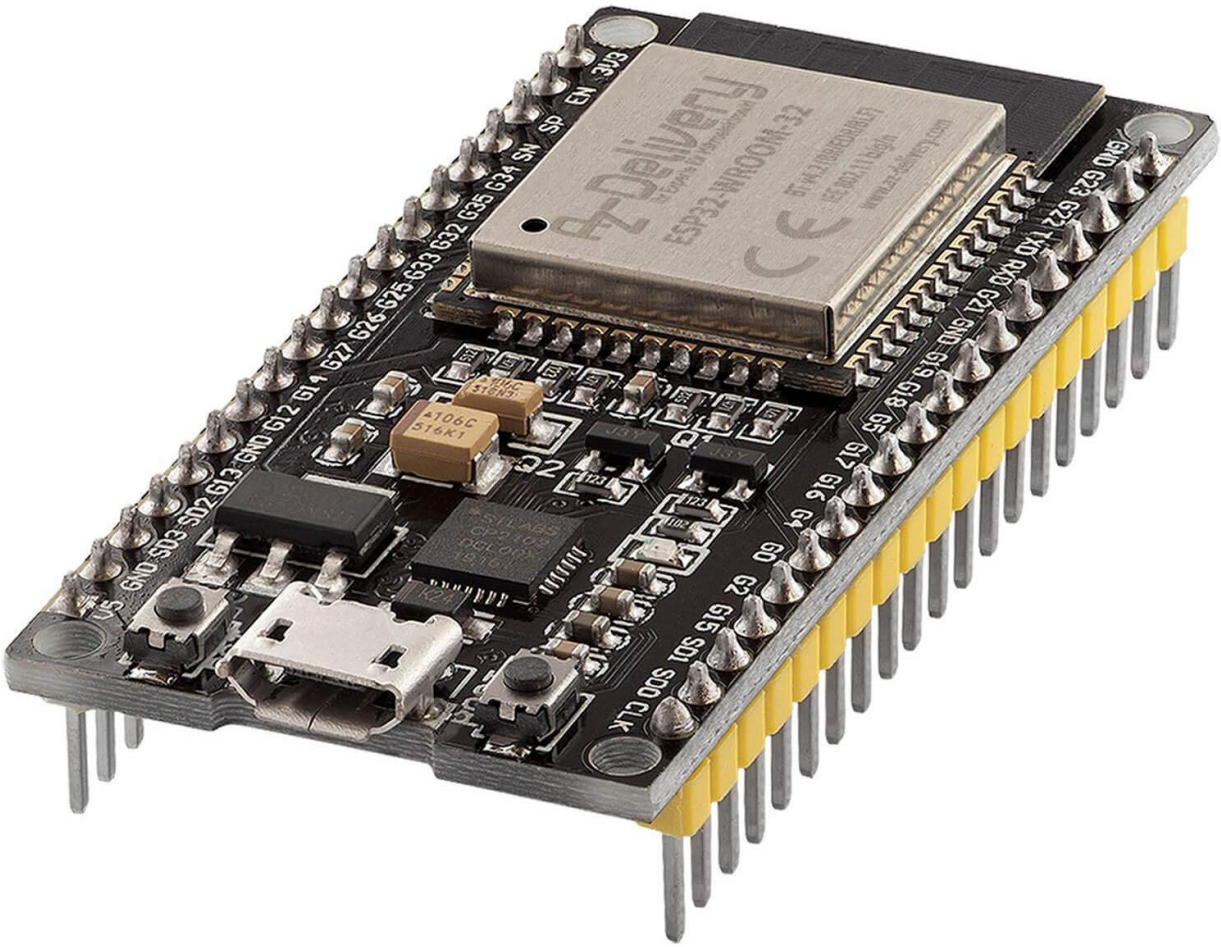
How to Set Up an ESP32 WiFi Access Point Using ESP-IDF and FreeRTOS



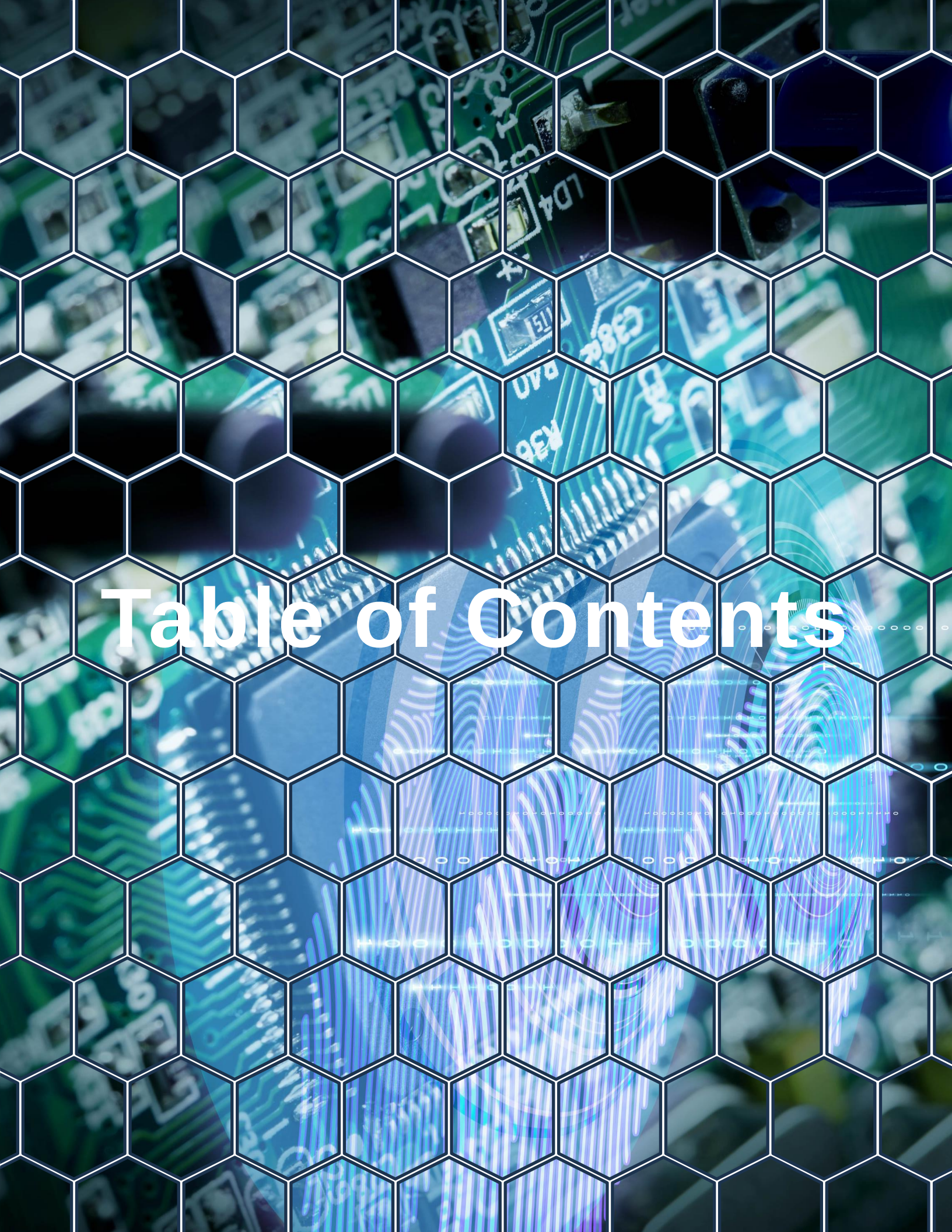
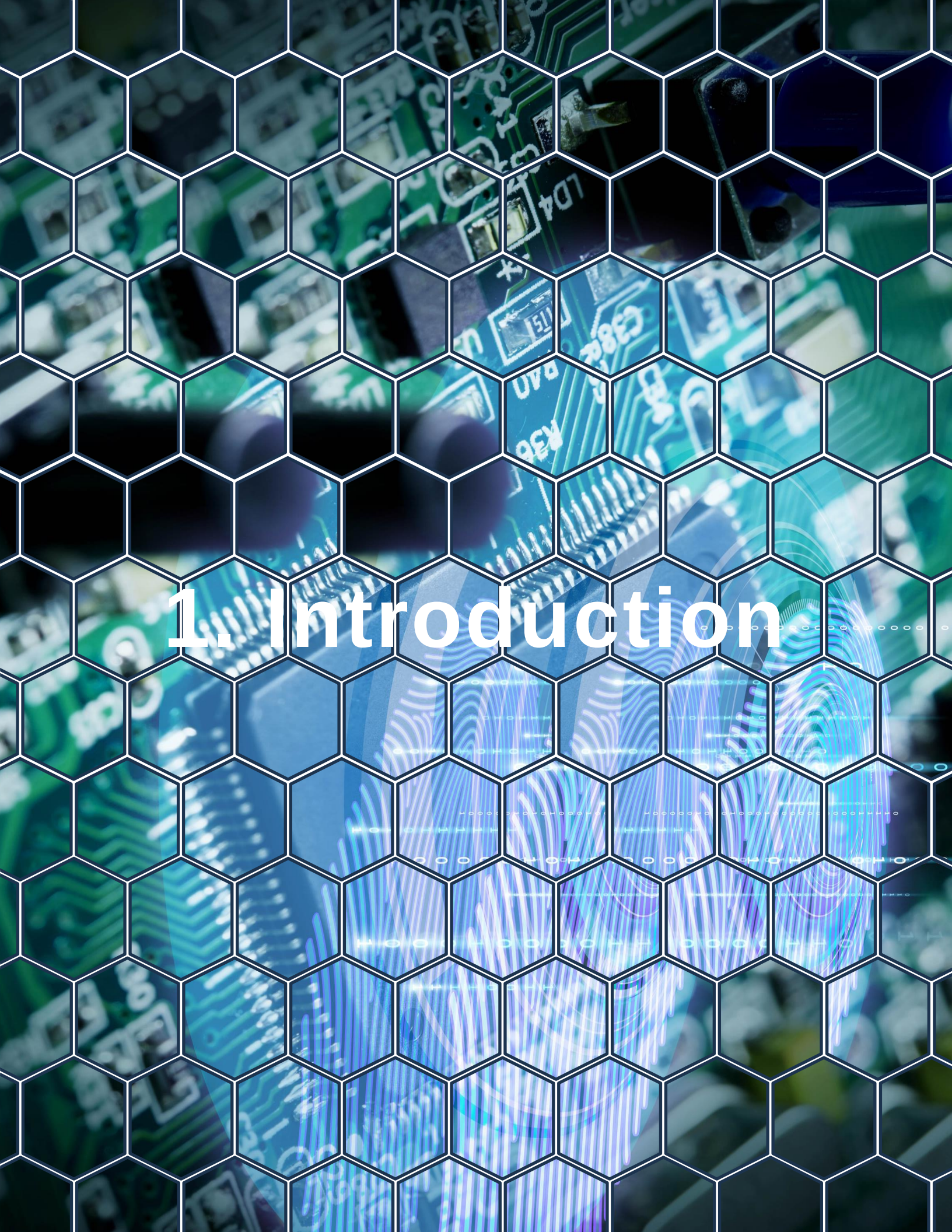


Table of Contents

Table of Contents

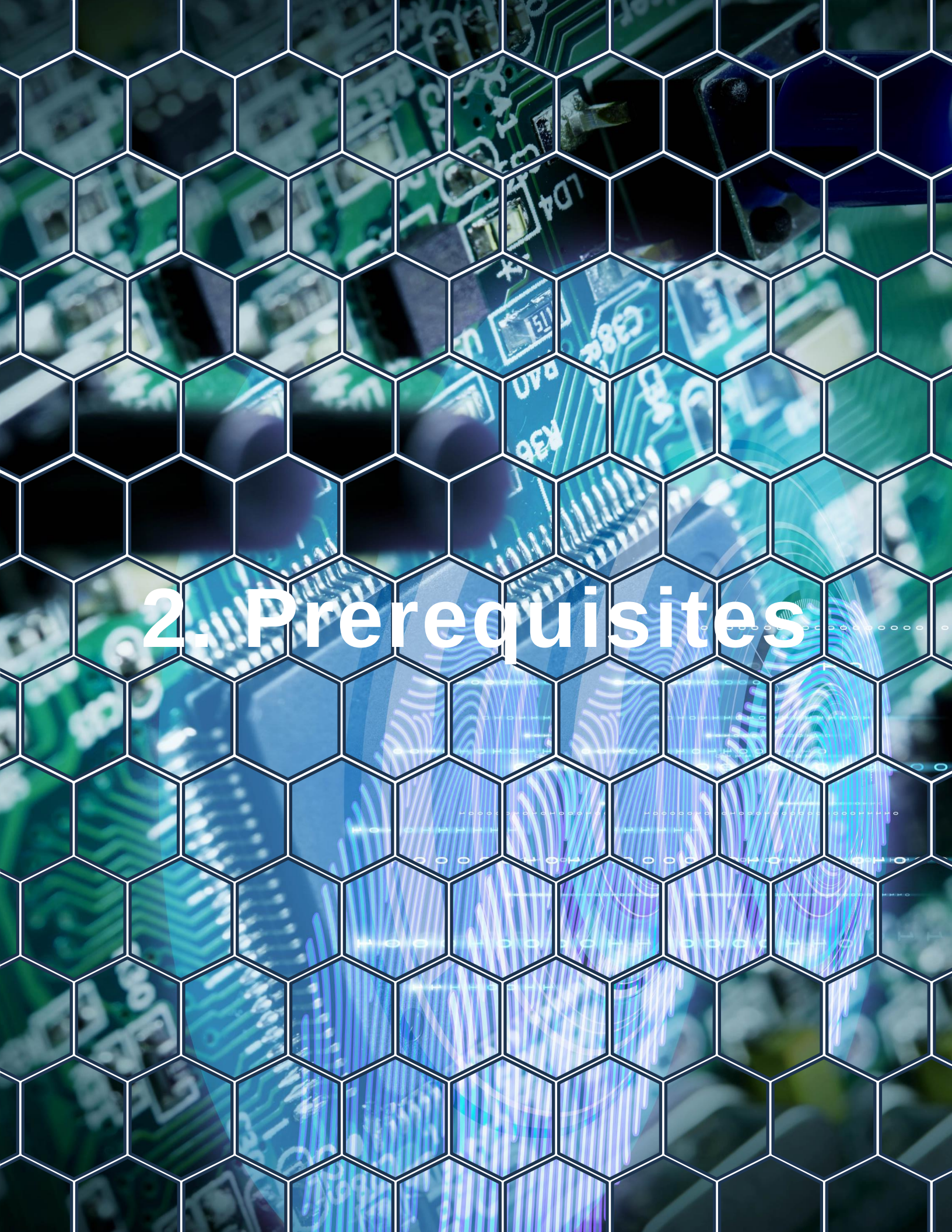


1. Introduction

1. Introduction

In IoT development, configuring your ESP32 as a WiFi Access Point (AP) is essential for creating closed and secure local networks without relying on external routers. This is particularly useful for edge devices, configuration interfaces, or sensor networks.

In this article, we will walk through how to configure your ESP32 as a WiFi AP using the **ESP-IDF framework** and **FreeRTOS**. We'll also explore advanced settings like assigning a custom IP address, adjusting transmission power, and hiding your network for added security.



2. Prerequisites

2. Prerequisites

Before diving in, ensure the following:

- ESP32 development board
- ESP-IDF framework properly installed and set up
- Familiarity with FreeRTOS and basic ESP-IDF APIs
- Serial monitor (e.g., idf.py monitor or minicom) for logging

3. How to Set Up a WiFi Access Point

3. How to Set Up a WiFi Access Point

Here's a simple example to set up the ESP32 as a WiFi Access Point using ESP-IDF:

```c
#include "esp_wifi.h"
#include "esp_event.h"
#include "esp_log.h"
#include "nvs_flash.h"
#include "freertos/FreeRTOS.h"
#include "freertos/task.h"

#define WIFI_SSID "MyESP32_AP"
#define WIFI_PASS "password123"
#define MAX_STA_CONN 4

static const char *TAG = "wifi_ap";

void wifi_init_softap(void) {
    // Initialize TCP/IP and default network interfaces
    ESP_ERROR_CHECK(esp_netif_init());
    ESP_ERROR_CHECK(esp_event_loop_create_default());
    esp_netif_create_default_wifi_ap();

    // Initialize Wi-Fi driver with default config
    wifi_init_config_t cfg = WIFI_INIT_CONFIG_DEFAULT();
    ESP_ERROR_CHECK(esp_wifi_init(&cfg));

    // Configure Access Point credentials and parameters
    wifi_config_t wifi_config = {
        .ap = {
```

3. How to Set Up a WiFi Access Point

```
    // Configure Access Point credentials and parameters
    wifi_config_t wifi_config = {
        .ap = {
            .ssid = WIFI_SSID,
            .ssid_len = strlen(WIFI_SSID),
            .password = WIFI_PASS,
            .max_connection = MAX_STA_CONN,
            .authmode = WIFI_AUTH_WPA_WPA2_PSK
        },
    };

    // WPA/WPA2 mode requires password; else use open
    if (strlen(WIFI_PASS) == 0) {
        wifi_config.ap.authmode = WIFI_AUTH_OPEN;
    }

    // Set Wi-Fi mode to Access Point
    ESP_ERROR_CHECK(esp_wifi_set_mode(WIFI_MODE_AP));

    // Apply the configuration
    ESP_ERROR_CHECK(esp_wifi_set_config(WIFI_IF_AP, &wifi_config));

    // Start the Wi-Fi
    ESP_ERROR_CHECK(esp_wifi_start());

    ESP_LOGI(TAG, "WiFi Access Point started. SSID: %s", WIFI_SSID);
}
```

4. Configure WiFi Access Point with a Custom IP, Gateway, and Netmask

4. Configure WiFi Access Point with a Custom IP, Gateway, and Netmask

To manually configure the IP settings, modify the default `esp_netif_dhcps_stop()` behavior:

```c
#include "esp_netif_ip_addr.h"
#include "esp_netif.h"

void set_custom_ip(esp_netif_t *netif) {
    esp_netif_dhcps_stop(netif);

    esp_netif_ip_info_t ip_info;
    IP4_ADDR(&ip_info.ip, 192, 168, 10, 1);
    IP4_ADDR(&ip_info.gw, 192, 168, 10, 1);
    IP4_ADDR(&ip_info.netmask, 255, 255, 255, 0);
    ESP_ERROR_CHECK(esp_netif_set_ip_info(netif, &ip_info));

    esp_netif_dhcps_start(netif);
}
```

Call this function right after creating the default AP interface:

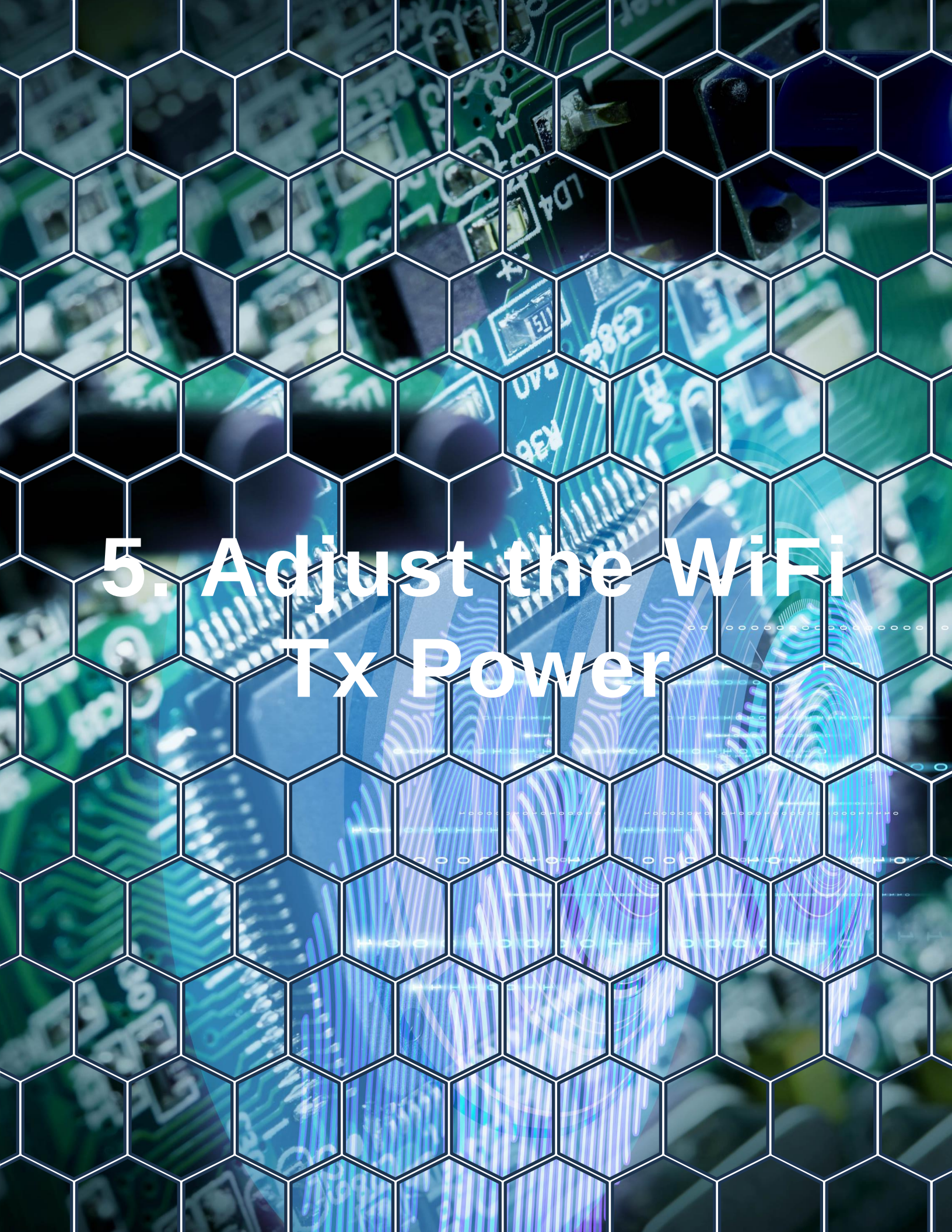
```
```

4. Configure WiFi Access Point with a Custom IP, Gateway, and Netmask

Call this function right after creating the default

AP interface:

```
esp_netif_t *ap_netif = esp_netif_create_default_wifi_ap();
set_custom_ip(ap_netif);
```



5. Adjust the WiFi Tx Power

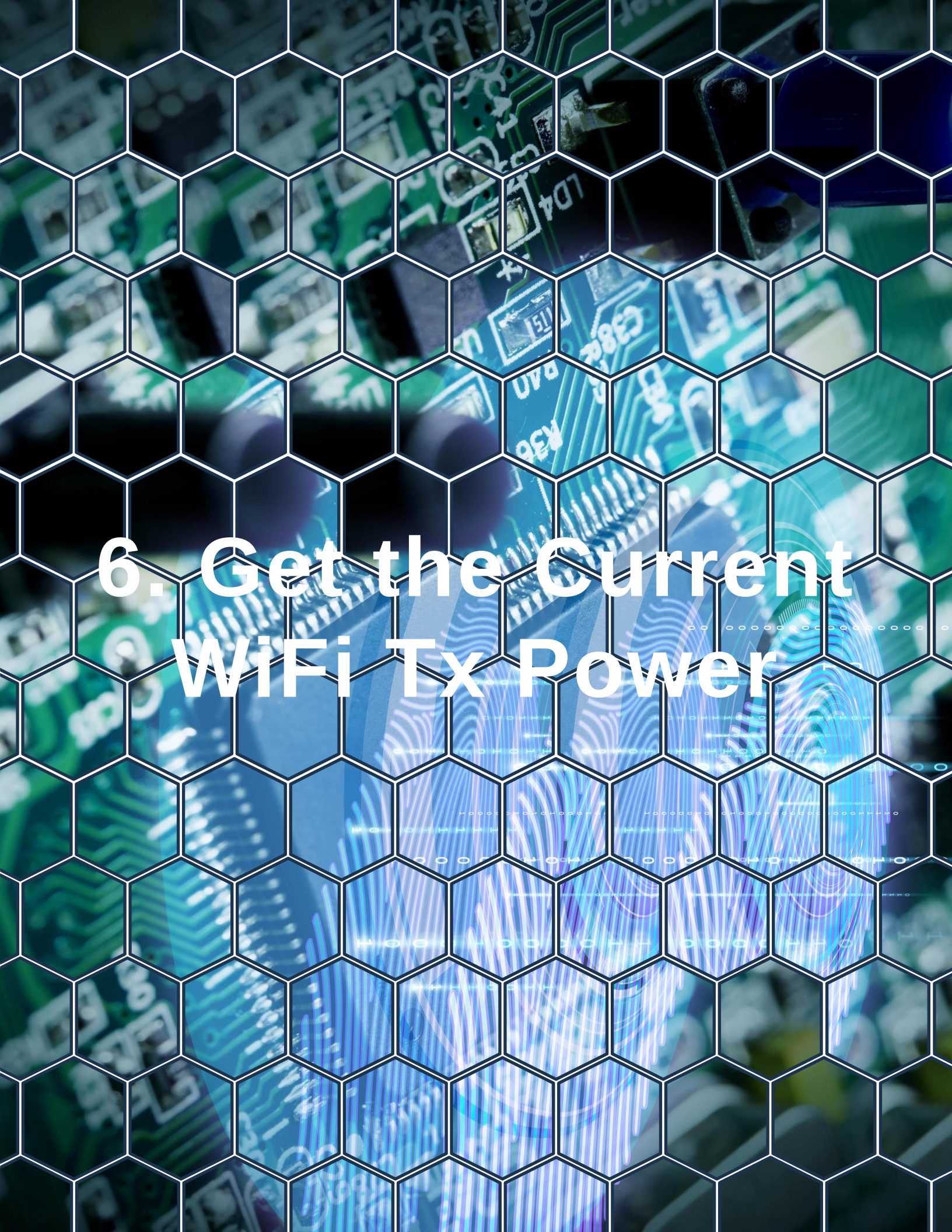
5. Adjust the WiFi Tx Power

To optimize range and power consumption, use `esp_wifi_set_max_tx_power()`. The value is in units of 0.25 dBm (e.g., 78 = 19.5 dBm):

```
esp_err_t set_wifi_tx_power(int8_t dbm) {
    if (dbm < 8 || dbm > 20) {
        ESP_LOGW(TAG, "Invalid Tx power. Range: 8 - 20 dBm");
        return ESP_ERR_INVALID_ARG;
    }

    int8_t power_quarter_dbm = dbm * 4;
    return esp_wifi_set_max_tx_power(power_quarter_dbm);
}
```

Use it like this:

```
set_wifi_tx_power(15); // Sets Tx power to 15 dBm
```



6. Get the Current WiFi Tx Power

6. Get the Current WiFi Tx Power

To retrieve the current power setting:

```
void print_current_tx_power(void) {
    int8_t tx_power = 0;
    esp_wifi_get_max_tx_power(&tx_power);
    ESP_LOGI(TAG, "Current Tx Power: %.2f dBm", tx_power / 4.0);
}
```

This is useful to verify adjustments or perform diagnostics.

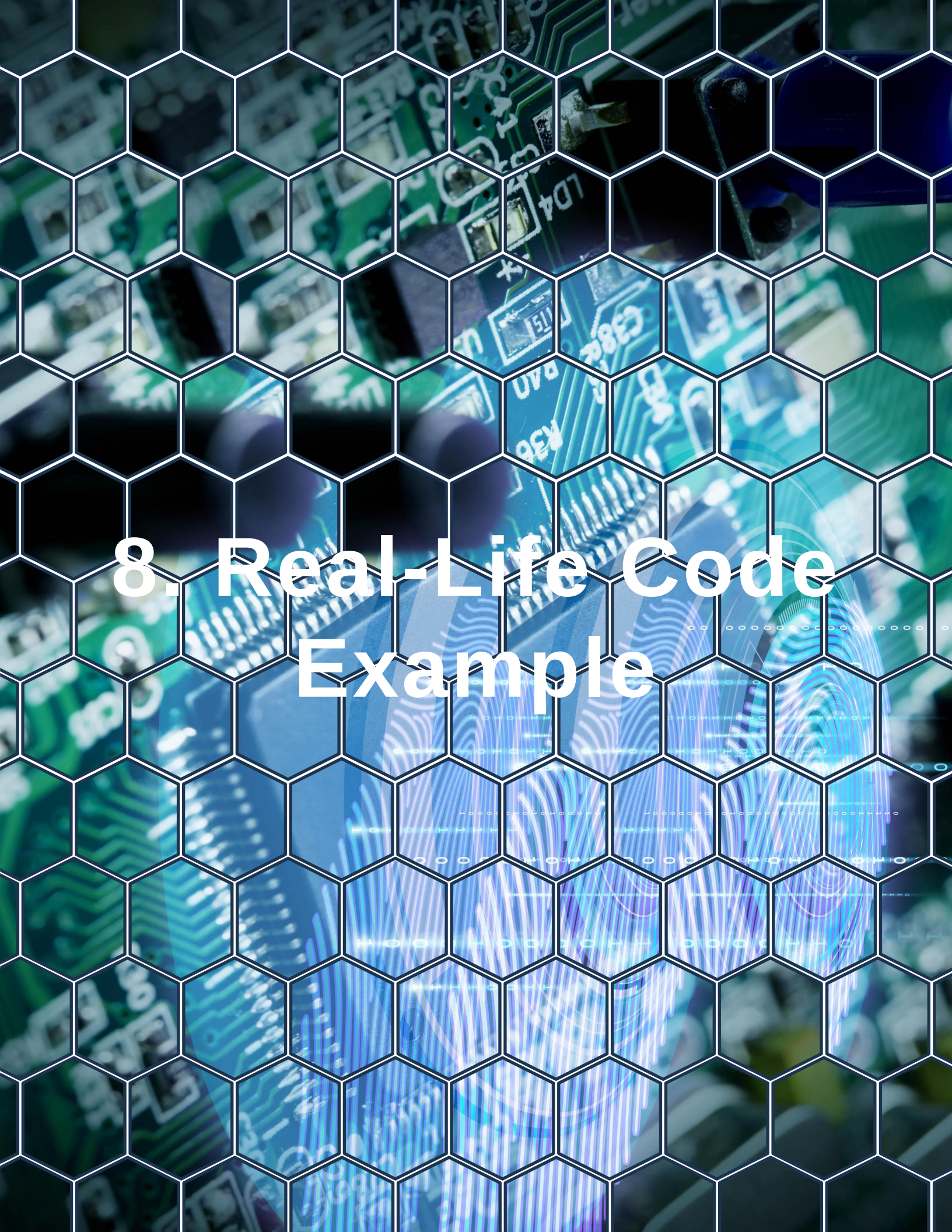
7. Create a Hidden WiFi Access Point

7. Create a Hidden WiFi Access Point

To hide the SSID (making the network invisible to casual scans):

```
wifi_config_t wifi_config = {
    .ap = {
        .ssid = WIFI_SSID,
        .ssid_len = strlen(WIFI_SSID),
        .password = WIFI_PASS,
        .max_connection = MAX_STA_CONN,
        .authmode = WIFI_AUTH_WPA_WPA2_PSK,
        .ssid_hidden = 1,  // This hides the SSID
    },
};
```

💡 Note: While hiding the SSID adds a layer of obscurity, it is not a substitute for WPA2 encryption.



8. Real-Life Code Example

8. Real-Life Code Example

This code sets up a robust Wi-Fi Access Point on the ESP32 and captures key connection events such as device joins, IP assignment, and disconnects. Logging MAC and IP addresses is useful for device tracking, logging, and network management in local IoT systems.

```c
#include <stdio.h>
#include <string.h>
#include "freertos/FreeRTOS.h"
#include "freertos/task.h"
#include "esp_wifi.h"
#include "esp_event.h"
#include "esp_log.h"
#include "nvs_flash.h"
#include "esp_mac.h"
#include "esp_netif.h"
#include "esp_system.h"
#include "esp_event.h"
#include "esp_err.h"

// Define the Access Point credentials
#define SSID        "ESP32_AccessPoint"
#define PASSWORD    "12345678"

static const char *TAG = "wifi_ap";

/**
 * @brief Callback function to handle Wi-Fi and IP events in Access Point mode.
```

8. Real-Life Code Example

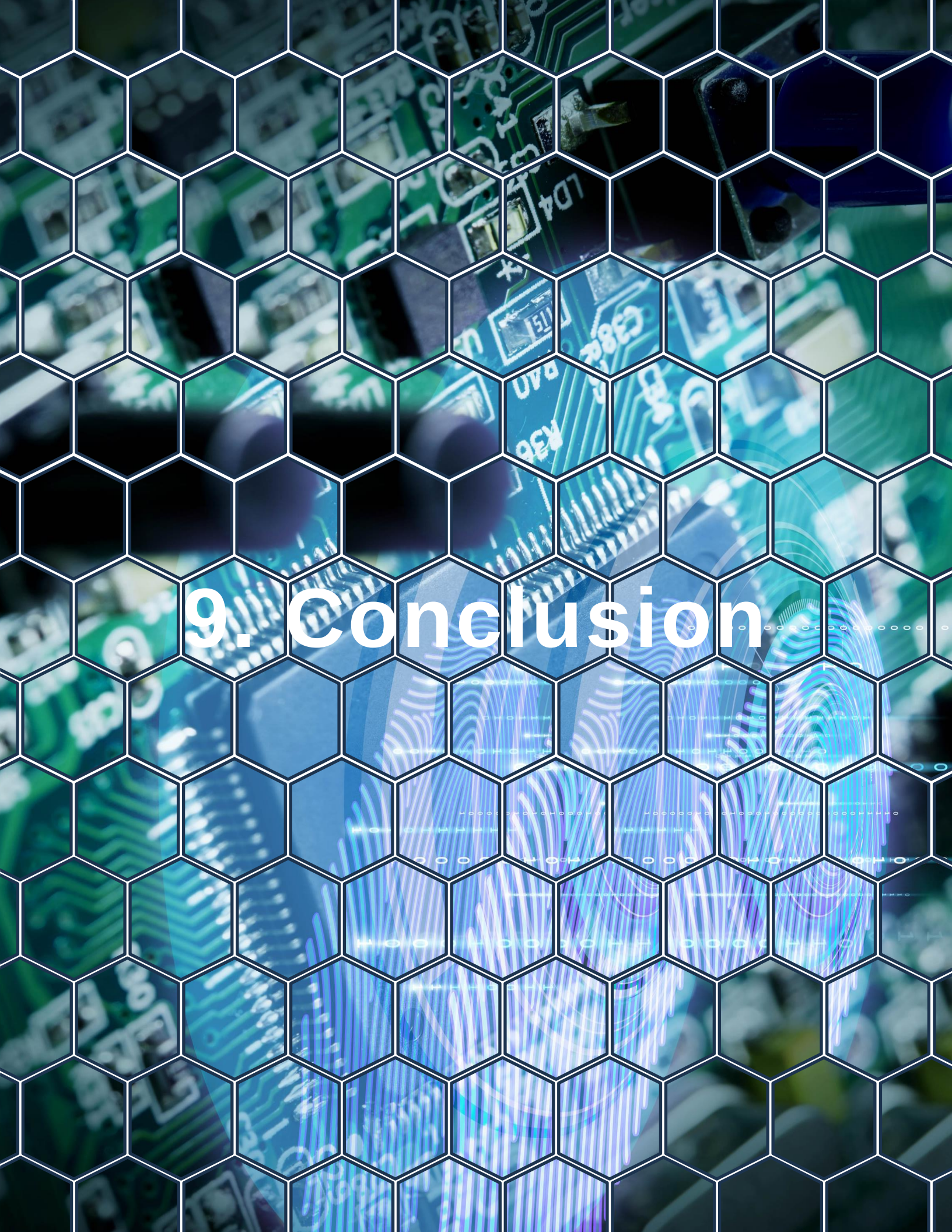
```c
/**
 * @brief Callback function to handle Wi-Fi and IP events in Access Point mode.
 *
 * This function is registered to handle system events related to the Wi-Fi and IP stack
 * when the ESP32 is configured as a Wi-Fi Access Point. It logs key events such as:
 *  - AP start and stop
 *  - Station (client) connection and disconnection
 *  - IP address assignment to connected stations via DHCP
 *
 * @param arg         User-defined argument passed during registration
 *                    (not used here, set to NULL).
 * @param event_base  Event base identifier, e.g., WIFI_EVENT or IP_EVENT.
 * @param event_id    Specific event ID, such as WIFI_EVENT_AP_START or
 *                    IP_EVENT_AP_STAIPASSIGNED.
 * @param event_data  Pointer to event-specific data structure depending on
 *                    the event type.
 *
 * @return void
 */

static void wifi_event_handler(void *arg, esp_event_base_t event_base,
                               int32_t event_id, void *event_data) {
    if (event_base == WIFI_EVENT) {
        switch (event_id) {
            case WIFI_EVENT_AP_START:
                ESP_LOGI(TAG, "Access Point started.");
                break;

            case WIFI_EVENT_AP_STOP:
                ESP_LOGI(TAG, "Access Point stopped.");
                break;

            case WIFI_EVENT_AP_STACONNECTED: {
                wifi_event_ap_staconnected_t *conn = (wifi_event_ap_staconnected_t *)event_data;
                ESP_LOGI(TAG, "Device connected: MAC="MACSTR", AID=%ld",
                    MAC2STR(conn->mac), (long)conn->aid);
                break;
            }

            case WIFI_EVENT_AP_STADISCONNECTED: {
                wifi_event_ap_stadisconnected_t *disc = (wifi_event_ap_stadisconnected_t *)event_data;
                ESP_LOGI(TAG, "Device disconnected: MAC="MACSTR", AID=%ld",
                    MAC2STR(disc->mac), (long)disc->aid);
                break;
            }

            default:
                ESP_LOGW(TAG, "Unhandled WiFi event: %ld", (long)event_id);
        }
```

8. Real-Life Code Example

```
            default:
                ESP_LOGW(TAG, "Unhandled WiFi event: %ld", (long)event_id);
        }

    } else if (event_base == IP_EVENT && event_id == IP_EVENT_AP_STAIPASSIGNED) {
        ip_event_ap_staipassigned_t *ip = (ip_event_ap_staipassigned_t *)event_data;
        ESP_LOGI(TAG, "IP assigned to station: " IPSTR, IP2STR(&ip->ip));
    }
}

/**
 * @brief Initializes the Wi-Fi stack and configures ESP32 as an Access Point.
 *
 * This function:
 * - Initializes NVS
 * - Sets up default Wi-Fi configuration
 * - Registers Wi-Fi and IP events
 * - Starts Wi-Fi in AP mode
 */
void wifi_init_softap(void) {
    // Initialize TCP/IP and default network interfaces
    esp_netif_init();
    esp_event_loop_create_default();
    esp_netif_create_default_wifi_ap();

    // Initialize Wi-Fi driver with default config
    wifi_init_config_t cfg = WIFI_INIT_CONFIG_DEFAULT();
    ESP_ERROR_CHECK(esp_wifi_init(&cfg));

    // Register event handlers for Wi-Fi and IP events
    ESP_ERROR_CHECK(esp_event_handler_instance_register(WIFI_EVENT,
                                                        ESP_EVENT_ANY_ID,
                                                        &wifi_event_handler,
                                                        NULL,
                                                        NULL));
    ESP_ERROR_CHECK(esp_event_handler_instance_register(IP_EVENT,
                                                        IP_EVENT_AP_STAIPASSIGNED,
                                                        &wifi_event_handler,
                                                        NULL,
                                                        NULL));

    // Set Wi-Fi mode to Access Point
    ESP_ERROR_CHECK(esp_wifi_set_mode(WIFI_MODE_AP));

    // Configure Access Point credentials and parameters
    wifi_config_t ap_config = {
        .ap = {
            .ssid = SSID,                       // SSID
            .ssid_len = 0,
            .channel = 1,                       // Wi-Fi channel
```

8. Real-Life Code Example

```
    // Configure Access Point credentials and parameters
    wifi_config_t ap_config = {
        .ap = {
            .ssid = SSID,                       // SSID
            .ssid_len = 0,
            .channel = 1,                       // Wi-Fi channel
            .password = PASSWORD,               // Password (min 8 chars)
            .max_connection = 4,                // Max number of stations
            .authmode = WIFI_AUTH_WPA_WPA2_PSK  // Authentication
        },
    };

    // WPA/WPA2 mode requires password; else use open
    if (strlen((char *)ap_config.ap.password) == 0) {
        ap_config.ap.authmode = WIFI_AUTH_OPEN;
    }

    // Apply the configuration
    ESP_ERROR_CHECK(esp_wifi_set_config(WIFI_IF_AP, &ap_config));

    // Start the Wi-Fi
    ESP_ERROR_CHECK(esp_wifi_start());

    ESP_LOGI(TAG, "Wi-Fi Access Point initialized. SSID:%s password:%s",
             ap_config.ap.ssid, ap_config.ap.password);
}

/**
 * @brief Main application entry point.
 *
 * Initializes NVS and starts Wi-Fi as AP.
 */
void app_main(void) {
    // Initialize NVS (required before Wi-Fi)
    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);

    // Start Wi-Fi in AP mode
    wifi_init_softap();
}
```



9. Conclusion

9. Conclusion

The ESP32's WiFi AP functionality, when combined with FreeRTOS and ESP-IDF, makes it an excellent tool for setting up reliable, configurable, and secure local networks. Whether you're setting up a provisioning interface, a sensor gateway, or a standalone network, understanding how to fine-tune AP settings — including IP configuration, transmission power, and SSID visibility — can significantly improve performance and security.

With minimal code, you can go from a basic AP to a fully customized and optimized local

9. Conclusion

With minimal code, you can go from a basic AP to a fully customized and optimized local network, giving your IoT project the robust wireless connectivity it needs.