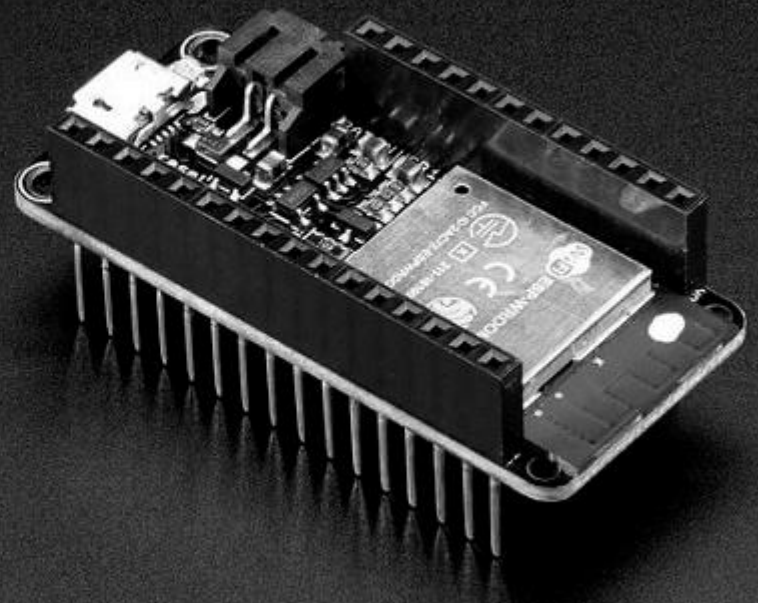
메이커 스페이스
G·캠프

중급과정

ESP32
온라인 워크숍



www.CodeZoo.co.kr

과정 목차

❖ 1일차

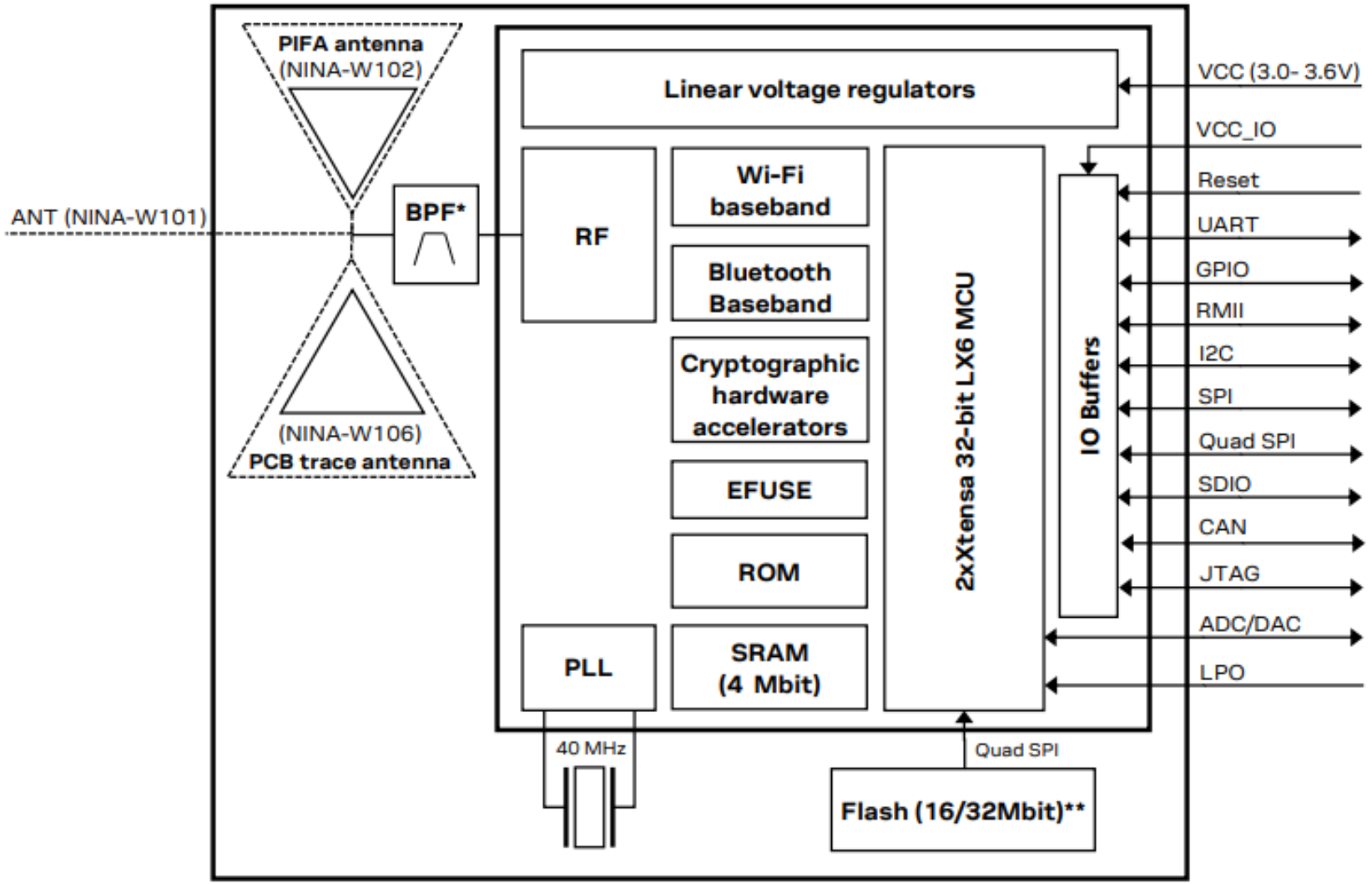
1. ESP32 Peripheral 과정 간단 리뷰 및 Network 과정 소개
2. lwip + WIFI
   > lwip 소개
   > WIFI Driver : 소개, Station, SoftAP, SCAN
   > TCP Client 실습

1. 시작하기

Arduino Nano 33 IoT

Ublox NINA-W102 (WIFI+BLE)

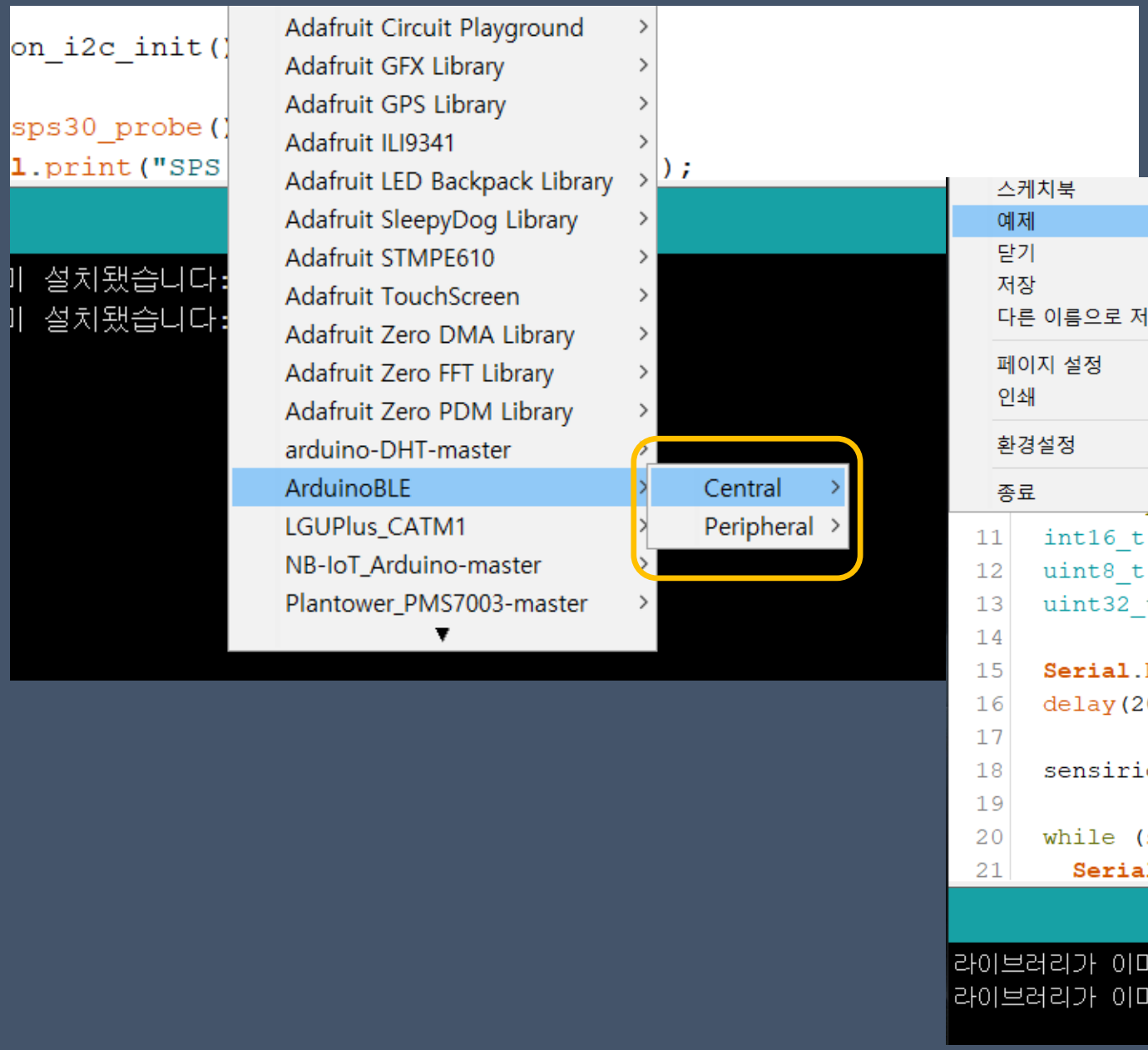
1.4 Block diagram

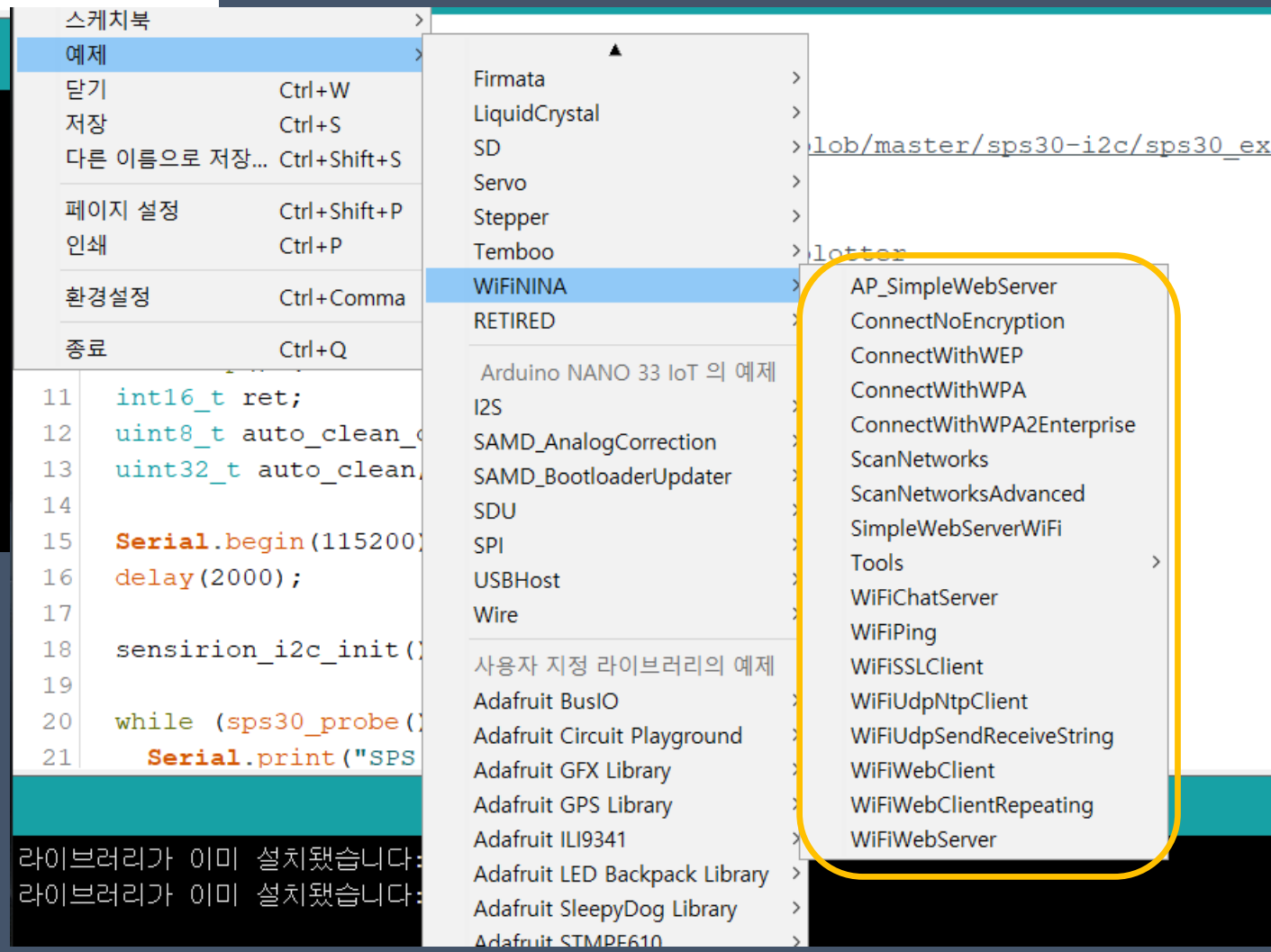


* Only on NINA-W101 and NINA-W102.

** 16 Mbit NINA-W101 and NINA-W102; 32 Mbit NINA-W106.

Figure 1: NINA-W10 series block diagram





ESP32-IDF Example

protocol

```
asio
cbor
coap_client
coap_server
esp_http_client
esp_local_ctrl
http2_request
http_request
http_server
https_mbedtls
https_request
https_server
https_x509_bundle
icmp_echo
mdns
modbus
mqtt
openssl_client
openssl_server
pppos_client
README.md
smtp_client
sntp
sockets
websocket
```

WIFI

```
espnow
fast_scan
getting_started
iperf
power_save
README.md
scan
simple_sniffer
smart_config
wpa2_enterprise
wps
```

lwIP (Lightweight TCP/IP stack)

LwIP ?
https://ko.wikipedia.org/wiki/LwIP

lwIP(lightweight IP)는 임베디드 시스템에서 널리 사용되는 **오픈 소스 TCP/IP 스택**이다. lwIP는 스웨덴 컴퓨터 과학 연구소 (Swedish Institute of Computer Science)의 Adam Dunkels 에 의해 처음 개발되었으며 현재는 전세계 개발자 네트워크에 의해 개발 및 유지 관리되고 있다.

lwIP는 많은 임베디드 시스템 제조업체에서 사용한다. 예로 알테라 (니오스 II 운영체제), 아날로그 디바이스 (블랙 핀 DSP의 칩),[1] 자일링스,[2] 하니웰 (FAA 인증 항법 시스템의 일부)과 프리 스케일 세미컨덕터 (자동차 마이크로 컨트롤러용 이더넷 스트리밍 SW)가 있다.

lwIP 네트워크 스택 구현의 초점은 풀 스케일 TCP 스택을 사용하면서도 리소스 사용을 줄이는 것이다.[3] 따라서 lwIP는 **수십 킬로바이트의 여유 RAM**과 **코드를 위한 약 40킬로바이트 이상의 ROM**이 있는 임베디드 시스템에서 사용하기에 적합하다.

출처: <https://ko.wikipedia.org/wiki/LwIP>

esp-lwip https://github.com/espressif/esp-lwip

Supported APIs
ESP-IDF supports the following lwIP TCP/IP stack functions:

- BSD Sockets API
- Netconn API - is enabled but not officially supported for ESP-IDF applications

Adapted APIs

아래 항목들은 ESP-IDF에서 간접적으로 지원

- DHCP 서버 및 클라이언트는 ESP-NETIF 기능을 통해 간접적으로 지원
- SNMP (Simple Network Time Protocol)는 lwip / include / apps / sntp / sntp.h lwip / lwip / src / include / lwip / apps / sntp.h 기능을 통해 지원 (SNTP 시간 동기화 참조)
- ICMP Ping은 lwIP ping API의 변형을 사용하여 지원 (ICMP 에코 참조)
- NetBIOS 조회는 표준 lwIP API를 사용하여 사용할 수 있음 protocols / http_server / restful_serve에는 NetBIOS를 사용하여 LAN에서 호스트를 찾는 방법을 보여주는 옵션이 있음
- mDNS는 lwIP 기본 mDNS (mDNS 서비스 참조)와는 다른 구현을 사용하지만 **CONFIG_LWIP_DNS_SNS_SUPPORT_MDNS_QUERIES** 설정이 활성화 된 경우 lwIP는 gethostbyname () 및 표준 hostname.local과 같은 표준 API를 사용하여 mDNS 호스트를 조회 할 수 있음

BSD Sockets API
BSD 소켓 API는 UNIX의 버클리 표준 배포판에서 시작되었지만 POSIX 사양의 섹션에서 표준화 된 일반적인 크로스 플랫폼 TCP / IP 소켓 API, BSD 소켓은 **POSIX 소켓 또는 버클리 소켓**이라고 함

ESP-IDF에서 구현 된 lwIP는 BSD 소켓 API의 모든 일반적인 사용법을 지원합니다.

References
Single UNIX Specification BSD Sockets page
Berkeley Sockets Wikipedia page

```
esp-idf/examples/protocols/sockets$ ls -al

096  6월 27 00:21 .
096  6월 27 00:21 ..
632  6월 27 00:21 README.md
096  6월 27 00:21 tcp_client
096  6월 27 00:21 tcp_client_multi_net
096  6월 27 00:21 tcp_server
096  6월 27 00:21 udp_client
096  6월 27 00:21 udp_multicast
096  6월 27 00:21 udp_server
```

제공되는 Socket 관련 Example

Socket 기반 서비스 다수 제공
coap, http_client, http,
websocket, mqtt 외

https://github.com/espressif/esp-lwip/blob/64b640e/src/include/lwip/sockets.h

- `socket()`
- `bind()`
- `accept()`
- `shutdown()`
- `getpeername()`
- `getsockopt()` & `setsockopt()` (see Socket Options)
- `close()` (via Virtual filesystem component)
- `read()`, `readv()`, `write()`, `writev()` (via Virtual filesystem component)
- `recv()`, `recvmsg()`, `recvfrom()`
- `send()`, `sendmsg()`, `sendto()`
- `select()` (via Virtual filesystem component)
- `poll()` (Note: on ESP-IDF, `poll()` is implemented by calling select internally, so using `select()` directly is recommended if a choice of methods is available.)
- `fcntl()` (see fcntl)

Non-standard functions:

- `ioctl()` (see ioctls)

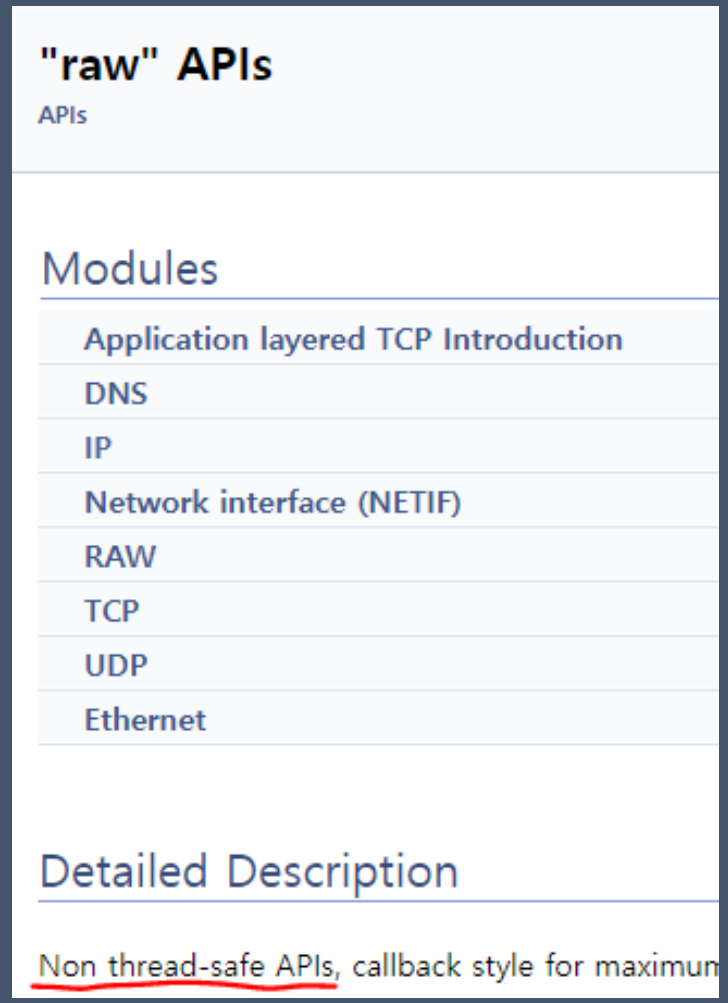
TCP/UDP
소켓 표준함수로
프로그래밍 할것

poll() 대신
select() 사용

Netconn API

lwIP는 Netconn API와 Raw API의 두 가지 하위 수준 API와 BSD 소켓 API를 지원하지만 (!)

lwIP Raw API는 단일 스레드 장치 용으로 설계되었으며 ESP-IDF에서 지원되지 않음



Netconn API는 lwIP 내에 BSD 소켓 API를 구현하는 데 사용되며 ESP-IDF 앱에서 직접 호출 할 수도 있음. 이 API는 BSD 소켓 API보다 리소스 사용량이 낮으며, 특히 내부 lwIP 버퍼에 데이터를 복사하지 않고도 데이터를 주고받을 수 있음

Important

Espressif는 ESP-IDF에서 ***Netconn API를 테스트하지 않음.*** *따라서 이 기능은 사용 가능하지만 지원되지 않음.* ***일부 기능은 BSD 소켓 API에서 사용될 때만 제대로 작동 할 수 있음***

결론,
BSD 소켓 API 적극사용

Netconn API

https://www.nongnu.org/lwip/2_0_x/api_8h.html

Data Structures

struct netconn

Macros

#define NETCONN_FLAG_NON_BLOCKING 0x02
#define NETCONN_FLAG_IN_NONBLOCKING_CONNECT 0x04
#define NETCONN_FLAG_CHECK_WRITESPACE 0x10
#define NETCONN_FLAG_IPV6_V6ONLY 0x20
#define API_EVENT(c, e, l)
#define NETCONN_SET_SAFE_ERR(conn, err)
#define netconn_new(t) netconn_new_with_proto_and_callback(t, 0, NULL)
#define netconn_type(conn) (conn->type)
#define netconn_peer(c, i, p) netconn_getaddr(c,i,p,0)
#define netconn_addr(c, i, p) netconn_getaddr(c,i,p,1)
#define netconn_listen(conn) netconn_listen_with_backlog(conn, TCP_DEFAULT_LISTEN_BACKLOG)
#define netconn_write(conn, dataptr, size, apiflags) netconn_write_partly(conn, dataptr, size, apiflags, NULL)
#define netconn_set_nonblocking(conn, val)
#define netconn_is_nonblocking(conn) (((conn)->flags & NETCONN_FLAG_NON_BLOCKING) != 0)
#define netconn_set_ipv6only(conn, val)
#define netconn_get_ipv6only(conn) (((conn)->flags & NETCONN_FLAG_IPV6_V6ONLY) != 0)
#define netconn_set_sendtimeout(conn, timeout) ((conn)->send_timeout = (timeout))
#define netconn_get_sendtimeout(conn) ((conn)->send_timeout)
#define netconn_set_recvbufsize(conn, recvbufsize) ((conn)->recv_bufsize = (recvbufsize))
#define netconn_get_recvbufsize(conn) ((conn)->recv_bufsize)

lwIP는 다른 작업의 소켓 API 요청을 처리하기 위한 **전용 TCP / IP FreeRTOS task**을 만듬

TCP / IP task와 데이터를 주고 받는 데 사용되는 task 및 queues ("mailboxes")을 수정하기 위해 여러 구성 항목을 설정 할 수 있음

- CONFIG_LWIP_TCPIP_RECVMBOX_SIZE
- CONFIG_LWIP_TCPIP_TASK_STACK_SIZE
- CONFIG_LWIP_TCPIP_TASK_AFFINITY

RTOS 동작을 위해 esp-lwip에서 수정한 내용

1. thread safe socket
다른 스레드에서 만든 스레드로 소켓을 닫을 수 있음. **close() 호출은 현재 다른 작업에서 해당 소켓을 사용하는 함수 호출이 반환 될 때까지 차단**

2. On demand timers (주문형 타이머)
lwIP IGMP 및 MLD6 기능은 특정 시간에 타임 아웃 이벤트를 트리거하기 위해 타이머를 초기화
IGMP (Internet Group Management Protocol)
인터넷에서 IPTV 와 같은 멀티캐스트 실시간 전송을 위해서 사용하는 프로토콜이다. IPTV에 쓰이는 핵심적인 프로토콜이지만, 그 외에는 사용하지 않고 있다.
MLD6 (Multicast listener discovery for IPv6) Aims to be compliant with RFC 2710. No support for MLDv2.

기본 lwIP 구현은 시간 초과 이벤트가 활성화되지 않은 경우에도 이러한 타이머를 항상 활성화. 이로 인해 자동 절전 모드를 사용할 때 CPU 사용량과 전력 소비가 증가. esp-lwip 기본 동작은 각 타이머를 "주문형"으로 설정하여 **이벤트가 보류(event is pending)**중인 경우에만 타이머 활성화

기본 lwIP 동작 (항상 켜짐 타이머)으로 돌아가려면 **CONFIG_LWIP_TIMERS_ONDEMAND를 disable(비활성화)** 설정

3. IP 변경시 TCP 연결 중단
CONFIG_LWIP_TCP_KEEP_CONNECTION_WHEN_IP_CHANGES는 **기본적으로 비활성화**되어 있음. 인터페이스 IP가 변경되는 경우 (예 : 인터페이스 연결이 끊어졌다가 다시 시작되는 경우) **인터페이스 IP가 변경되면 TCP 연결을 열린 상태로 유지하는 기본 lwIP 동작이 비활성화 됨.** 이 경우 TCP 연결이 정상적으로 시간 종료 될 때까지 열린 상태로 유지하려면 이 옵션을 사용 **네트워크 인터페이스가 일시적으로 다운되면 사용중인 소켓 수가 늘어날 수 있음**

RTOS 동작을 위해 esp-lwip에서 수정한 내용

4. Additional Socket Options(추가 소켓 옵션)
   일부 표준 IPV4 및 IPV6 멀티 캐스트 소켓 옵션이 구현 (소켓 옵션 참조).

5. IPV6_V6ONLY 소켓 옵션을 사용하여 IPV6 전용 UDP 및 TCP 소켓을 설정할 수 있음 (normal lwIP is TCP only).

6. IP layer features
   IPV4 소스 기반 라우팅 구현이 다름
   IPV4 매핑 IPV6 주소가 지원 됨

ESP lwip 제약사항

UDP 소켓에서 send () 또는 sendto ()를 반복적으로 호출하면 errno가 ENOMEM과 함께 실패 할 수 있음.

하위 계층 네트워크 인터페이스 드라이버의 **버퍼 크기 제한 때문** 임.

모든 드라이버 전송 버퍼가 가득 차면 UDP 전송이 실패 함.

발신자가 삭제하지 않으려는 대량의 UDP 데이터 그램을 보내는 응용 프로그램은 이 오류 코드를 확인하고

짧은 지연 후에 데이터 그램을 다시 보내야 함

==> ENOMEM 이 발생할 경우 vTaskDelay 등을 활용하여 약간의 지연을 만들어서 해결 (간단한 팁)

Performance Optimization

•Maximum throughput (최대 처리량)
wifi / iperf / sdkconfig.defaults 파일에는 일반적으로 RAM 사용량이 높아 TCP / IP 처리량을 최대화하는 것으로 알려진 설정이 포함되어 있음. 다른 요소를 희생하면서 응용 프로그램에서 최대 TCP / IP 처리량을 얻으려면 이 파일의 설정을 프로젝트 sdkconfig에 적용

• 한 번에 몇 가지 변경 사항을 적용하고 특정 응용 프로그램 작업 부하로 매번 성능을 확인하도록 제안

• 많은 작업이 시스템에서 CPU 시간을 놓고 경쟁하는 경우 lwIP 작업에 구성 가능한 CPU 선호도 (CONFIG_LWIP_TCPIP_TASK_AFFINITY)가 있고 고정 된 우선 순위 ESP_TASK_TCPIP_PRIO 로 실행되는 것을 고려. 경쟁 작업을 다른 코어에 고정하거나 낮은 우선 순위로 실행하도록 구성. (코어할당 0, 1, Task Priority 조정)

•소켓 인수만으로 select () 함수를 사용하는 경우 CONFIG_LWIP_USE_ONLY_LWIP_SELECT를 설정하면 select () 호출이 더 빨라집니다. (poll() 쓰지 말고 select() 사용할 것)

•Wi-Fi 네트워크 인터페이스를 사용하는 경우 **Wi-Fi 버퍼** 사용.

•Minimum latency (최소 대기 시간)
버퍼 크기 증가를 제외하고 처리량을 증가시키는 대부분의 변경 사항은 lwIP 기능에 소비되는 CPU 시간을 줄여 대기 시간을 줄임

•TCP 소켓의 경우 lwIP는 Nagle 알고리즘을 비활성화하기 위해 표준 TCP_NODELAY 플래그 설정을 지원

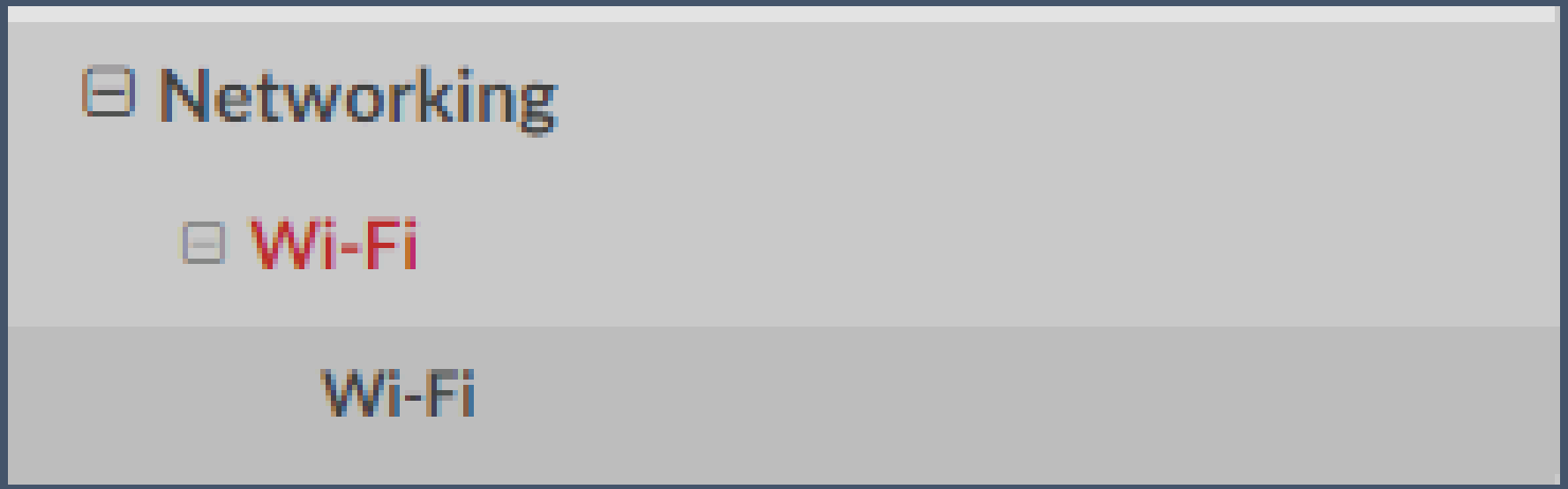
•Minimum RAM usage (최소 RAM 사용량)
RAM이 필요에 따라 힙에서 할당되므로 대부분의 lwIP RAM 사용량은 주문형. 따라서 lwIP 설정을 변경하여 RAM 사용을 줄이면 유휴 상태에서 RAM 사용이 변경되지 않지만 최대로 변경 될 수 있음

•CONFIG_LWIP_MAX_SOCKETS를 줄이면 시스템의 최대 소켓 수 조정. 또한 WAIT_CLOSE 상태의 TCP 소켓이 더 빨리 닫히고 (새로운 소켓을 여는 데 필요한 경우) 더 빨리 재활용되어 RAM 사용량이 최대로 줄어 듬

•CONFIG_LWIP_TCPIP_RECVMBOX_SIZE, CONFIG_LWIP_TCP_RECVMBOX_SIZE 및 CONFIG_LWIP_UDP_RECVMBOX_SIZE를 줄이면 사용량에 따라 처리량을 희생하여 메모리 사용량이 줄어 듬

•피크 버퍼 사용량
•lwIP가 사용하는 최대 힙 메모리는 이론적으로 lwIP 드라이버가 사용하는 최대 메모리 임. 일반적으로 lwIP가 사용하는 최대 힙 메모리는 다음에 따라 다름
LwIP가 사용하는 최대 힙 메모리는 다음 공식으로 계산할 수 있음
lwip_dynamic_peek_memory = (lwip_udp_con_num * lwip_udp_conn) + (lwip_tcp_con_num * (lwip_tcp_tx_win_size + lwip_tcp_rx_win_size + lwip_tcp_conn))

2. WIFI



위 내용은 API 설명만 있습니다.
실제 필요한 내용은 동작 방식을 정확하게 파악하는 것

여기서 찾으시면 앙데요.. ^^

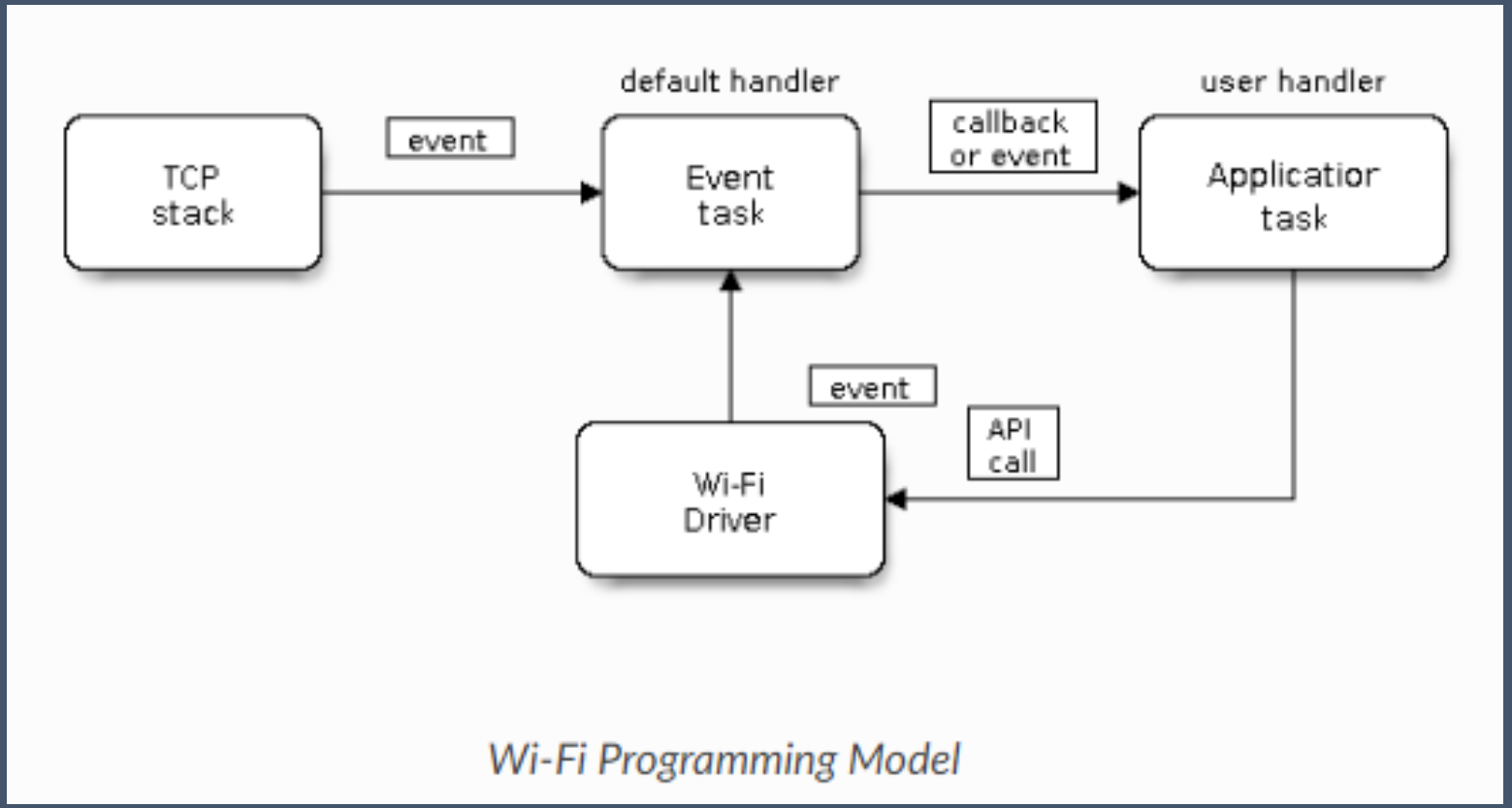
ESP32 Wi-Fi Feature List

- Support Station-only mode, AP-only mode, Station/AP-coexistence mode
- Support IEEE-802.11B, IEEE-802.11G, IEEE802.11N and APIs to configure the protocol mode
- Support WPA/WPA2/WPA2-Enterprise and WPS
- Support AMPDU, HT40, QoS and other key features
- Support Modem-sleep
- Support an Espressif-specific protocol which, in turn, supports up to 1 km of data traffic (최대 1km의 데이터 트래픽을 지원하는 Espressif 특정 프로토콜 지원)
- Up to 20 MBit/sec TCP throughput and 30 MBit/sec UDP throughput over the air
- Support Sniffer
- Support set fast_crypto algorithm and normal algorithm switch which used in wifi connect
- Support both fast scan and all channel scan feature
- Support multiple antennas
- Support channel state information

How To Write a Wi-Fi Application (Espressif 개발자의 말)

" <ESP32 Wi-Fi 프로그래밍 모델> 및 <ESP32 Wi-Fi 이벤트 설명>, <ESP32 Wi-Fi API 오류 코드> 섹션에 익숙해져야 합니다 "

ESP32 Wi-Fi Programming Model



Wi-Fi Programming Model

1. Wi-Fi 드라이버는 TCP / IP 스택, 응용 프로그램 작업, 이벤트 작업 등과 같은 상위 계층 코드에 대해서는 전혀 모르는 블랙 박스로 간주 될 수 있음.

2. 응용 프로그램 작업 (코드)은 일반적으로 Wi-Fi 드라이버 API를 호출하여 Wi-Fi를 초기화하고 필요한 경우 Wi-Fi 이벤트를 처리. Wi-Fi 드라이버는 API 호출을 수신하고 처리하며 응용 프로그램에 이벤트를 게시

3. Wi-Fi 이벤트 처리는 esp_event 라이브러리를 기반으로 진행. 이벤트는 Wi-Fi 드라이버에 의해 기본 이벤트 루프로 전송. 애플리케이션은 **esp_event_handler_register()**를 사용하여 등록 된 콜백에서 이러한 이벤트를 처리 할 수 있음. Wi-Fi 이벤트는 esp_netif 구성 요소에 의해 처리되어 기본 동작 집합을 제공.

ex) Wi-Fi 스테이션이 AP에 연결되면 esp_netif가 자동으로 DHCP 클라이언트를 시작(default)

ESP32 Wi-Fi Event Description

✓ WIFI_EVENT_WIFI_READY
Wi-Fi 드라이버는 이 이벤트를 생성하지 않으므로 결과적으로 응용 프로그램 이벤트 콜백에서 무시할 수 있음. 이 이벤트는 다음 릴리스에서 제거 될 수 있음.

✓ **WIFI_EVENT_SCAN_DONE**
스캔 완료 이벤트는 esp_wifi_scan_start ()에 의해 트리거되며 다음 시나리오에서 발생
- 스캔이 완료되었습니다 (예 : 대상 AP가 성공적으로 발견되었거나 모든 채널이 스캔)
- 스캔은 esp_wifi_scan_stop ()에 의해 중지
스캔이 완료되기 전에 esp_wifi_scan_start ()가 호출. 새로운 스캔은 현재 스캔을 무시하고 스캔 완료 이벤트가 생성

다음 시나리오에서는 스캔 완료 이벤트가 발생하지 않음

- 스캔이 차단되었습니다.
스캔은 esp_wifi_connect ()에 의해 발생

이 이벤트를 수신하면 이벤트 작업이 수행하지 않음. 응용 프로그램 이벤트 콜백은 esp_wifi_scan_get_ap_num () 및 esp_wifi_scan_get_ap_records ()를 호출하여 스캔 된 AP 목록을 가져오고 Wi-Fi 드라이버를 트리거하여 스캔 중에 할당 된 내부 메모리를 해제해야 함 (주의!!)

✓ WIFI_EVENT_STA_START

esp_wifi_start()가 ESP_OK를 반환하고 현재 Wi-Fi 모드가 스테이션 또는 AP + 스테이션이면 해당 이벤트가 발생합니다. 이 이벤트를 수신하면 이벤트 작업이 LwIP 네트워크 인터페이스 (netif)를 초기화합니다. 일반적으로 응용 프로그램 이벤트 콜백은 esp_wifi_connect ()를 호출하여 구성된 AP에 연결해야 합니다.

```
44 static void event_handler(void* arg, esp_event_base_t event_base,
45                                 int32_t event_id, void* event_data)
46 {
47     if (event_base == WIFI_EVENT && event_id == WIFI_EVENT_STA_START) {
48         esp_wifi_connect();
```

✓ WIFI_EVENT_STA_STOP

esp_wifi_stop ()이 ESP_OK를 반환하고 현재 Wi-Fi 모드가 스테이션 또는 AP + 스테이션이면 해당 이벤트가 발생합니다. 이 이벤트를 수신하면 이벤트 작업이 스테이션의 IP 주소를 해제하고 DHCP 클라이언트를 중지하며 TCP / UDP 관련 연결을 제거하고 LwIP 스테이션 netif 등을 지웁니다. 일반적으로 응용 프로그램 이벤트 콜백은 아무것도 할 필요가 없습니다.

✓ WIFI_EVENT_STA_CONNECTED

esp_wifi_connect ()가 ESP_OK를 반환하고 스테이션이 대상 AP에 성공적으로 연결되면 연결 이벤트가 발생합니다. 이 이벤트를 수신하면 이벤트 작업이 DHCP 클라이언트를 시작하고 IP 주소를 가져 오는 DHCP 프로세스를 시작합니다. 그런 다음 Wi-Fi 드라이버가 데이터를 송수신 할 준비가 되었습니다. 이 순간은 응용 프로그램이 LwIP, 즉 IP 주소에 의존하지 않는 한 응용 프로그램 작업을 시작하는 데 좋습니다. 그러나 **응용 프로그램이 LwIP 기반 인 경우, ip 이벤트가 수신 될 때까지 기다려야 합니다.**

```
} 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);
}
```

✓ WIFI_EVENT_STA_DISCONNECTED

이 이벤트는 다음 시나리오에서 생성 될 수 있습니다.

✓ IP_EVENT_STA_GOT_IP
이 이벤트는 DHCP 클라이언트가 DHCP 서버에서 IPV4 주소를 성공적으로 가져 오거나 IPV4 주소가 변경 될 때 발생합니다. 이 이벤트는 모든 준비가 완료되었으며 응용 프로그램이 작업을 시작할 수 있음을 의미합니다 (예 : 소켓 만들기).

✓ IP_EVENT_GOT_IP6
이 이벤트는 IPV6 SLAAC 지원이 ESP32의 주소를 자동 구성하거나이 주소가 변경 될 때 발생합니다. 이 이벤트는 모든 준비가 완료되었으며 응용 프로그램이 작업을 시작할 수 있음을 의미합니다 (예 : 소켓 만들기).

✓ IP_STA_LOST_IP
이 이벤트는 IPV4 주소가 유효하지 않을 때 발생합니다.

✓ WIFI_EVENT_STA_DISCONNECTED
이 이벤트는 다음 시나리오에서 생성 될 수 있습니다.

✓ WIFI_EVENT_AP_START
<WIFI_EVENT_STA_START>와 유사합니다.

✓ WIFI_EVENT_AP_STOP
<WIFI_EVENT_STA_STOP>과 유사합니다.

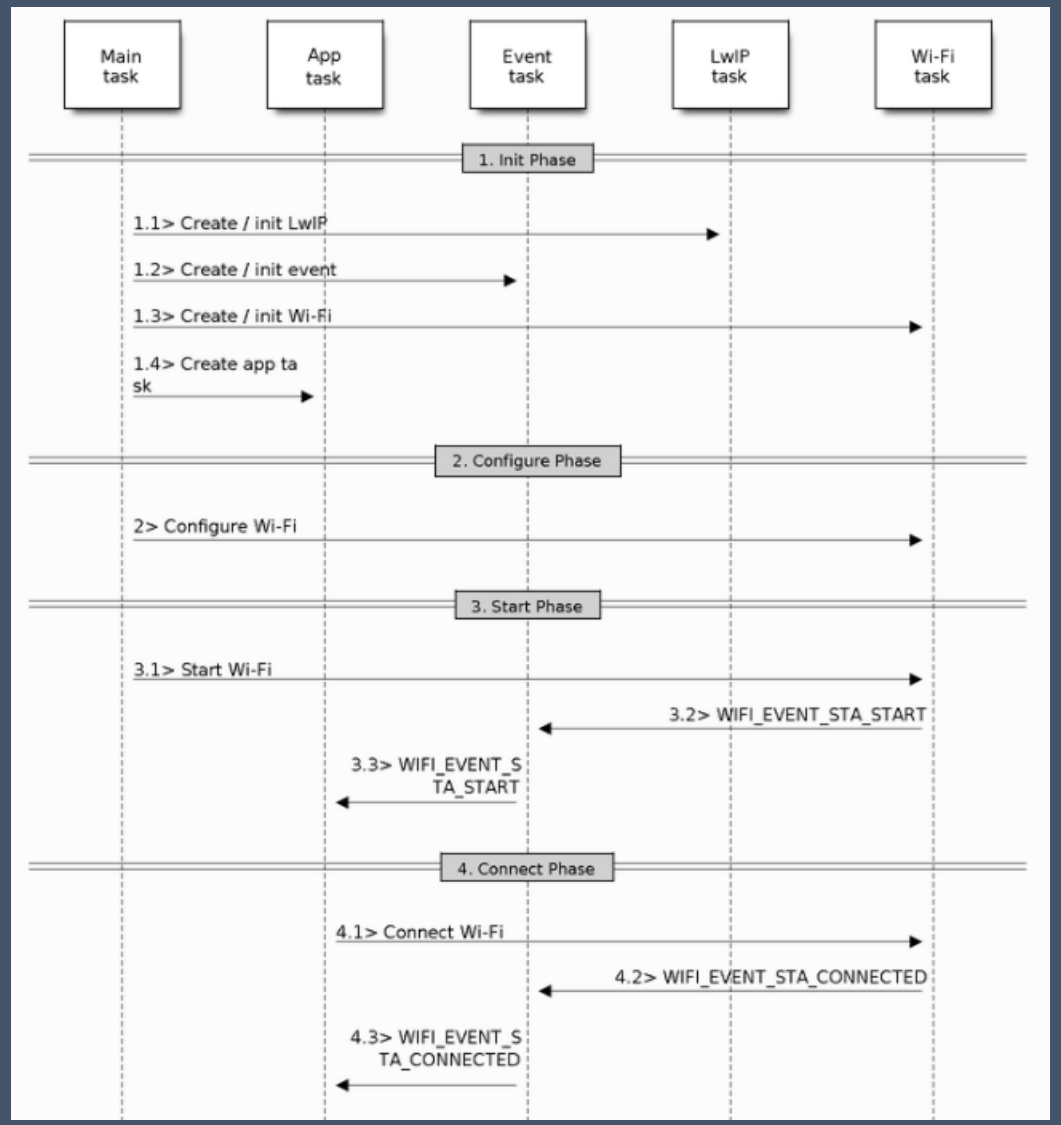
✓ WIFI_EVENT_AP_STACONNECTED
스테이션이 ESP32 AP에 연결될 때마다 <WIFI_EVENT_AP_STACONNECTED>가 발생합니다. 이 이벤트를 수신하면 이벤트 작업이 아무 것도 수행하지 않으며 응용 프로그램 콜백도이를 무시할 수 있습니다. 그러나 연결된 STA의 정보 등을 얻는 등의 작업을 수행 할 수 있습니다.

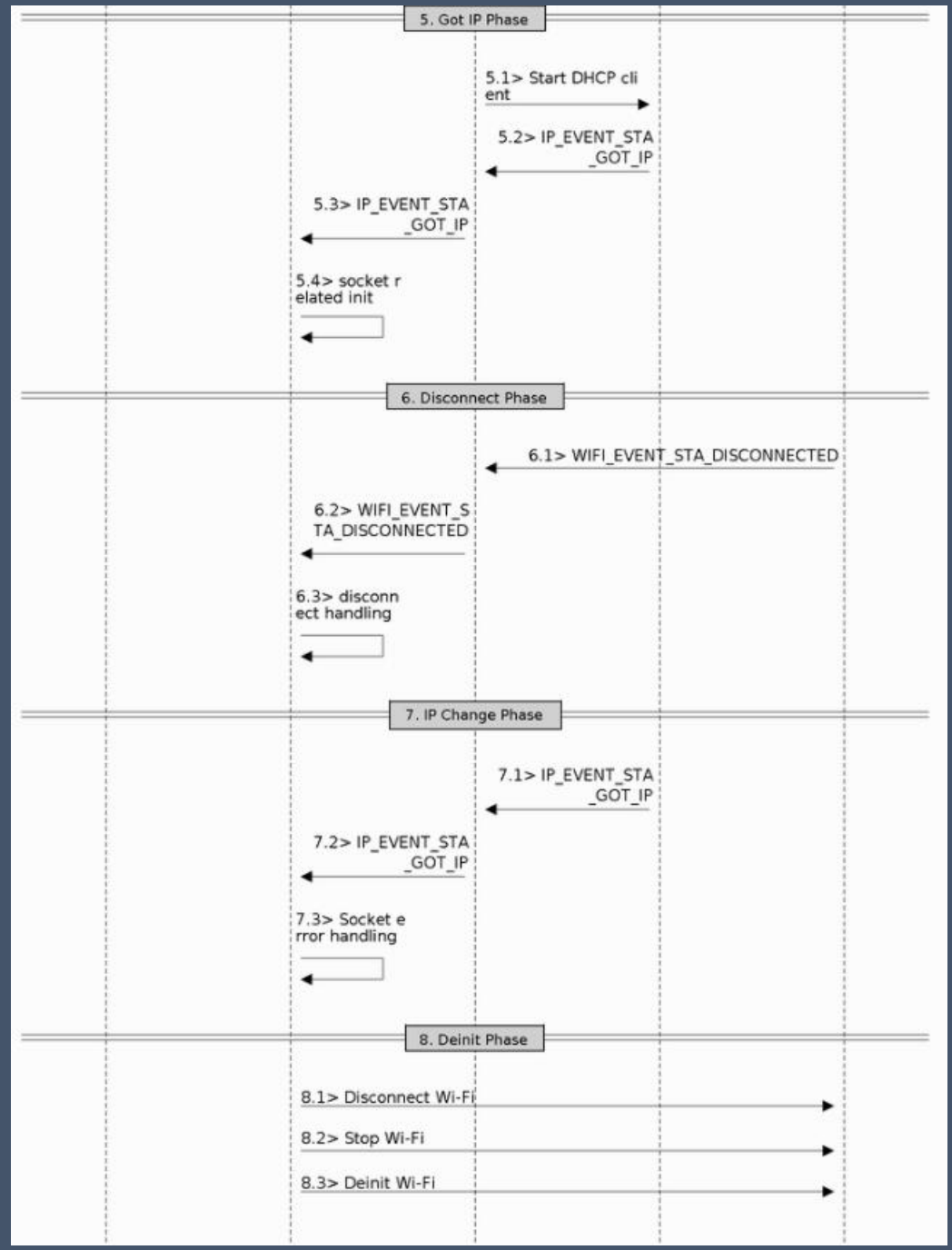
```
33 static void wifi_event_handler(void* arg, esp_event_base_t event_base,
34                                 int32_t event_id, void* event_data)
35 {
36     if (event_id == WIFI_EVENT_AP_STACONNECTED) {
37         wifi_event_ap_staconnected_t* event = (wifi_event_ap_staconnected_t*) event_data;
38         ESP_LOGI(TAG, "station "MACSTR" join, AID=%d",
39                  MAC2STR(event->mac), event->aid);
40     } else if (event_id == WIFI_EVENT_AP_STADISCONNECTED) {
41         wifi_event_ap_stadisconnected_t* event = (wifi_event_ap_stadisconnected_t*) event_data;
42         ESP_LOGI(TAG, "station "MACSTR" leave, AID=%d",
43                  MAC2STR(event->mac), event->aid);
44     }
45 }
```

✓ WIFI_EVENT_AP_STADISCONNECTED
이 이벤트는 다음 시나리오에서 발생할 수 있습니다.

Wi-Fi 초기화 시나리오

ESP32 Wi-Fi Station General Scenario





1. Init Phase

s1.1 : 기본 작업은 **esp_netif_init ()**를 호출하여 LwIP 핵심 작업을 생성하고 LwIP 관련 작업을 초기화합니다.
--> ESP_ERROR_CHECK(esp_netif_init());

s1.2 : 기본 작업은 **esp_event_loop_init ()**를 호출하여 시스템 이벤트 작업을 생성하고 응용 프로그램 이벤트의 콜백 함수를 초기화합니다. 위의 시나리오에서 응용 프로그램 이벤트의 콜백 함수는 이벤트를 응용 프로그램 작업에 릴레이하는 것 외에는 아무것도하지 않습니다.
--> esp_err_t esp_event_loop_init(system_event_cb_t cb, void *ctx) __attribute__ ((**deprecated**));
--> ESP_ERROR_CHECK(esp_event_loop_create_default());

s1.3 : 기본 작업은 **esp_netif_create_default_wifi_ap()** 또는 **esp_netif_create_default_wifi_sta()**를 호출하여 TCP/IP 스택을 사용하여 기본 네트워크 인터페이스 인스턴스 바인딩 스테이션 또는 AP를 만듭니다.

s1.4 : 기본 작업은 **esp_wifi_init ()**를 호출하여 **Wi-Fi 드라이버 작업을 생성하고 Wi-Fi 드라이버를 초기화**합니다.

s1.5 : 기본 작업은 OS API를 호출하여 응용 프로그램 작업을 만듭니다.

1.1 ~ 1.5 단계는 Wi-Fi / LwIP 기반 응용 프로그램을 초기화하는 권장 순서입니다. 그러나 반드시 따라야하는 순서는 아닙니다. 따라서 1.1 단계에서 응용 프로그램 작업을 만들고 다른 모든 초기화를 응용 프로그램 작업에 넣을 수 있습니다. 또한 응용 프로그램 작업이 소켓에 의존하는 경우 초기화 단계에서 응용 프로그램 작업을 생성하지 않을 수 있습니다. 대신 IP를 얻을 때까지 작업 생성을 연기 할 수 있습니다.

2. Wi-Fi Configuration Phase

Wi-Fi 드라이버가 초기화되면 Wi-Fi 드라이버 구성을 시작할 수 있습니다. 이 시나리오에서 모드는 스테이션이므로 Wi-Fi 모드를 스테이션으로 구성하려면 esp_wifi_set_mode (WIFI_MODE_STA)를 호출해야합니다. 다른 esp_wifi_set_xxx API를 호출하여 프로토콜 모드, 국가 코드, 대역폭 등과 같은 추가 설정을 구성 할 수 있습니다. <ESP32 Wi-Fi 구성>을 참조하십시오.

```
ESP_ERROR_CHECK(esp_wifi_set_mode(WIFI_MODE_STA) );
ESP_ERROR_CHECK(esp_wifi_set_config(ESP_IF_WIFI_STA, &wifi_config) );
```

일반적으로 Wi-Fi 연결을 설정하기 전에 Wi-Fi 드라이버를 구성하지만 필수 사항은 아닙니다. 즉, Wi-Fi 드라이버가 성공적으로 초기화 된 경우 언제든지 Wi-Fi 연결을 구성 할 수 있습니다. 그러나 Wi-Fi 연결을 설정 한 후 구성을 변경할 필요가없는 경우 구성 API (예 : esp_wifi_set_protocol)로 인해 Wi-Fi가 다시 연결되므로이 단계에서 Wi-Fi 드라이버를 구성해야합니다. 바람직하지 않을 수 있습니다.

menuconfig에서 Wi-Fi NVS 플래시를 사용하도록 설정하면이 단계 이후의 모든 Wi-Fi 구성이 플래시에 저장됩니다. 보드 전원을 켜거나 재부팅 할 때 Wi-Fi 드라이버를 처음부터 구성 할 필요가 없습니다. 이전에 플래시에 저장된 구성을 가져 오려면 esp_wifi_get_xxx API 만 호출하면됩니다. 이전 구성이 원하는 것이 아닌 경우 Wi-Fi 드라이버를 구성할 수도 있습니다.

3. Wi-Fi Start Phase

s3.1 : esp_wifi_start를 호출하여 Wi-Fi 드라이버를 시작하십시오.
ESP_ERROR_CHECK(esp_wifi_start());

s3.2 : Wi-Fi 드라이버는 <WIFI_EVENT_STA_START>를 이벤트 작업에 게시합니다. 그런 다음 이벤트 작업은 몇 가지 일반적인 작업을 수행하고 응용 프로그램 이벤트 콜백 함수를 호출합니다.

s3.3 : 응용 프로그램 이벤트 콜백 기능은 <WIFI_EVENT_STA_START>를 응용 프로그램 작업으로 릴레이합니다. esp_wifi_connect ()를 호출하는 것이 좋습니다. 그러나 <WIFI_EVENT_STA_START>가 발생한 후 다른 구에서 esp_wifi_connect ()를 호출 할 수도 있습니다.

```
if (event_base == WIFI_EVENT && event_id == WIFI_EVENT_STA_START) {
    esp_wifi_connect();
```

4. Wi-Fi Connect Phase

s4.1 : esp_wifi_connect()가 호출되면 Wi-Fi 드라이버는 내부 스캔 / 연결 프로세스를 시작합니다.

s4.2 : 내부 스캔 / 연결 프로세스가 성공하면 <WIFI_EVENT_STA_CONNECTED>가 생성됩니다. 이벤트 작업에서 DHCP 클라이언트를 시작하면 DHCP 프로세스가 트리거됩니다.

s4.3 : 위에서 언급 한 시나리오에서 응용 프로그램 이벤트 콜백은 이벤트를 응용 프로그램 작업으로 릴레이합니다. 일반적으로 응용 프로그램은 아무것도 할 필요가 없으며 로그 인쇄 등 원하는 모든 작업을 수행 할 수 있습니다.

4.2 단계에서 비밀번호가 잘못되었거나 AP를 찾을 수없는 등의 이유로 Wi-Fi 연결이 실패 할 수 있습니다. 이와 같은 경우 <WIFI_EVENT_STA_DISCONNECTED>가 발생하고 이러한 실패의 원인이 제공됩니다. . Wi-Fi 연결을 방해하는 이벤트를 처리하려면 6 단계를 참조하십시오.

5. Wi-Fi 'Got IP' Phase

s5.1 : 단계 4.2에서 DHCP 클라이언트가 초기화되면 IP 가져 오기 단계가 시작됩니다.

s5.2 : IP 주소가 DHCP 서버로부터 성공적으로 수신되면 <IP_EVENT_STA_GOT_IP>가 발생하고 이벤트 작업이 일반적인 처리를 수행합니다.

s5.3 : 응용 프로그램 이벤트 콜백에서 <IP_EVENT_STA_GOT_IP>가 응용 프로그램 작업으로 릴레이됩니다. LwIP 기반 응용 프로그램의 경우이 이벤트는 매우 특별하며 응용 프로그램이 작업을 시작할 수있는 모든 준비가 완료되었음을 의미합니다. TCP / UDP 소켓 작성 등. 매우 일반적인 실수는 <IP_EVENT_STA_GOT_IP>가 수신되기 전에 소켓을 초기화하는 것입니다. IP를 받기 전에 소켓 관련 작업을 시작하지 마십시오.

```
} 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);
}
```

6. Wi-Fi 연결 해제 단계

s6.1 : Wi-Fi 연결이 중단 된 경우 (예 : AP의 전원이 꺼졌기 때문에 RSSI가 불량합니다. <WIFI_EVENT_STA_DISCONNECTED>가 발생합니다. 이 이벤트는 3 단계에서도 발생할 수 있습니다. 여기서 이벤트 작업은 모든 UDP/TCP 연결을 지우거나 제거하도록 LwIP 작업에 알립니다. 그런 다음 모든 응용 프로그램 소켓의 상태가 잘못됩니다. 즉, 해당 이벤트가 발생하면 소켓이 제대로 작동하지 않습니다.

s6.2 : 위에서 설명한 시나리오에서 응용 프로그램 이벤트 콜백 함수는 <WIFI_EVENT_STA_DISCONNECTED>를 응용 프로그램 작업으로 릴레이합니다. Wi-Fi를 다시 연결하고 모든 소켓을 닫은 후 필요한 경우 다시 작성하려면 esp_wifi_connect ()를 호출하는 것이 좋습니다. <WIFI_EVENT_STA_DISCONNECTED>를 참조하십시오.

7. Wi-Fi IP 변경 단계

s7.1 : IP 주소가 변경되면 "ip_change"가 true로 설정된 상태에서 <IP_EVENT_STA_GOT_IP>가 발생합니다.

s7.2 :이 이벤트는 응용 프로그램에 중요합니다. 발생하면 타이밍은 작성된 모든 소켓을 닫고 다시 만드는게 좋습니다.

8. Wi-Fi 초기화 해제 단계

s8.1 : esp_wifi_disconnect ()를 호출하여 Wi-Fi 연결을 끊습니다.

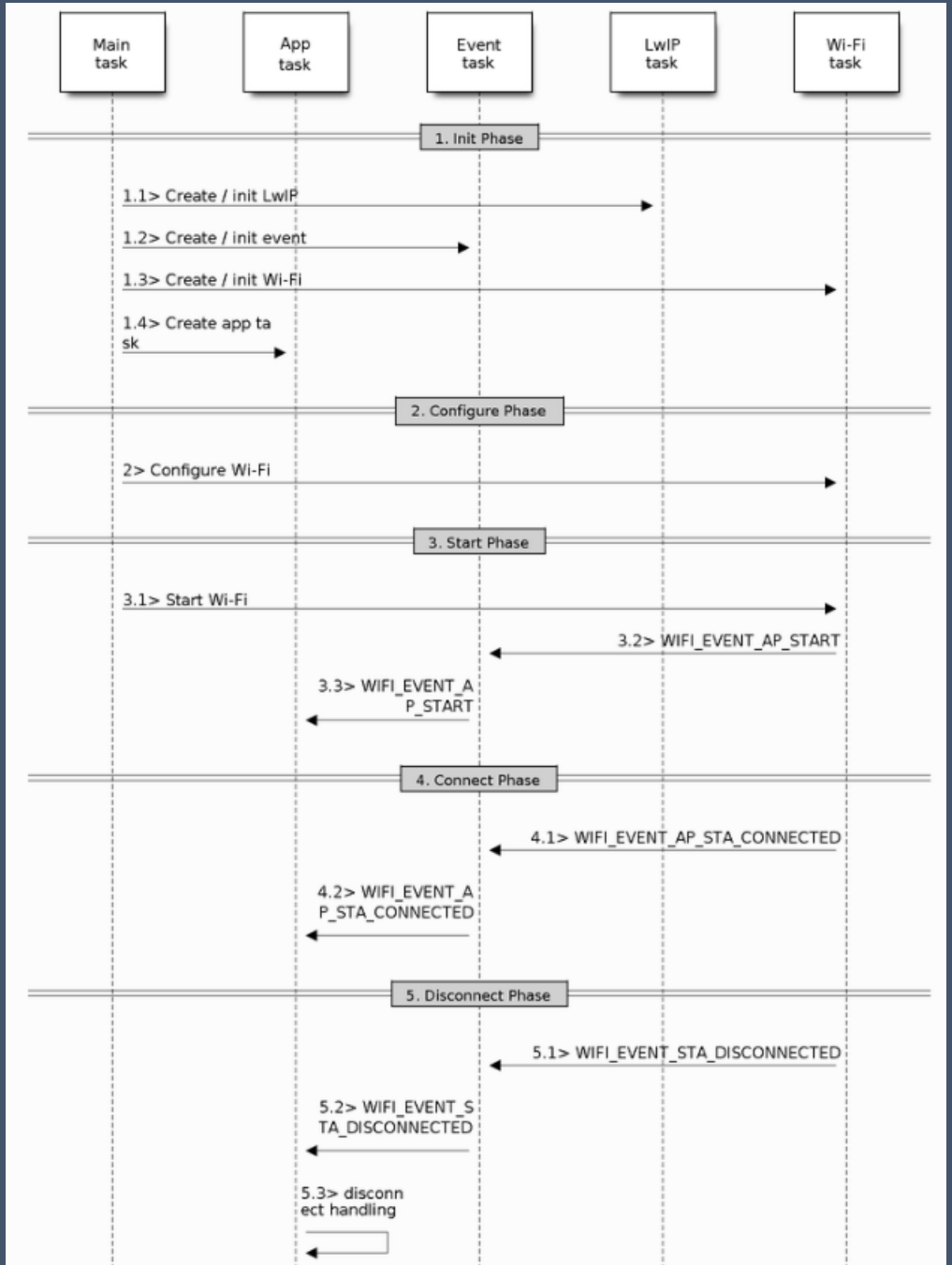
s8.2 : esp_wifi_stop ()을 호출하여 Wi-Fi 드라이버를 중지하십시오.

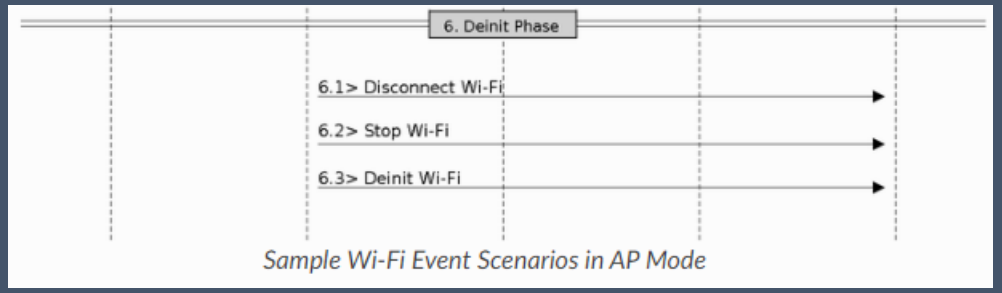
s8.3 : esp_wifi_deinit ()를 호출하여 Wi-Fi 드라이버를 언로드하십시오.

코드리뷰
station_example_main.c

실습
동작 여부와 IP 획득 체크

ESP32 Wi-Fi AP General Scenario





Sample Wi-Fi Event Scenarios in AP Mode

ESP32 Wi-Fi AP General Scenario

코드리뷰
softap_example_main.c

실습
동작 여부와 스마트폰으로 접속 확인 체크

ESP32 Wi-Fi API Error Code

모든 ESP32 Wi-Fi API에는 잘 정의 된 반환 값, 즉 오류 코드가 있습니다. 오류 코드는 다음과 같이 분류 할 수 있습니다.
오류 없음 (예 : ESP_OK는 API가 성공적으로 반환 함을 의미합니다

✓ ESP_ERR_NO_MEM 등과 같은 복구 가능한 오류

✓ 복구 불가능하고 중요하지 않은 오류

복구 불가능하고 치명적인 오류

오류가 중요한지 여부는 API 및 애플리케이션 시나리오에 따라 다르며 API 사용자가 정의합니다.

Wi-Fi API를 사용하여 강력한 응용 프로그램을 작성하는 기본 원칙은 항상 오류 코드를 확인하고 오류 처리 코드를 작성하는 것입니다. 일반적으로 오류 처리 코드를 사용할 수 있습니다.

복구 가능한 오류의 경우 복구 가능한 오류 코드를 작성할 수 있습니다. 예를 들어 esp_wifi_start가 ESP_ERR_NO_MEM을 반환하면 복구 시도 오류 코드 vTaskDelay를 호출하여 다른 시도에 대한 마이크로 초의 지연을 얻을 수 있습니다.

복구 불가능하지만 치명적이지 않은 오류의 경우 오류 코드를 print(인쇄)하는 것이 오류 처리에 좋은 방법입니다.

복구 불가능한 치명적 오류의 경우 "assert"이 오류 처리에 좋은 방법 일 수 있습니다. 예를 들어 esp_wifi_set_mode가 ESP_ERR_WIFI_NOT_INIT를 반환하면 Wi-Fi 드라이버가 esp_wifi_init에 의해 초기화되지 않은 것입니다. 응용 프로그램 개발 단계에서 이러한 종류의 오류를 매우 빠르게 감지 할 수 있습니다.

esp_err.h에서 ESP_ERROR_CHECK는 반환 값을 확인합니다. 다소 일반적인 오류 처리 코드이며 응용 프로그램 개발 단계에서 기본 오류 처리 코드로 사용할 수 있습니다. 그러나 API 사용자는 자체 오류 처리 코드를 작성하는 것이 좋습니다.

ESP32 Wi-Fi Scan

Currently, the esp_wifi_scan_start() API is supported only in Station or Station+AP mode.

Scan Type

Active Scan	Scan by sending a probe request. The default scan is an active scan.
Passive Scan	프로브 요청이 전송되지 않습니다. 특정 채널로 전환하고 비콘을 기다리십시오. 응용 프로그램은 wifi_scan_config_t의 scan_type 필드를 통해 활성화 할 수 있습니다.
Foreground Scan	이 스캔은 스테이션 모드에 Wi-Fi 연결이없는 경우에 적용됩니다. 포 그라운드 또는 백그라운드 검색은 Wi-Fi 드라이버로 제어되며 응용 프로그램에서 구성 할 수 없습니다.
Background Scan	이 스캔은 스테이션 모드 또는 스테이션 + AP 모드에 Wi-Fi 연결이있는 경우에 적용됩니다. 전경 스캔인지 배경 스캔인지는 Wi-Fi 드라이버에 따라 다르며 응용 프로그램에서 구성 할 수 없습니다.
All-Channel Scan	모든 채널을 스캔합니다. wifi_scan_config_t의 채널 필드가 0으로 설정되면 모든 채널 스캔입니다.
Specific Channel Scan	특정 채널 만 스캔합니다. wifi_scan_config_t의 채널 필드가 1로 설정되면 특정 채널 스캔입니다.

ESP32 Wi-Fi Scan

Currently, the esp_wifi_scan_start() API is supported only in Station or Station+AP mode.

combined arbitrarily

- All-Channel Background Active Scan
- All-Channel Background Passive Scan
- All-Channel Foreground Active Scan
- All-Channel Foreground Passive Scan
- Specific-Channel Background Active Scan
- Specific-Channel Background Passive Scan
- Specific-Channel Foreground Active Scan
- Specific-Channel Foreground Passive Scan

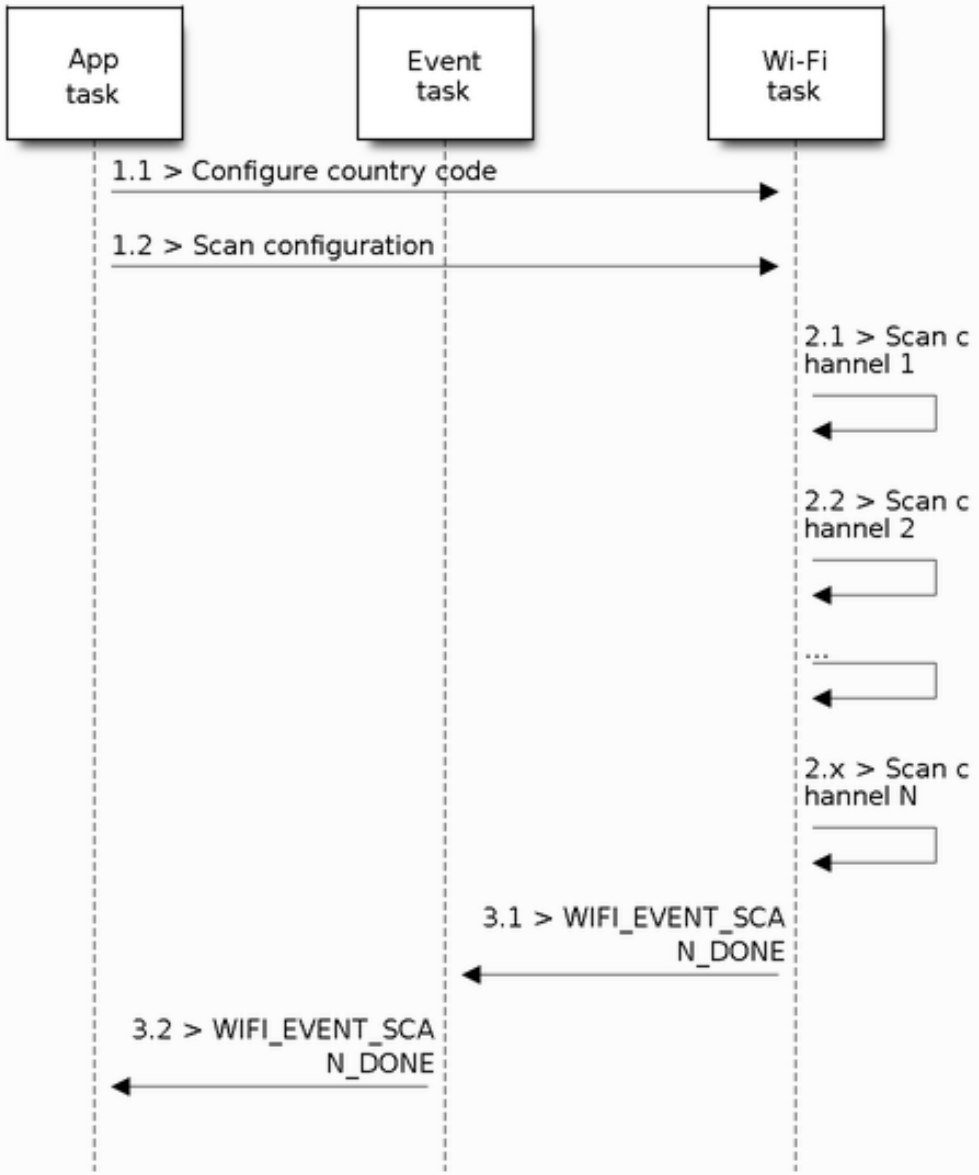
Scan Configuration

스캔 유형 및 기타 스캔 별 속성은 esp_wifi_scan_start에 의해 구성됩니다. 아래 표는 wifi_scan_config_t에 대한 자세한 설명을 제공합니다.

Field	Description
ssid	If the SSID is not NULL, it is only the AP with the same SSID that can be scanned.
bssid	If the BSSID is not NULL, it is only the AP with the same BSSID that can be scanned.
channel	If “channel” is 0, there will be an all-channel scan; **otherwise, there will be a specific-channel scan.**
show_hidden	show_hidden"이 0이면 스캔은 숨겨진 SSID를 가진 AP를 무시합니다. 그렇지 않으면 스캔은 숨겨진 AP를 일반 AP로 간주합니다.
scan_type	If “scan_type” is WIFI_SCAN_TYPE_ACTIVE, the scan is “**active**”; otherwise, it is a “passive” one.
scan_time	이 필드는 각 채널에서 스캔이 지속되는 시간을 제어하는 데 사용됩니다. 수동 검색의 경우 scan_time.passive는 각 채널의 드웰 시간을 지정합니다. 활성 스캔의 경우 각 채널의 체류 시간이 아래 표에 나열되어 있습니다. 여기서 min은 scan time.active.min의 약자이고 max는 scan_time.active.max의 약자입니다. • min = 0, max = 0 : 스캔이 각 채널에서 120ms 동안 유지됩니다. • min> 0, max = 0 : 스캔이 각 채널에서 120ms 동안 유지됩니다. • min = 0, max> 0 : 스캔이 각 채널에서 max ms 동안 유지됩니다. • min> 0, max> 0 : 각 채널에서 스캔이 머무르는 최소 시간은 min ms입니다. 이 시간 동안 AP를 찾지 못하면 스캔이 다음 채널로 전환됩니다. 그렇지 않으면 스캔이 채널에 max ms 동안 머무 릅니다. 스캔 성능을 향상 시키려면이 두 매개 변수를 수정하십시오.

Scan All APs In All Channels(foreground)

Scenario:



Foreground Scan of all Wi-Fi Channels

스캔 구성 단계
s1.1 : 기본 국가 정보가 원하는 정보가 아닌 경우 esp_wifi_set_country ()를 호출하여 국가 정보를 설정하십시오. <Wi-Fi 국가 코드>를 참조하십시오.

s1.2 : esp_wifi_scan_start ()를 호출하여 스캔을 구성하십시오. 이를 위해 <스캔 구성>을 참조하십시오. 이것은 전체 채널 스캔이므로 SSID / BSSID / 채널을 0으로 설정하면됩니다.

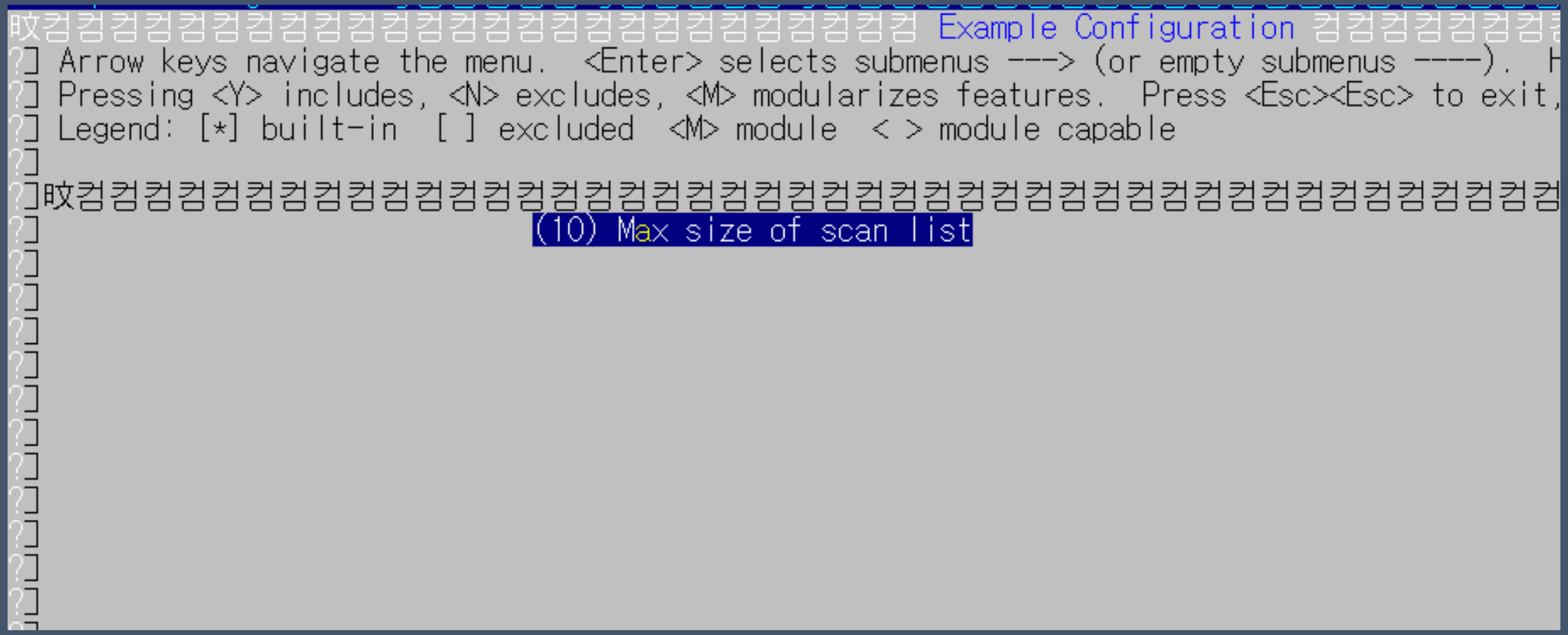
Wi-Fi 드라이버의 내부 스캔 단계
s2.1 : 스캔 유형이 WIFI_SCAN_TYPE_ACTIVE 인 경우 Wi-Fi 드라이버가 채널 1로 전환되고 프로브 요청을 브로드 캐스트합니다. 그렇지 않으면 Wi-Fi는 AP에서 비콘을 기다립니다. Wi-Fi 드라이버는 한동안 채널 1에 유지됩니다. 드웰 시간은 최소 / 최대 시간으로 구성되며 기본값은 120ms입니다.

s2.2 : Wi-Fi 드라이버가 채널 2로 전환되고 2.1 단계와 동일한 작업을 수행합니다.

s2.3 : Wi-Fi 드라이버는 마지막 채널 N을 스캔합니다. 여기서 N은 1.1 단계에서 구성된 국가 코드에 따라 결정됩니다.

스캔 완료 이벤트 처리 단계
s3.1 : 모든 채널이 스캔되면 <WIFI_EVENT_SCAN_DONE>이 발생합니다.

s3.2 : 응용 프로그램의 이벤트 콜백 기능은 응용 프로그램 작업에 <WIFI_EVENT_SCAN_DONE>이 수신되었음을 알립니다. esp_wifi_scan_get_ap_num ()은이 스캔에서 발견 된 AP의 수를 얻기 위해 호출됩니다. 그런 다음 충분한 항목을 할당하고 esp_wifi_scan_get_ap_records ()를 호출하여 AP 레코드를 가져옵니다. esp_wifi_scan_get_ap_records ()가 호출되면 Wi-Fi 드라이버의 AP 레코드가 해제됩니다. 단일 스캔 완료 이벤트에 대해 esp_wifi_scan_get_ap_records ()를 두 번 호출하지 마십시오. 스캔 완료 이벤트가 발생할 때 esp_wifi_scan_get_ap_records ()가 호출되지 않으면 Wi-Fi 드라이버가 할당 한 AP 레코드가 해제되지 않습니다. 따라서 esp_wifi_scan_get_ap_records ()를 한 번만 호출해야합니다.



실습 : scan
동작시키고 AP 목록 획득

실습 : tcp_client
에코서버에 접속해서 5초마다 1회씩 패킷 송신 및 수신

감사합니다.