



Tutorial y ejemplos de bibliotecas de ESP32 WiFi (IDE de Arduino)

Por Khaled Magdy

Esta es la guía definitiva para ESP32 WiFi. Si estás empezando con ESP32 WiFi, este es el tutorial guía definitivo que debes seguir. Aprenderás todo sobre ESP32 WiFi con ejemplos paso a paso (en Arduino Core). Los temas están desglosados, categorizados y organizados de una manera tan clara que te garantiza un progreso sistemático constante hacia el aprendizaje y el dominio de las aplicaciones basadas en ESP32 WiFi y el desarrollo de IoT.

El objetivo final de este tutorial es ayudar a los principiantes con ESP32 WiFi a saber todo sobre este enorme tema y trazar una hoja de ruta clara sobre lo que hay que aprender de forma sistemática. Uno puede sentirse fácilmente abrumado al empezar con ESP32 WiFi por la cantidad de temas y bibliotecas que existen (por ejemplo, HTTP, TCP/IP, UDP, WebServers, AsyncWebServers, WebSocket, MQTT, mDNS, MESH, ESP-NOW, etc.). Hay demasiadas bibliotecas, modos, topologías y protocolos que hacen que empezar sea una tarea aún más difícil de lo que debería ser.

Como principiante, es bastante fácil sentirse abrumado por la enorme cantidad de artículos y tutoriales que tratan sobre el tema WiFi ESP32. Y la pregunta más razonable es ¿por dónde debería empezar? ¡Este artículo está aquí para responder a esta misma pregunta!

Tabla de contenido

1. ESP32 Wi-Fi
2. Introducción a ESP32 WiFi
3. Modos WiFi del ESP32
4. Redes WiFi ESP32
5. Proyectos de ejemplo de WiFi ESP32

- 8. Solución de problemas de WiFi del ESP32
- 9. Observaciones finales
- 10. Excelentes recursos sobre WiFi ESP32

ESP32 Wi-Fi

El ESP32 es un potente microcontrolador de doble núcleo con hardware RF integrado que admite Bluetooth, BLE y WiFi. Hemos analizado las funcionalidades Bluetooth del ESP32 en un tutorial anterior, pero en este tutorial nos interesa analizar en profundidad las capacidades WiFi del ESP32, que incluyen, entre otras, las siguientes:

- ESP32 WiFi implementa TCP/IP y el protocolo MAC WiFi 802.11 b/g/n completo
- Velocidad de datos hasta 150 Mbps
- Potencia de transmisión de señal ajustable
- Hasta 20,5 dBm de potencia de transmisión
- Diversidad de antena: hay un interruptor de RF externo para seleccionar la mejor antena para minimizar los efectos del desvanecimiento del canal.
- MAC WiFi ESP32 que aplica funciones de protocolo de bajo nivel automáticamente



Vamos a utilizar la biblioteca Arduino ESP32 WiFi.h para aprovechar las capacidades WiFi del hardware del microcontrolador ESP32. También necesitamos saber más sobre los modos de funcionamiento WiFi y algunos conceptos básicos de redes. Eso es lo que cubriremos en las siguientes secciones.

Introducción a ESP32 WiFi

Ahora comenzemos con ESP32 WiFi y preparemos la configuración necesaria para experimentar con ESP32 WiFi y crear algunas aplicaciones para probar todo lo que vamos a aprender en cada sección.

Requisitos para empezar con ESP32 WiFi

Requisitos de hardware

Solo necesitas una placa de desarrollo ESP32 como mínimo. Sin embargo, es muy recomendable que consultes la [lista completa de hardware](#) para esta serie de tutoriales si estás interesado en seguir todos los ejemplos y proyectos de [tutoriales ESP32](#) en DeepBlueMbedded.

Requisitos de SW

Antes de continuar con este tutorial, debe haber instalado el núcleo Arduino ESP32 en su IDE de Arduino para poder compilar y crear proyectos para ESP32 en el IDE de Arduino. Siga el tutorial a continuación para comenzar si aún no lo ha hecho.

- Instalación de ESP32 en Arduino IDE

Si recién estás comenzando con ESP32, es muy recomendable comenzar con el siguiente tutorial para comenzar tu viaje con los microcontroladores ESP32. Luego, ve al segundo enlace que te dirige a la página principal de la serie de tutoriales de ESP32, donde encontrarás todos los tutoriales de ESP32 ordenados y categorizados de una manera lógica que te garantiza un progreso sistemático en el aprendizaje de la programación ESP32 y la IoT.

- Introducción a ESP32
- Serie de tutoriales ESP32 (página principal)

Biblioteca WiFi ESP32 (IDE de Arduino)

en Wi-Fi. Sin embargo, se puede acceder a funciones adicionales de WiFi ESP32 y protocolos específicos a través de bibliotecas externas que necesitaremos instalar a medida que avancemos con los proyectos de ejemplo.

Dado que la biblioteca WiFi ESP32 ya está en Arduino Core, el siguiente paso será incluirla en tu código antes de poder usar las funcionalidades WiFi con ESP32. A continuación te indicamos cómo hacerlo:

```
#include <WiFi.h>
```

Modos WiFi del ESP32

El ESP32 WiFi puede funcionar en uno de los siguientes modos: estación WiFi, punto de acceso o ambos al mismo tiempo. Hay un modo adicional llamado modo promiscuo en el que el ESP32 actuará como un rastreador de WiFi. La biblioteca ESP32 admite los primeros 3 modos de forma predeterminada. Para configurar el modo WiFi del ESP32, puede usar la función `WiFi . mode ()` que toma un argumento como entrada (el modo deseado).

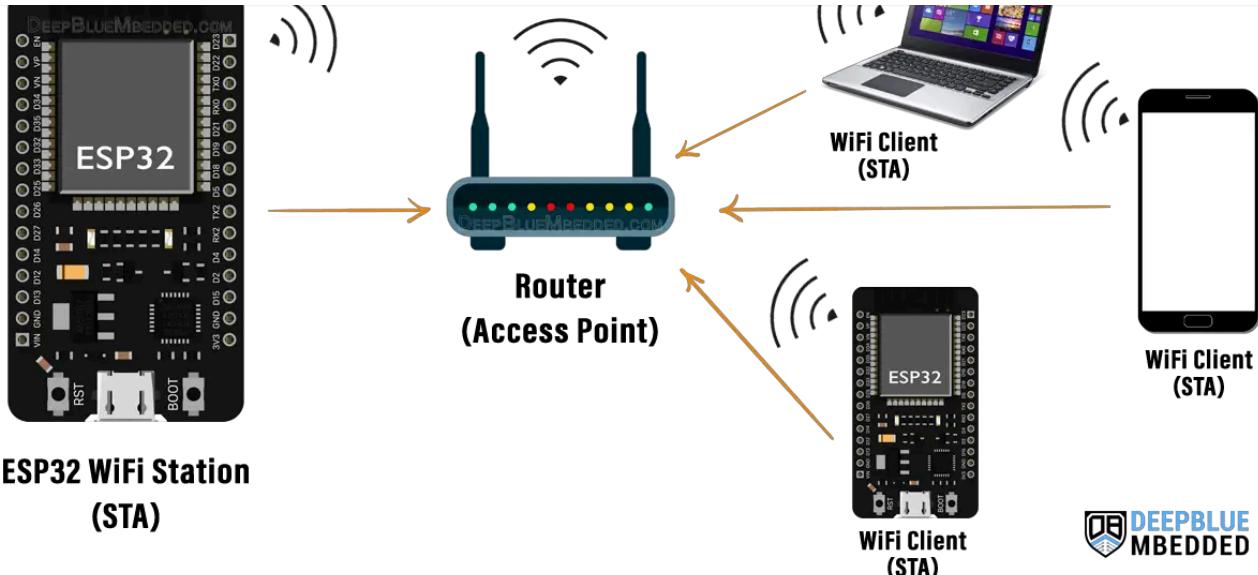
Modo estación (STA) : el ESP32 se conecta a otras redes existentes (como su enrutador en casa o cualquier otro punto de acceso)	<code>WiFi.mode (WIFI_STA) ;</code>
Modo de punto de acceso (AP) : el ESP32 crea su propia red actuando como un punto de acceso al que otras estaciones WiFi pueden conectarse.	<code>WiFi . mode (WIFI_AP) ;</code>
Modo Punto de acceso + Estación (AP_STA) : en este modo, el ESP32 WiFi actuará como Punto de acceso y Estación WiFi simultáneamente	<code>WiFi.mode (WIFI_AP_STA) ;</code>

A continuación, profundizaremos en estos modos y cómo funcionan.

Modo de estación WiFi

Cuando configura el ESP32 en modo de estación WiFi, puede conectarse a cualquier otra red WiFi (como el enrutador de su hogar). En este modo, el ESP32 puede enviar y recibir datos hacia y desde otros dispositivos conectados a la misma red utilizando su dirección IP única.

Por ejemplo, puede conectar un ESP32 a su red Wi-Fi doméstica y usarlo para controlar dispositivos domésticos inteligentes, como luces, termostatos y cerraduras de puertas.



La red aquí la establece el enrutador (AP) al que están conectados los dispositivos. El ESP32 se conecta a la red WiFi de su enrutador y se le asigna una dirección IP única que puede usarse para comunicarse con otros dispositivos en la red.

Configurar el ESP32 en modo estación (STA)

Para configurar el ESP32 para que funcione en modo estación, es necesario llamar a la función `WiFi.mode(WIFI_STA)`, que toma WiFi_Mode como parámetro.

```
WiFi.mode(WIFI_STA);
```

También es necesario definir las credenciales de la red WiFi a la que se quiere conectar el ESP32 (por ejemplo, el router de casa). Este paso se realiza de la siguiente manera.

```
// Replace with your own network credentials
const char* ssid = "yourNetworkSSID";
const char* password = "yourNetworkPassword";
```

Ahora, ya está listo para comenzar a conectar su ESP32 a la red WiFi en modo estación llamando a la función `WiFi.begin(ssid, password)`, que toma las credenciales de red predefinidas como argumentos.

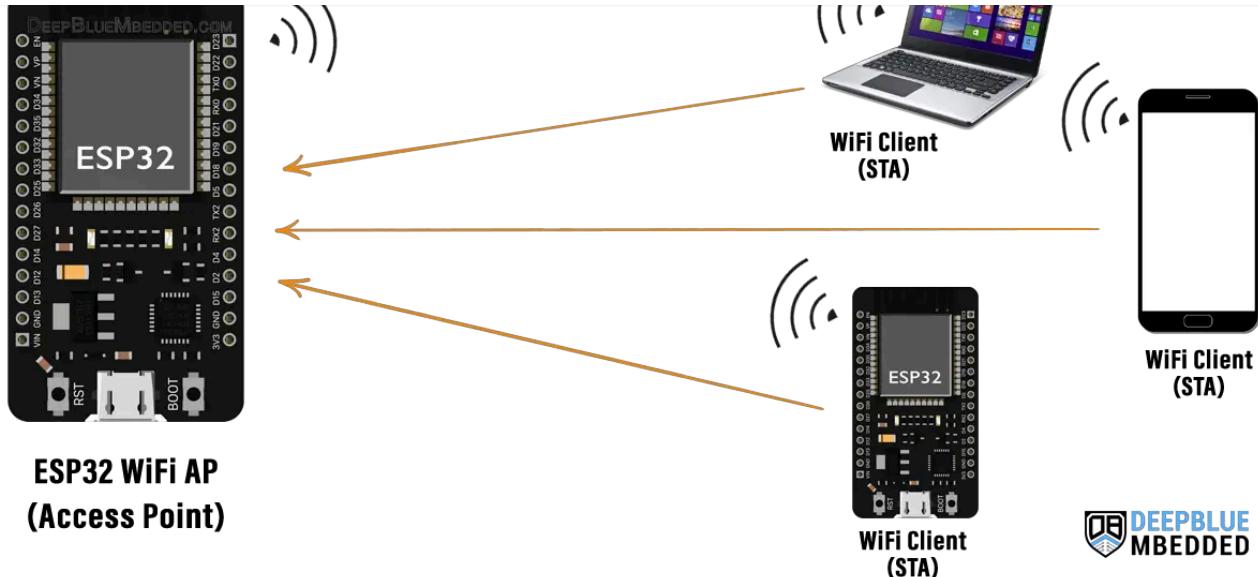
```
WiFi.begin(ssid, password);
```

Modo de punto de acceso (AP) (modo SoftAP)

Cuando configura el ESP32 en modo Punto de acceso, le permite crear una red Wi-Fi. Esto significa que tendrá su propia red Wi-Fi y no necesitará un enrutador externo para conectar sus dispositivos. Otros dispositivos (estaciones) pueden conectarse fácilmente a la red del ESP32 y comunicarse con él.

Por ejemplo, puede utilizar ESP32 en modo de punto de acceso para crear una red local a la que otros ESP32 (estaciones), su computadora o su teléfono inteligente puedan conectarse e intercambiar datos sin necesidad de tener un enrutador.

La ausencia de un enrutador en esta red significa que solo es una red local y sus dispositivos no pueden acceder a la web de ningún modo.



El modo de punto de acceso (AP) también se conoce como SoftAP (Soft Access Point) porque le permite crear su propia red sin un enrutador ni ningún punto de acceso de hardware. Solo necesita ejecutar un software en el ESP32 y tendrá una nueva red creada a través del punto de acceso suave (SoftAP) del ESP32.

Nota

En modo de punto de acceso suave (AP), el ESP32 actúa como un enrutador y asigna direcciones IP a los clientes conectados a él. Puede tener hasta 5 clientes como máximo.

Configurar el ESP32 en modo de punto de acceso (AP)

Para configurar el ESP32 para que funcione en modo de punto de acceso (SoftAP o AP), debe llamar a la función `WiFi_Mode (WIFI_AP)`, que toma WiFi_Mode como parámetro.

```
WiFi.mode(WIFI_AP);
```

Luego, debes asegurarte de haber definido las credenciales de red de la red WiFi ESP32 que vamos a crear. Ten en cuenta que este es el SSID que verán otros dispositivos al buscar redes WiFi cercanas y la contraseña definida será la que usarán otros dispositivos para unirse a tu red WiFi ESP32. Se pueden definir de la siguiente manera

```
// Replace With Your Desired SSID & Password
const char* ssid = "ESP32-AP";
const char* password = "01234567";
```

Y finalmente, debes llamar a la función `WiFi . softAP ()` para iniciar la operación del punto de acceso ESP32.

```
WiFi.softAP(ssid, password);
```

La función `WiFi . softAP ()` también puede tomar más parámetros opcionales en caso de que necesites utilizar alguno de ellos.

```
WiFi.softAP(const char* ssid, const char* password, int channel, int ssid_hidden, int max_connection);
```

Aquí está la lista completa de parámetros para esta función.

- `ssid` : nombre del punto de acceso (máximo de 63 caracteres)
- `contraseña` : tiene una longitud mínima de 8 caracteres, configúrela como NULL si desea que el punto de acceso esté abierto
- `canal` : número de canal Wi-Fi (1-13)
- `ssid_hidden` : (0 = SSID de difusión, 1 = SSID oculto)

Modo AP-STA (Punto de acceso + Estación) ESP32

Este modo permite que el ESP32 actúe como una estación WiFi y un punto de acceso simultáneamente. En este modo, el ESP32 puede conectarse a una red WiFi existente como una estación y, al mismo tiempo, actuar como un punto de acceso para que otros dispositivos se conecten a él. Esto puede resultar útil en situaciones en las que el ESP32 necesita conectarse a una red existente para acceder a Internet, pero también necesita crear su propia red para que otros dispositivos se conecten a ella.

Por ejemplo, un sistema de automatización del hogar donde el ESP32 necesita conectarse a la red WiFi del hogar para acceder a Internet, pero también necesita crear su propia red para que otros dispositivos inteligentes (estaciones) se conecten a él.

Configurar el ESP32 en modo AP-STA (estación WiFi + punto de acceso)

Para configurar el ESP32 WiFi para que funcione en modo AP-STA, debe llamar a esta función.

```
WiFi.mode(WIFI_AP_STA);
```

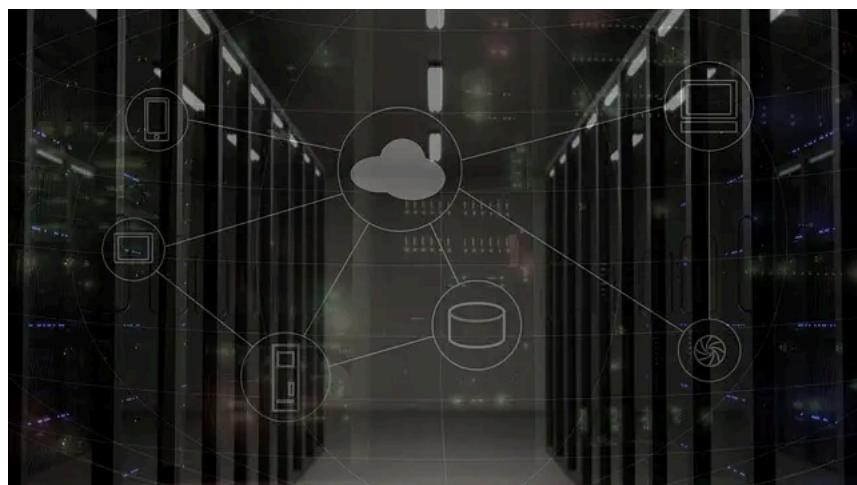
Detector de WiFi ESP32 (modo promiscuo)

Este modo permite que el ESP32 monitoree y capture todo el tráfico de Wi-Fi que pasa por un canal específico dentro de su rango. En otras palabras, el ESP32 se puede utilizar como un rastreador de redes inalámbricas para analizar y depurar el tráfico de la red Wi-Fi. Esto puede ser particularmente útil para los desarrolladores que trabajan en dispositivos o aplicaciones habilitados para Wi-Fi y necesitan analizar el tráfico de Wi-Fi para depurar problemas u optimizar el rendimiento.

Este modo no está definido ni implementado explícitamente en la biblioteca WiFi.h de ESP32, pero se puede lograr desarrollando una aplicación dedicada a esta tarea. Si suena interesante, podemos hacerlo en un tutorial aparte, por supuesto.

Redes WiFi ESP32

Ahora pasaremos a los conceptos y temas básicos de redes WiFi ESP32 con fragmentos de código de ejemplo sobre cómo implementar cada uno de ellos en sus proyectos de Arduino. Comprender los conceptos básicos de las redes WiFi ESP32 es fundamental para cualquier proyecto de IoT.



En esta sección, cubriremos conceptos clave y realizaremos operaciones básicas con la biblioteca WiFi del ESP32. Por ejemplo, cómo configurar el ESP32 para buscar redes WiFi, conectarse a una red existente, verificar el estado y la intensidad de la conexión y obtener las direcciones IP y MAC del WiFi del ESP32.

También aprenderemos cómo desconectarnos y reconectarnos a una red WiFi después de una falla o pérdida de conexión, cómo manejar eventos WiFi ESP32 sin sondear algunos indicadores o bloquear la CPU, y cómo crear aplicaciones de servidor web ESP32 (tanto en modo STA como en modo SoftAP).

ESP32 a continuación en este tutorial.

ESP32 escanea redes WiFi

El ESP32 puede escanear redes WiFi dentro de su rango y devolver los SSID de las redes encontradas y la intensidad de la señal de cada red. Encontrará un ejemplo de código de muestra en Arduino IDE para el escáner WiFi ESP32. Abra **Archivo > Ejemplos > WiFi > Esquema WiFiScan**.

Aquí está la lista completa del código de este ejemplo:

```
/*
 *  ESP32 WiFi Scanner Example. Examples > WiFi > WiFiScan
 *  Full Tutorial @ https://deepbluemedded.com/esp32-wifi-library-examples-tutorial-arduino/
 */

#include "WiFi.h"

void setup()
{
    Serial.begin(115200);

    // Set WiFi to station mode and disconnect from an AP if it was previously connected.
    WiFi.mode(WIFI_STA);
    WiFi.disconnect();
    delay(100);

    Serial.println("Setup done");
}

void loop()
{
    Serial.println("Scan start");

    // WiFi.scanNetworks will return the number of networks found.
    int n = WiFi.scanNetworks();
    Serial.println("Scan done");
    if (n == 0) {
        Serial.println("no networks found");
    } else {
        Serial.print(n);
        Serial.println(" networks found");
        Serial.println("Nr | SSID" | RSSI | CH | Encryption");
        for (int i = 0; i < n; ++i) {
            // Print SSID and RSSI for each network found
            Serial.printf("%2d", i + 1);
            Serial.print(" | ");
            Serial.printf("%-32.32s", WiFi.SSID(i).c_str());
            Serial.print(" | ");
            Serial.printf("%4d", WiFi.RSSI(i));
            Serial.print(" | ");
            Serial.printf("%2d", WiFi.channel(i));
            Serial.print(" | ");
            switch (WiFi.encryptionType(i))
            {
                case WIFI_AUTH_OPEN:
                    Serial.print("open");
                    break;
                case WIFI_AUTH_WEP:
                    Serial.print("WEP");
                    break;
                case WIFI_AUTH_WPA_PSK:
                    Serial.print("WPA");
                    break;
                case WIFI_AUTH_WPA2_PSK:
                    Serial.print("WPA2");
                    break;
                case WIFI_AUTH_WPA_WPA2_PSK:

```

```

        case WIFI_AUTH_WPA2_ENTERPRISE:
            Serial.print("WPA2-EAP");
            break;
        case WIFI_AUTH_WPA3_PSK:
            Serial.print("WPA3");
            break;
        case WIFI_AUTH_WPA2_WPA3_PSK:
            Serial.print("WPA2+WPA3");
            break;
        case WIFI_AUTH_WAPI_PSK:
            Serial.print("WAPI");
            break;
        default:
            Serial.print("unknown");
    }
    Serial.println();
    delay(10);
}
Serial.println("");

// Delete the scan result to free memory for code below.
WiFi.scanDelete();

// Wait a bit before scanning again.
delay(5000);
}

```

Este ejemplo es particularmente útil cuando intentas conectarte a una red cercana pero sigues sin poder hacerlo. Este ejemplo de escáner WiFi te mostrará la intensidad de la señal de cada red WiFi dentro del rango ESP32. El **RSSI** (indicador de intensidad de señal recibida) es lo que estás buscando si quieras saber la intensidad de la señal de cada red cercana.

La función `WiFi.scanNetworks()` se utiliza para manejar la lógica central del escaneo WiFi y devuelve la cantidad de redes encontradas.

```
int n = WiFi.scanNetworks();
```

En este punto tendrás el número de redes cercanas encontradas y podrás acceder a cualquiera de los siguientes parámetros de cada red.

- SSID
- RSSI
- Canal
- Tipo de cifrado

Para obtener e imprimir el SSID de cualquier red encontrada en la lista, llame a la siguiente función `WiFi.SSID(i)` y páselle el índice de la red encontrada.

```
Serial.print(WiFi.SSID(i));
```

Para obtener e imprimir el número de canal WiFi de una red encontrada, utilice la función `WiFi.channel(i)` y páselle el ID de la red encontrada como se muestra a continuación.

```
Serial.print(WiFi.channel(i));
```

Y, por último, puede obtener e imprimir el tipo de cifrado WiFi de cualquier red encontrada utilizando la función `WiFi.encryptionType(i)` como se muestra a continuación.

```

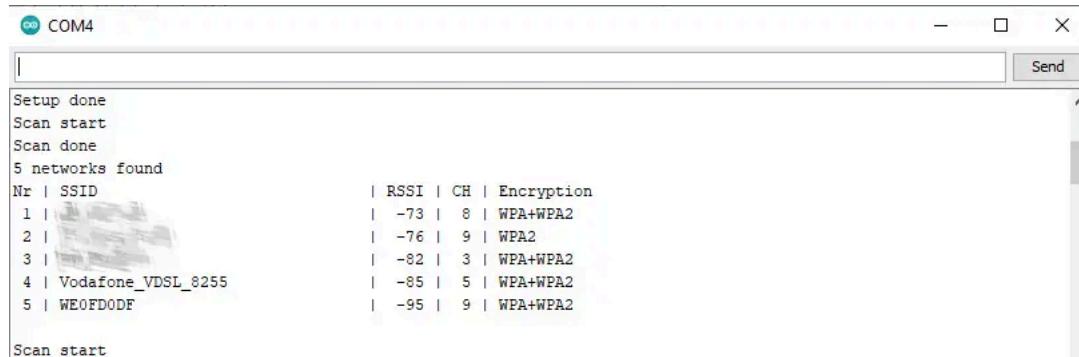
int WiFi_EncType = WiFi.encryptionType(i);
if(WiFi_EncType == xyz)
{
    Serial.println("EncType = xyz")
}

```

Donde las posibles opciones para el tipo de cifrado de la red WiFi son las siguientes:

- Autenticación WIFI WPA PSK
- Autenticación WIFI WPA2 PSK
- Autenticación WIFI WPA WPA2 PSK
- WIFI_AUTH_WPA2_EMPRESA
- Autenticación WIFI WPA3 PSK
- WIFI_AUTOMÁTICO_WPA2_WPA3_PSK
- WIFI_AUTOMÁTICO_WAPI_PSK

Y este es el resultado de ejecutar el boceto del escáner WiFi ESP32 en mi extremo.



```

Setup done
Scan start
Scan done
5 networks found
Nr | SSID           | RSSI | CH | Encryption
1  | [REDACTED]       | -73  | 8  | WPA+WPA2
2  | [REDACTED]       | -76  | 9  | WPA2
3  | [REDACTED]       | -82  | 3  | WPA+WPA2
4  | Vodafone_VDSL_8255 | -85  | 5  | WPA+WPA2
5  | WEOFDOUF         | -95  | 9  | WPA+WPA2

Scan start

```

Y puedo confirmar definitivamente que el WiFi de mi enrutador doméstico tiene la señal más fuerte, ya que está en la habitación de al lado, seguido por mi otro enrutador en otro piso, que tiene una señal un poco más débil, como era de esperar. Así que podemos concluir este tema y pasar al siguiente, que es intentar conectarse a una red WiFi existente con ESP32.

???? Lea también



Tutorial de escaneo de red WiFi ESP32

Este artículo proporcionará información más detallada sobre el escaneo de redes WiFi para encontrar todas las redes WiFi (puntos de acceso) en el alcance del WiFi ESP32.

Conexión del ESP32 a una red WiFi

Para conectar la placa ESP32 a una red WiFi existente, es necesario tener el SSID y la contraseña de esa red. Además, debe tener una intensidad de señal bastante buena para tener una conexión estable.

En la sección anterior, aprendimos a buscar redes cercanas con ESP32. Y deberías haber encontrado la red WiFi de tu enrutador doméstico en la lista. Entonces, el siguiente paso es llamar a la `función WiFi.begin (ssid , password)` que toma el SSID y la contraseña de la red deseada e intenta conectar tu ESP32 a ella.

Debe reemplazar los valores `ssid` y `password` por el nombre de su red WiFi y su contraseña real antes de crear y probar el ejemplo de código a continuación.

Aquí está el ejemplo de código completo:

```

#include <WiFi.h>

// Replace with your own network credentials
const char* ssid = "yourNetworkSSID";

```



```
void setup(){
    Serial.begin(115200);
    WiFi.mode(WIFI_STA);
    WiFi.begin(ssid, password);
    Serial.println("\nConnecting to WiFi Network ..");

    while(WiFi.status() != WL_CONNECTED){
        Serial.print(".");
        delay(100);
    }

    Serial.println("\nConnected to the WiFi network");
    Serial.print("Local ESP32 IP: ");
    Serial.println(WiFi.localIP());
}

void loop(){
    // Do Nothing
}
```

Y veamos cómo funciona.

En primer lugar, ya sabéis que el ESP32 WiFi puede funcionar como una estación WiFi (**STA**) que se conecta a redes existentes o como un punto de acceso WiFi (**AP**) que crea su propia red. Dado que estamos intentando conectarnos a una red WiFi existente, deberíamos configurar el modo WiFi del ESP32 en modo STA. Y esto es lo que hemos hecho en el primer paso.

```
WiFi.mode(WIFI_STA);
```

A continuación, llamamos a la función `WiFi . begin ()` que intenta conectar el ESP32 a la red deseada. Debes haber editado las cadenas de `SSID` y `contraseña` para que coincidan con el nombre de red y la contraseña reales de tu red WiFi.

```
WiFi.begin(ssid, password);
```

El proceso de conexión no es instantáneo, sino que llevará algún tiempo. Por lo tanto, debemos agregar un retraso y seguir verificando (sondeando) el estado de la conexión. Usaremos la función `WiFi . status ()` para obtener el estado de la conexión WiFi, y debería devolver `WL_CONNECTED` siempre que la conexión se establezca correctamente.

```
while(WiFi.status() != WL_CONNECTED)
{
    Serial.print(".");
    delay(100);
}
```

Cada vez que se establezca una conexión exitosa, el bucle while se interrumpirá e imprimiremos un mensaje en el monitor serial indicando este evento. También obtendremos la IP local del ESP32 y la imprimiremos también.

```
Serial.println("\nConnected to the WiFi network");
Serial.print("Local ESP32 IP: ");
Serial.println(WiFi.localIP());
```

Y ahora finalmente hemos conectado nuestro ESP32 a una red WiFi con éxito.

Nota

No se recomienda consultar el estado de la conexión WiFi en un bucle while. Una mejor solución es configurar una **interrupción del temporizador** y verificar periódicamente el estado de la conexión o incluso implementar un mecanismo **de tiempo de espera** después del cual podamos volver a intentar el proceso de conexión.

También existe una solución aún mejor que consiste en utilizar las funciones de devolución de llamadas **de eventos WiFi** para WiFi ESP32, que analizaremos más adelante en este tutorial, cerca del final. Así que asegúrate de quedarte y, por el bien de la

Consulte el siguiente tutorial para obtener más información sobre cómo conectar ESP32 a una red WiFi.

???? Lea también



Tutorial para conectar el ESP32 a una red WiFi

Este artículo brindará información más detallada sobre las diferentes formas de conectar ESP32 a una red WiFi existente, cómo depurar la pérdida de conexión e implementar una aplicación de reconexión automática.

Obtener el estado de la conexión WiFi del ESP32

Para obtener el estado de la conexión WiFi, puede utilizar la función `WiFi . status ()`, que devuelve un indicador que indica el estado actual de la conexión WiFi. A continuación, se incluye una tabla con los posibles valores de indicadores de estado y el significado de cada valor de estado.

IDENTIFICACIÓN	Estado de WiFi del ESP32	Descripción del estado
0	<code>WL_IDLE_STATUS</code>	Estado predeterminado antes de intentar conectarse a una red WiFi
1	<code>WL_NO_SSID_AVAIL</code>	El ESP32 no pudo encontrar la red WiFi. La red está demasiado lejos del ESP32 o el SSID de la red es incorrecto.
2	<code>WL_SCAN_COMPLETED</code>	Se ha completado el escaneo de redes WiFi cercanas
3	<code>WL_CONNECTED</code>	Conectado a una red WiFi (AP)
4	<code>WL_CONNECT_FAILED</code>	La conexión a una red WiFi (AP) ha fallado
5	<code>WL_CONNECTION_LOST</code>	Se ha perdido la conexión con una red WiFi
6	<code>WL_DISCONNECTED</code>	Desconectado de una red WiFi

Aquí hay un ejemplo de código que verifica el estado de la conexión WiFi y lo imprime mientras se intenta una nueva conexión. Esto puede ser muy útil durante la depuración de problemas de conexión WiFi porque puede determinar fácilmente dónde se bloquea o falla el ESP32.

No olvide cambiar el SSID y la contraseña para que coincidan con sus propias credenciales de red WiFi.

```
/*
 * ESP32 WiFi Connection Status Debugging
 * Full Tutorial @ https://deepbluemedded.com/esp32-wifi-library-examples-tutorial-arduino/
 */
#include <WiFi.h>
```



```

const char* password = "yourNetworkPassword";

int WiFiStatus;

String Get_WiFiStatus(int Status){
    switch(Status){
        case WL_IDLE_STATUS:
            return "WL_IDLE_STATUS";
        case WL_SCAN_COMPLETED:
            return "WL_SCAN_COMPLETED";
        case WL_NO_SSID_AVAIL:
            return "WL_NO_SSID_AVAIL";
        case WL_CONNECT_FAILED:
            return "WL_CONNECT_FAILED";
        case WL_CONNECTION_LOST:
            return "WL_CONNECTION_LOST";
        case WL_CONNECTED:
            return "WL_CONNECTED";
        case WL_DISCONNECTED:
            return "WL_DISCONNECTED";
    }
}

void setup(){
    Serial.begin(115200);
    Serial.println("Connecting..");
    WiFi.begin(ssid, password);
    WiFiStatus = WiFi.status();
    while(WiFiStatus != WL_CONNECTED){
        delay(250);
        WiFiStatus = WiFi.status();
        Serial.println(Get_WiFiStatus(WiFiStatus));
    }
    Serial.println("\nConnected To The WiFi Network");
    Serial.print("Local ESP32 IP: ");
    Serial.println(WiFi.localIP());
}

void loop(){
    // Do Nothing
}

```

Puedes ver los resultados a continuación. Se imprime el estado actual de la conexión WiFi, por lo que sabrás si se bloquea en algún momento o algo así.

```

COM4
| Send
rst:0xl (POWERON_RESET),boot:0xl3 (SPI_FAST_FLASH_BOOT)
configsip: 0, SPIWP:0xee
clk_drv:0x00,q_drv:0x00,d_drv:0x00,cs0_drv:0x00,hd_drv:0x00,wp_drv:0x00
mode:DIO, clock div:1
load:0x3fff0030,len:1184
load:0x40078000,len:13220
ho 0 tail 12 room 4
load:0x40080400,len:3028
entry 0x400805e4
Connecting..
WL_IDLE_STATUS
WL_IDLE_STATUS
WL_IDLE_STATUS
WL_CONNECTED

Connected to the WiFi network
Local ESP32 IP: 192.168.1.104

```

Autoscroll Show timestamp Newline 115200 baud Clear output

Comprueba la intensidad de la conexión WiFi del ESP32

```
Serial.print(WiFi.RSSI(i));
```

Esto le brinda un valor estimado de la intensidad de la señal para cualquier red circundante que haya encontrado durante el escaneo.

Nota

RSSI es una medida estimada de la intensidad de la señal WiFi para una red específica (enrutador o punto de acceso). El valor de retorno tiene el siguiente formato y unidad (- **x dBm**). Esto significa que un valor absoluto más bajo indica una conexión más potente. A continuación, se muestra cómo juzgar el valor RSSI y decidir la clasificación de intensidad de la señal:

- **RSSI > -30 dBm** : ¡Increíble!
- **RSSI < -55 dBm** : Muy buena señal
- **RSSI < -67 dBm** : bastante bueno
- **RSSI < -70 dBm** : aceptable
- **RSSI < -80 dBm** : No es bueno
- **RSSI < -90 dBm** : señal extremadamente débil (inutilizable)

Aquí hay un código de ejemplo para imprimir el valor RSSI de la red WiFi (establecida por mi enrutador doméstico) después de conectarme a ella con mi placa ESP32. Pruébelo usted mismo y no olvide cambiar el SSID y la contraseña de la red para que coincidan con sus credenciales de red.

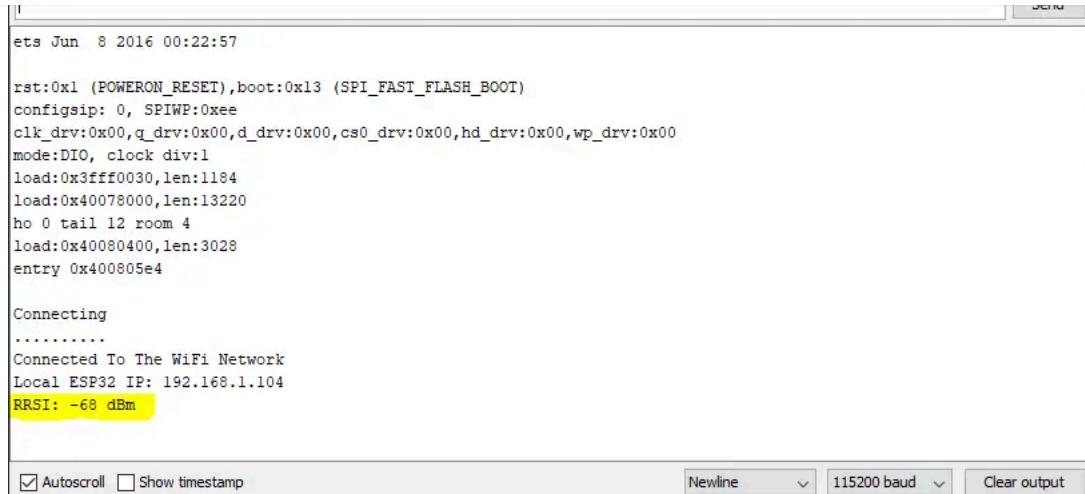
```
/*
 *  ESP32 WiFi Network RSSI (Signal Strength) Example
 *  Full Tutorial @ https://deepbluemedded.com/esp32-wifi-library-examples-tutorial-arduino/
 */
#include <WiFi.h>

// Replace with your own network credentials
const char* ssid = "yourNetworkSSID";
const char* password = "yourNetworkPassword";

void setup(){
    Serial.begin(115200);
    WiFi.mode(WIFI_STA);
    WiFi.begin(ssid, password);
    Serial.println("\nConnecting");
    // Keep Waiting Until Connection is Established
    while(WiFi.status() != WL_CONNECTED){
        Serial.print(".");
        delay(100);
    }
    // Connected Successfully
    Serial.println("\nConnected To The WiFi Network");
    Serial.print("Local ESP32 IP: ");
    Serial.println(WiFi.localIP());
    // Print The RSSI (Received Signal Strength Indicator)
    Serial.print("RSSI: ");
    Serial.print(WiFi.RSSI());
    Serial.println(" dBm");
}

void loop(){
    // Do Nothing
}
```

Como puede ver en el resultado a continuación, imprimió el valor RSSI para mi red WiFi, lo que muestra un nivel de intensidad de señal aceptable; el enrutador está moderadamente lejos de mi placa y está en otra habitación.



```

ets Jun 8 2016 00:22:57

rst:0xl (POWERON_RESET),boot:0x13 (SPI_FAST_FLASH_BOOT)
configsip: 0, SPIWP:0xee
clk_drv:0x00,q_drv:0x00,d_drv:0x00,cs0_drv:0x00,hd_drv:0x00,wp_drv:0x00
mode:DIO, clock div:1
load:0x3fff0030,len:1184
load:0x40078000,len:13220
ho 0 tail 12 room 4
load:0x40080400,len:3028
entry 0x400805e4

Connecting
.....
Connected To The WiFi Network
Local ESP32 IP: 192.168.1.104
RSSI: -68 dBm

```

Autoscroll Show timestamp Newline 115200 baud Clear output

También puede consultar el siguiente tutorial para saber más sobre cómo obtener la intensidad de la señal WiFi con ESP32.

???? Lea también



Intensidad de la señal WiFi del ESP32 y valor RSSI

Este artículo proporcionará información más detallada sobre la intensidad de la señal WiFi ESP32 y cómo obtener y juzgar los valores RSSI para determinar la calidad de la conexión WiFi entre su ESP32 y cualquier punto de acceso.

Obtenga la dirección MAC ESP32 y cámbiela

La dirección MAC es una dirección de identificación única que el fabricante asigna a cada dispositivo que se conectará a otras redes a nivel de hardware. Esto significa que es un identificador único para cada dispositivo de la red y tiene el siguiente formato: **25 : 3A : 42 : CE : B7 : FF** (por ejemplo).

Es una matriz de 6 bytes, cada byte está representado por 2 dígitos hexadecimales que pueden tomar cualquier valor entre (0...9-A...F). Y no importa si las letras son mayúsculas o minúsculas. La dirección MAC: **25 : 3A : 42 : CE : B7 : FF** es exactamente la misma que la dirección **MAC 25 : 3a : 42 : ce : b7 : ff**.

La dirección MAC del ESP32 se puede obtener fácilmente utilizando la biblioteca WiFi que ya está incorporada en Arduino Core para ESP32. Debe llamar a la **función WiFi . macAddress ()** que devuelve la dirección MAC en forma de una matriz de 6 bytes con formato de cadena.

A continuación se muestra un código de ejemplo que demuestra cómo obtener la dirección MAC del ESP32:

```

/*
 * ESP32 Get MAC Address Example
 * Full Tutorial @ https://deepbluemedded.com/esp32-wifi-library-examples-tutorial-arduino/
 */
#include <WiFi.h>

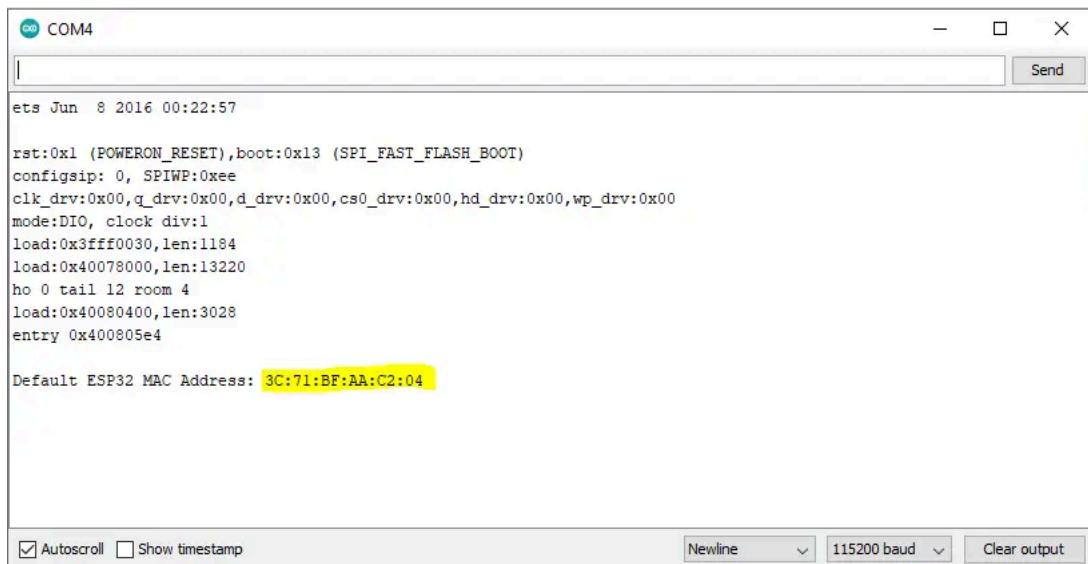
void setup(){
  Serial.begin(115200);
  Serial.print("\nDefault ESP32 MAC Address: ");
  Serial.println(WiFi.macAddress());
}

```



```
// Do Nothing
}
```

Y este es el resultado después de ejecutar este código en mi placa ESP32. Es posible que deba reiniciar su ESP32 si no captó el mensaje en el monitor serial en la primera ejecución.



Para cambiar la dirección MAC del ESP32 en Arduino IDE, puedes usar la función `esp_wifi_set_mac ()` de la biblioteca `esp_wifi.h`, que toma el modo WiFi y la matriz de direcciones MAC como argumentos .

Aquí hay un código de ejemplo para cambiar la dirección MAC a una dirección personalizada en Arduino IDE.

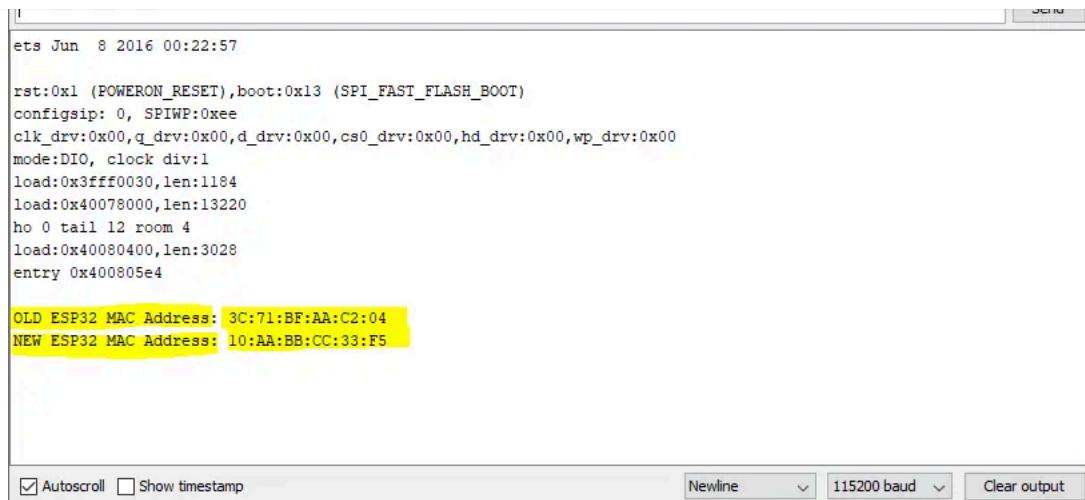
```
/*
 * ESP32 Set New MAC Address Example
 * Full Tutorial @ https://deepbluemedded.com/esp32-wifi-library-examples-tutorial-arduino/
 */
#include <WiFi.h>
#include <esp_wifi.h>

uint8_t New_MAC_Address[] = {0x10, 0xAA, 0xBB, 0xCC, 0x33, 0xF5};

void setup(){
    Serial.begin(115200);
    WiFi.mode(WIFI_STA);
    Serial.print("\nOLD ESP32 MAC Address: ");
    Serial.println(WiFi.macAddress());
    esp_wifi_set_mac(WIFI_IF_STA, New_MAC_Address);
    Serial.print("NEW ESP32 MAC Address: ");
    Serial.println(WiFi.macAddress());
}

void loop(){
    // Do Nothing
}
```

Este fue el resultado en la terminal serial después de ejecutar este código de ejemplo en mi placa ESP32.



```

ets Jun  8 2016 00:22:57

rst:0xl (POWERON_RESET),boot:0x13 (SPI_FAST_FLASH_BOOT)
configsip: 0, SPIWP:0xee
clk_drv:0x00,q_drv:0x00,d_drv:0x00,cs0_drv:0x00,hd_drv:0x00,wp_drv:0x00
mode:DIO, clock div:1
load:0x3fff0030,len:1184
load:0x40078000,len:13220
ho 0 tail 12 room 4
load:0x40080400,len:3028
entry 0x400805e4

OLD ESP32 MAC Address: 3C:71:BF:AA:C2:04
NEW ESP32 MAC Address: 10:AA:BB:CC:33:F5

```

Autoscroll Show timestamp Newline 115200 baud Clear output

Así que ahora hemos cambiado con éxito la dirección MAC.

Nota

El cambio de dirección MAC es temporal y se elimina cuando se reinicia el ESP32. Debe incluir el mismo código para cambiar la dirección MAC en cada aplicación donde sea necesario. De lo contrario, la dirección MAC predeterminada del ESP32 será la dirección efectiva del dispositivo.

También puedes consultar el siguiente tutorial para obtener más información sobre cómo obtener la dirección MAC predeterminada y configurar una dirección MAC personalizada para tu placa ESP32. Además, puedes obtener más consejos sobre cómo solucionar cualquier problema que pueda surgir al realizar esta operación.

???? Lea también



ESP32 Obtener y configurar la dirección MAC

Este artículo brindará información más detallada sobre la dirección MAC ESP32 y cómo obtener la dirección MAC predeterminada y cómo configurar una dirección MAC personalizada para su placa ESP32.

Obtener la dirección IP WiFi del ESP32

Una vez que se haya establecido una conexión exitosa a una red WiFi, podrá obtener fácilmente la dirección IP asignada a su ESP32 por el punto de acceso (router). Solo necesita llamar a la función `WiFi.localIP()`.

Y puedes imprimirla en el terminal serial tal como se muestra en el ejemplo a continuación.

```

Serial.print("Local ESP32 IP: ");
Serial.println(WiFi.localIP());

```

Establecer dirección IP estática para WiFi ESP32

dirección IP. La dirección IP asignada la decide aleatoriamente el DHCP.

Para asignar una dirección IP estática al ESP32 sin DHCP, tendremos que definir los siguientes parámetros.

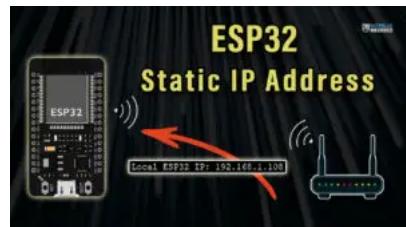
```
IPAddress local_IP(192, 168, 1, x); // Your Desired Static IP Address
IPAddress subnet(255, 255, 255, 0);
IPAddress gateway(192, 168, 1, 1);
IPAddress primaryDNS(192, 168, 1, 1); // Not Mandatory
IPAddress secondaryDNS(0, 0, 0, 0); // Not Mandatory
```

Luego debes llamar a la `función WiFi.config()` dentro de la función `setup()` como se muestra a continuación.

```
void setup() {
  ...
  // Configures Static IP Address
  if (!WiFi.config(local_IP, gateway, subnet, primaryDNS, secondaryDNS))
  {
    Serial.println("Configuration Failed!");
  }
}
```

Se recomienda encarecidamente seguir el tutorial a continuación para obtener ejemplos más completos sobre cómo configurar una dirección IP estática para la placa ESP32 de dos maneras diferentes. Además, se ofrecen algunos consejos para asegurarse de no tener problemas al probar estas técnicas.

???? Lea también



Establecer dirección IP estática para ESP32

Este artículo brindará información más detallada sobre la asignación de direcciones IP estáticas de ESP32 y lo que hace el DHCP. Además, le indicará cómo configurar de forma segura una dirección IP estática para su placa ESP32 de dos maneras diferentes.

Cambiar el nombre de host del ESP32

Para cambiar el nombre de host del ESP32, debe llamar a la `función WiFi.setHostname()` antes de llamar a `WiFi.mode()` y luego a `WiFi.begin()` en exactamente el mismo orden.

Aquí hay un código de ejemplo que demuestra cómo configurar un nombre de host ESP32 personalizado en Arduino IDE.

```
/*
 * ESP32 WiFi HostName Change Example
 * Full Tutorial @ https://deepbluemedded.com/esp32-wifi-library-examples-tutorial-arduino/
 */
#include <WiFi.h>

// Replace with your own network credentials
const char* ssid = "yourNetworkSSID";
const char* password = "yourNetworkPassword";
const char* MyHostName = "ESP32-Test";

void setup(){
  Serial.begin(115200);
  WiFi.setHostname(MyHostName);
```



```

Serial.println("\nConnecting to WiFi Network ..");
// Wait until connection is established
while(WiFi.status() != WL_CONNECTED){
    Serial.print(".");
    delay(100);
}
// Print ESP32's IP & HostName
Serial.println("\nConnected to the WiFi network");
Serial.print("Local ESP32 IP: ");
Serial.println(WiFi.localIP());
Serial.print("ESP32 HostName: ");
Serial.println(WiFi.getHostname());
}

void loop(){
    // Do Nothing
}

```

Y este es el resultado después de ejecutar este código en mi placa ESP32. Es posible que deba reiniciar su ESP32 si no captó el mensaje en el monitor serial en la primera ejecución.

```

COM4
|
|
ets Jun  8 2016 00:22:57

rst:0x1 (POWERON_RESET),boot:0x13 (SPI_FAST_FLASH_BOOT)
configsip: 0, SPIWP:0xee
clk_drv:0x00,q_drv:0x00,d_drv:0x00,cs0_drv:0x00,hd_drv:0x00,wp_drv:0x00
mode:DIO, clock div:1
load:0x3fff0030,len:1184
load:0x40078000,len:13220
ho 0 tail 12 room 4
load:0x40080400,len:3028
entry 0x400805e4

Connecting to WiFi Network ..
.....
Connected to the WiFi network
Local ESP32 IP: 192.168.1.104
ESP32 HostName: ESP32-Test

```

Y ahora mi enrutador ve el nuevo nombre de host ESP32 después de ejecutar el código de ejemplo anterior.

Wireless Clients			
ID	Name	IP Address	MAC Address
1	ESP32-Test	192.168.1.104	3C-71-BF-AA-C2-04

A veces, es necesario esperar un poco o incluso reiniciar el enrutador para ver el nuevo nombre de host actualizado para el ESP32. Por lo tanto, puede probar esto si no se actualizó automáticamente al nuevo nombre. En mi caso, fue un evento casi instantáneo y no necesité realizar ningún paso adicional después de cargar el código.

Nota

La cadena de nombre de host de ESP32 solo puede tener letras, números y el símbolo de guión "-". Siga esta regla para no crear un nombre de host no válido. Intente que sea lo más compacto y descriptivo posible.

Se recomienda encarecidamente seguir el tutorial a continuación para obtener más información sobre cómo cambiar el nombre de host predeterminado del ESP32.



Establecer dirección IP estática para ESP32

Este artículo brindará información más detallada sobre el nombre de host ESP32 y cómo obtener y cambiar el nombre de host WiFi predeterminado para ESP32.

Desconectar WiFi ESP32

Para desconectarse de una red WiFi a la que está conectado actualmente, simplemente utilice la función de `desconexión WiFi()`.

```
WiFi.disconnect();
```

Luego, el ESP32 puede conectarse a una nueva red WiFi o volver a conectarse a la misma red anterior. Realmente depende de la aplicación y de la funcionalidad lógica que intente lograr.

Eventos WiFi ESP32

La biblioteca WiFi ESP32 tiene una característica muy poderosa conocida como Eventos WiFi. Básicamente, se trata de algunas funciones de devolución de llamada para casi todos los eventos WiFi que le permiten configurar una función de controlador para cualquier evento que le interese. Esto elimina la necesidad de sondear los indicadores de estado WiFi o poner la CPU en un estado de espera ocupado o mantenerla bloqueada.

La CPU puede realizar la lógica básica de su aplicación sin preocuparse por el indicador de estado de WiFi, que puede pasar a un estado desconectado, por ejemplo. Porque, siempre que el WiFi pase a un estado desconectado, se activará el evento correspondiente y se llamará a la función del controlador de devolución de llamada que haya definido y realizará la lógica necesaria para ese evento (para un evento desconectado, el controlador debe intentar la reconexión, por ejemplo).

Aquí hay una lista completa de todos los eventos WiFi ESP32 que se encuentran en la biblioteca WiFi con una breve descripción de lo que significa cada evento.

IDENTIFICACIÓN	Evento WiFi ESP32	Descripción del evento
0	<code>ARDUINO_EVENT_WIFI_READY</code>	Controlador WiFi ESP32 inicializado
1	<code>ARDUINO_EVENT_WIFI_SCAN_DONE</code>	Escaneo WiFi completado
2	<code>ARDUINO_EVENT_WIFI_STA_START</code>	Modo estación iniciado
3	<code>ARDUINO_EVENT_WIFI_STA_STOP</code>	Modo estación detenido
4	<code>ARDUINO_EVENT_WIFI_STA_CONNECTED</code>	Modo estación conectado a un AP
5	<code>ARDUINO_EVENT_WIFI_STA_DISCONNECTED</code>	Modo estación desconectado de un AP
6	<code>ARDUINO_EVENT_WIFI_STA_AUTHMODE_CHANGE</code>	El modo de autenticación del AP conectado al modo de estación ha cambiado

7	<code>ARDUINO_EVENT_WIFI_STA_GOT_IP</code>	El modo estación obtuvo una dirección IP del AP conectado
8	<code>ARDUINO_EVENT_WIFI_STA_GOT_IP6</code>	El modo estación obtuvo una dirección IPv6 del AP conectado
9	<code>ARDUINO_EVENT_WIFI_STA_LOST_IP</code>	El modo estación perdió la dirección IP del AP conectado
10	<code>ARDUINO_EVENT_WIFI_AP_START</code>	Modo de punto de acceso iniciado
11	<code>ARDUINO_EVENT_WIFI_AP_STOP</code>	El modo de punto de acceso se detuvo
12	<code>ARDUINO_EVENT_WIFI_AP_STACONNECTED</code>	Una estación conectada al modo de punto de acceso
13	<code>ARDUINO_EVENT_WIFI_AP_STADISCONNECTED</code>	Una estación desconectada del modo de punto de acceso
14	<code>ARDUINO_EVENT_WIFI_AP_STAIASSIGNED</code>	Dirección IP asignada a una estación conectada al modo de punto de acceso
15	<code>ARDUINO_EVENT_WIFI_AP_PROBEREQRECVED</code>	El modo de punto de acceso ha recibido una solicitud de sonda
16	<code>ARDUINO_EVENT_WIFI_AP_GOT_IP6</code>	Dirección IPv6 obtenida por el modo de punto de acceso
17	<code>ARDUINO_EVENT_WIFI_FTM_REPORT</code>	Se ha recibido un informe de Gestión de Transición Rápida (FTM)
18	<code>ARDUINO_EVENT_ETH_START</code>	Interfaz Ethernet iniciada
19	<code>ARDUINO_EVENT_ETH_STOP</code>	Interfaz Ethernet detenida
20	<code>ARDUINO_EVENT_ETH_CONNECTED</code>	Interfaz Ethernet conectada a una red
21	<code>ARDUINO_EVENT_ETH_DISCONNECTED</code>	Interfaz Ethernet desconectada de una red
22	<code>ARDUINO_EVENT_ETH_GOT_IP</code>	Dirección IP obtenida por la interfaz Ethernet
23	<code>ARDUINO_EVENT_ETH_GOT_IP6</code>	Dirección IPv6 obtenida por la interfaz Ethernet
24	<code>ARDUINO_EVENT_WPS_ER_SUCCESS</code>	Configuración protegida de Wi-Fi (WPS) exitosa
25	<code>ARDUINO_EVENT_WPS_ER_FAILED</code>	La configuración protegida de Wi-Fi (WPS) falló
26	<code>ARDUINO_EVENT_WPS_ER_TIMEOUT</code>	Se agotó el tiempo de espera de la configuración protegida de Wi-Fi (WPS)
27	<code>ARDUINO_EVENT_WPS_ER_PIN</code>	Se ha generado el PIN de configuración protegida de Wi-Fi (WPS)
28	<code>ARDUINO_EVENT_WPS_ER_PBC_OVERLAP</code>	Se detectó superposición de configuración protegida Wi-Fi (WPS)

29	<code>ARDUINO_EVENT_SC_SCAN_DONE</code>	Configuración inteligente completada
30	<code>ARDUINO_EVENT_SC_FOUND_CHANNEL</code>	Canal encontrado por Smart Config
31	<code>ARDUINO_EVENT_SC_GOT_SSID_PSWD</code>	Smart Config obtuvo el SSID y la contraseña
32	<code>ARDUINO_EVENT_SC_SEND_ACK_DONE</code>	ACK enviado por Smart Config
33	<code>ARDUINO_EVENT_PROV_INIT</code>	Se inició el aprovisionamiento
34	<code>ARDUINO_EVENT_PROV_deinit</code>	Aprovisionamiento detenido
35	<code>ARDUINO_EVENT_PROV_START</code>	Se ha iniciado el aprovisionamiento.
36	<code>ARDUINO_EVENT_PROV_END</code>	El aprovisionamiento ha finalizado.
37	<code>ARDUINO_EVENT_PROV_CRED_RECV</code>	Se han recibido las credenciales durante el aprovisionamiento.
38	<code>ARDUINO_EVENT_PROV_CRED_FAIL</code>	El aprovisionamiento falló debido a credenciales no válidas.
39	<code>ARDUINO_EVENT_PROV_CRED_SUCCESS</code>	Aprovisionamiento completado exitosamente.
40	<code>ARDUINO_EVENT_MAX</code>	NULO

Hay muchas aplicaciones en las que es necesario utilizar los eventos WiFi ESP32 integrados para configurar sus funciones de controlador personalizadas para los eventos de interés sin tener que mantener la CPU bloqueada esperando que el indicador de estado tenga un valor determinado o desperdiciar tiempo de CPU de cualquier otra manera.

En la siguiente sección, implementaremos una aplicación de reconexión automática de WiFi basada en eventos de WiFi. Configuraremos un controlador de devolución de llamada para el evento `ARDUINO_EVENT_WIFI_STA_DISCONNECTED`, que intentará la reconexión llamando a la función `WiFi.begin()`.

También puedes consultar el tutorial a continuación para ver dos ejemplos más sencillos de uso de eventos WiFi en proyectos prácticos explicados paso a paso para que puedas entenderlo fácilmente. Estoy bastante seguro de que, una vez que entiendas este concepto, la mayoría de tus proyectos WiFi ESP32 incorporarán funciones de manejo de devolución de llamadas de eventos WiFi.

???? Lea también

Conectarse y reconectar ESP32 a la red WiFi (con eventos WiFi)

Este tutorial le brindará 2 ejemplos más de eventos WiFi para aprender más sobre cómo configurar funciones de devolución de llamada (controladores de eventos) para proyectos WiFi ESP32.

Hay muchas formas de implementar una estrategia de reconexión WiFi para ESP32 en caso de que pierda la conexión WiFi a la red. Estas son algunas opciones posibles:

1. Reconectar con `WiFi` . Función `reconectar()`
2. Restablecimiento del tiempo de espera para lograr la reconexión automática
3. Uso de `WiFi . persistente()` Función incorporada de reconexión automática
4. Reconexión automática con eventos WiFi ESP32

Todas las opciones enumeradas anteriormente se demuestran en detalle en un tutorial separado, pero para simplificar, analizaremos solo una opción en este tutorial, que es la n.º 4 (Reconexión automática con eventos WiFi).

Por lo tanto, implementaremos una técnica de reconexión automática basada en las funciones de devolución de llamada de eventos WiFi ESP32. Estableceremos un controlador de devolución de llamada para el evento `ARDUINO_EVENT_WIFI_STA_DISCONNECTED` , que intentará la reconexión llamando a la función `WiFi . begin()` .

La función de controlador de eventos de desconexión de WiFi intentará restablecer la conexión siempre que se pierda la conexión WiFi entre ESP32 y el punto de acceso, sin necesidad de sondear ninguna bandera ni mantener la CPU en un estado bloqueado.

Aquí está el código de ejemplo completo:

```
/*
 *  ESP32 WiFi Auto Re-Connect
 *  Full Tutorial @ https://deepbluemedded.com/esp32-wifi-library-examples-tutorial-arduino/
 */
#include <WiFi.h>

// Replace with your own network credentials
const char* ssid = "yourNetworkSSID";
const char* password = "yourNetworkPassword";

void ConnectedToAP_Handler(WiFiEvent_t wifi_event, WiFiEventInfo_t wifi_info) {
    Serial.println("Connected To The WiFi Network");
}

void GotIP_Handler(WiFiEvent_t wifi_event, WiFiEventInfo_t wifi_info) {
    Serial.print("Local ESP32 IP: ");
    Serial.println(WiFi.localIP());
}

void WiFi_Disconnected_Handler(WiFiEvent_t wifi_event, WiFiEventInfo_t wifi_info) {
    Serial.println("Disconnected From WiFi Network");
    // Attempt Re-Connection
    WiFi.begin(ssid, password);
}

void setup() {
    Serial.begin(115200);

    WiFi.mode(WIFI_STA);
    WiFi.onEvent(ConnectedToAP_Handler, ARDUINO_EVENT_WIFI_STA_CONNECTED);
    WiFi.onEvent(GotIP_Handler, ARDUINO_EVENT_WIFI_STA_GOT_IP);
    WiFi.onEvent(WiFi_Disconnected_Handler, ARDUINO_EVENT_WIFI_STA_DISCONNECTED);
    WiFi.begin(ssid, password);
    Serial.println("Connecting to WiFi Network ...");
}

void loop() {
    // Do Nothing
}
```

Para realizar la prueba, tuve que desactivar el WiFi de mi enrutador (haga clic en el pequeño botón WLAN de su enrutador para activarlo o desactivarlo). Luego, lo volví a activar para verificar si mi ESP32 se conectaba automáticamente. ¡Y así fue!

Servidores web WiFi ESP32

En redes, un **servidor web** se define como un programa que sirve HTML y otros documentos web a navegadores web a través de Internet. Con ESP32, puede:

- Construye tus propios servidores web y controla los dispositivos conectados a ellos desde cualquier parte del mundo.
- Recopilar datos de sensores, procesarlos y enviar los resultados a la web en tiempo real para que otras personas puedan acceder a ellos.
- Obtener datos de la web, procesarlos y realizar acciones en función de ellos.
- Cree su propia red de dispositivos (servidor web + clientes) o cualquier otra topología para hacer que los dispositivos integrados se comuniquen entre sí dentro de una red local de su creación que puede o no estar conectada a la web mundial.
- ¡Las posibilidades son infinitas!

Dicho esto, las posibilidades son infinitas, pero la definición de servidor web seguirá siendo el nombre. Y no confunda el acto de usar ESP32 como servidor web con un modo o algo así. Al fin y al cabo, no es un modo de funcionamiento, es solo una aplicación. Si el ESP32 está ejecutando un programa para servir HTML o cualquier documento web a través de Internet, lo llamaremos servidor web.

Sin embargo, las aplicaciones de servidor web ESP32 se pueden implementar de diferentes maneras según el modo WiFi que queramos utilizar. En otras palabras, puedes crear un **servidor web ESP32 (en modo estación)** o un **servidor web (en modo punto de acceso)**. Depende de cómo te gustaría que sea la red en tu aplicación.

La elección entre un servidor web WiFi (**modo estación**) y un **servidor web WiFi (modo punto de acceso)** depende, en última instancia, de los requisitos específicos del proyecto. Si el dispositivo necesita interactuar con una red WiFi preexistente y acceder a Internet, un **servidor web en modo estación** es la mejor opción. Por otro lado, si el dispositivo necesita crear su propia red para que se conecten otros dispositivos, un **servidor web en modo punto de acceso** es la mejor opción.

A continuación, se muestra una comparación de ambas aplicaciones. En la siguiente sección, implementaremos un ejemplo práctico para cada una de ellas para asegurarnos de que comprenda claramente sus diferencias.

Servidor web ESP32 (modo estación)	Servidor web ESP32 (modo de punto de acceso)



una red para que otros dispositivos puedan conectarse a ella e intercambiar datos. El ejemplo anterior es para un servidor web ESP32 que lee el sensor de temperatura LM35 y proporciona estos datos a otros dispositivos.

El ESP32 se conecta a la red WiFi del enrutador (punto de acceso) y otros dispositivos de la misma red pueden ver los datos de temperatura compartidos por el servidor web ESP32 (modo estación).

propia red y otros dispositivos pueden conectarse a ella. El ejemplo anterior es para un servidor web ESP32 que lee el sensor de temperatura LM35 y proporciona estos datos a otros dispositivos.

El ESP32 crea una red WiFi y otros dispositivos en esta red pueden obtener los datos de temperatura compartidos por el servidor web ESP32 (modo de punto de acceso).

Proyectos de ejemplo de WiFi ESP32

Ahora, crearemos un par de proyectos de ejemplo para practicar lo que hemos aprendido hasta ahora sobre los modos WiFi, las redes y las funciones de la biblioteca WiFi de ESP32. Juntaremos todo para crear una aplicación de servidor web ESP32 muy simple, un punto de acceso WiFi basado en ESP32 y una comunicación WiFi cliente-servidor ESP32 entre dos placas ESP32.

Cada proyecto de ejemplo se ilustra con más detalle en un tutorial separado, pero resumiremos todo a continuación y será mejor que los revise uno por uno para obtener más detalles y consejos útiles.

1. Servidor web ESP32 (estación WiFi)

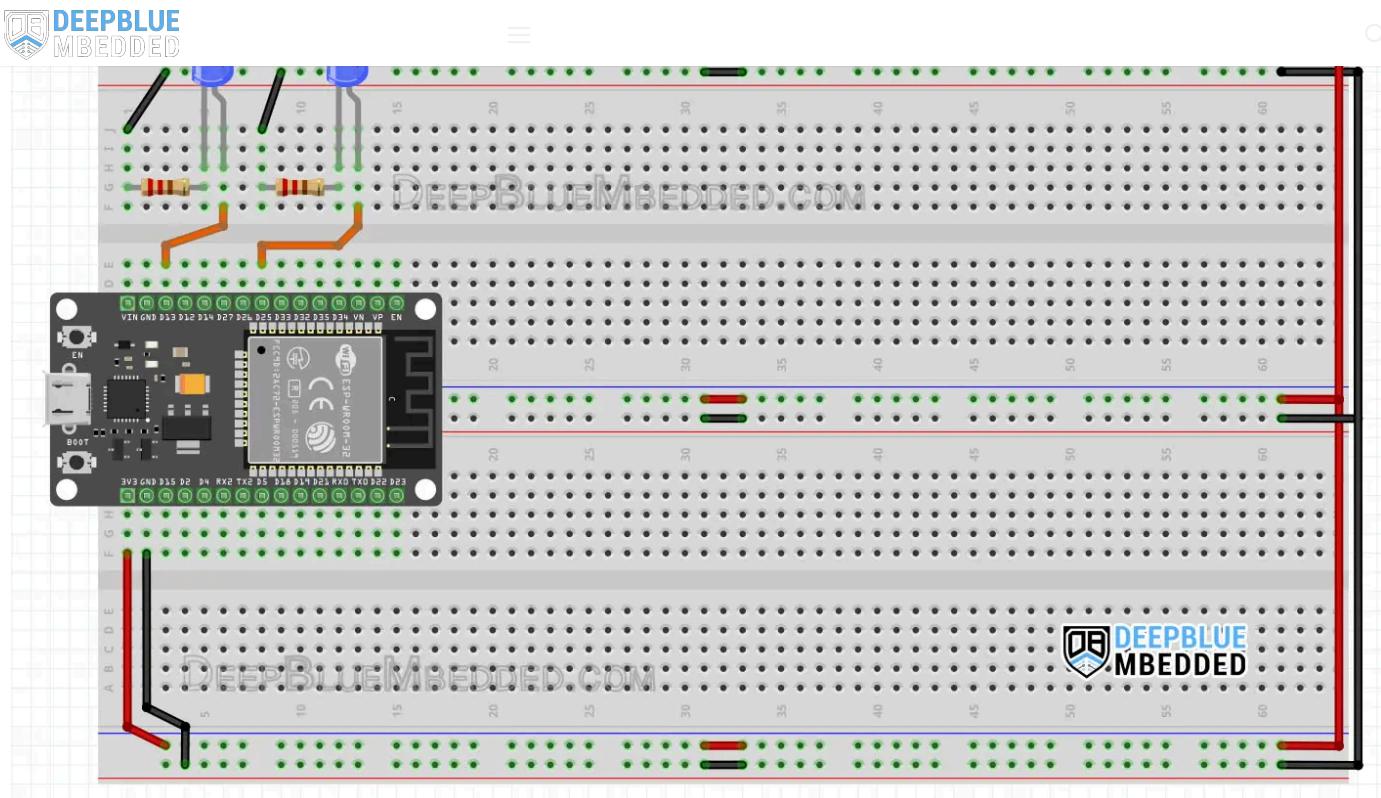
En este ejemplo, crearemos un servidor web ESP32 (en **modo de estación "STA"**) que servirá una página web que proporciona a los clientes las siguientes funciones:

1. Botón para controlar el estado de un LED (LED1)
2. Barra deslizante para controlar el brillo de un segundo LED (LED2)

El servidor web ESP32 funcionará en modo estación, lo que significa que se conectará a su red WiFi doméstica y luego será posible probar la aplicación con cualquier dispositivo de la red (computadora portátil, computadora de escritorio, teléfono inteligente, etc.).

Alambrado

Aquí se explica cómo conectar las señales de salida a los LED.



Código de ejemplo

Aquí está la lista completa de códigos para este ejemplo ([modo de estación de servidor web ESP32](#))

```
/*
*  ESP32 WebServer (Station Mode) Example
*  Full Tutorial @ https://deepbluembedded.com/esp32-wifi-library-examples-tutorial-arduino/
*/
#include <WiFi.h>
#include <WebServer.h>

// Replace with your own network credentials
const char* ssid = "yourNetworkSSID";
const char* password = "yourNetworkPassword";

WebServer server(80);

const int led1Pin = 25;
const int led2Pin = 13;
int ledState = LOW;
int ledBrightness = 0;

const char html[] PROGMEM = R"rawliteral(
<!DOCTYPE html>
<html>
<head>
  <title>ESP32 Web Server</title>
  <style>
    body {
      font-family: sans-serif;
    }
    .container {
      display: flex;
      flex-direction: column;
      align-items: center;
    }
    button {
      font-size: 1.5em;
      margin: 10px;
      padding: 10px;
      background-color: lightgray; /* Set the initial background color */
    }
  </style>
<body>
  <h1>ESP32 Web Server</h1>
  <div>
    <h2>LED Control</h2>
    <div>
      <input type="button" value="Turn On" onclick="document.getElementById('led1').style.display = 'block'; document.getElementById('led2').style.display = 'block';" />
      <input type="button" value="Turn Off" onclick="document.getElementById('led1').style.display = 'none'; document.getElementById('led2').style.display = 'none';" />
    </div>
    <div>
      <input type="button" value="Increase Brightness" onclick="document.getElementById('led1').style.display = 'block'; document.getElementById('led2').style.display = 'block';" />
      <input type="button" value="Decrease Brightness" onclick="document.getElementById('led1').style.display = 'block'; document.getElementById('led2').style.display = 'block';" />
    </div>
  </div>
</body>
)rawliteral";
```



```

        background-color: green; /* Set the background color when the LED is ON */
    }
    input[type=range] {
        width: 50%;
    }

```

</head>

</body>

<div class="container">

<h1>ESP32 Web Server</h1>

<h3>LED1 State Control</h3>

<button id="button" onclick="toggleLED()">LED: OFF</button>

<h3>LED2 Brightness Control</h3>

<input type="range" id="trackbar" min="0" max="255" step="1" value="0" oninput="setBrightness()">

</div>

<script>

var ledState = false;

var ledBrightness = 0;

var button = document.getElementById('button');

var trackbar = document.getElementById('trackbar');

function toggleLED() {

ledState = !ledState;

var xhr = new XMLHttpRequest();

xhr.open('GET', '/led?state=' + (ledState ? '1' : '0'), true);

xhr.send();

button.innerHTML = 'LED: ' + (ledState ? 'ON' : 'OFF');

button.classList.toggle('on', ledState); // Add or remove the 'on' class based on the LED state

}

function setBrightness() {

ledBrightness = trackbar.value;

var xhr = new XMLHttpRequest();

xhr.open('GET', '/brightness?value=' + ledBrightness, true);

xhr.send();

}

</script>

</body>

</html>

)rawliteral";

void handleRoot() {

server.send(200, "text/html", html);

}

void handleLED() {

ledState = server.arg("state").toInt();

digitalWrite(led1Pin, ledState);

server.send(200, "text/plain", String(ledState));

}

void handleBrightness() {

ledBrightness = server.arg("value").toInt();

analogWrite(led2Pin, ledBrightness);

server.send(200, "text/plain", String(ledBrightness));

}

void setup() {

Serial.begin(115200);

pinMode(led1Pin, OUTPUT);

pinMode(led2Pin, OUTPUT);

digitalWrite(led1Pin, ledState);

analogWrite(led2Pin, ledBrightness);

// WiFi Configuration & Init

WiFi.mode(WIFI_STA);

WiFi.begin(ssid, password);

Serial.print("Connecting To WiFi Network .");

while (WiFi.status() != WL_CONNECTED) {

```

}
// Connected Successfully
Serial.println("\nConnected To The WiFi Network");
Serial.print("Local ESP32 IP: ");
Serial.println(WiFi.localIP());
// Attach Server Handler Functions & Start WebServer
server.on("/", handleRoot);
server.on("/led", handleLED);
server.on("/brightness", handleBrightness);
server.begin();
}

void loop() {
  server.handleClient();
}

```

La página web HTML tiene todos los elementos que necesitamos para esta aplicación (botón de control LED, Trackbar, encabezados de etiquetas, hoja de estilo CSS y código javascript para manejar eventos).

Luego, definiremos algunas funciones de controlador para los siguientes eventos: servir la página web HTML, manejar el cambio de estado del LED1 y manejar el cambio de valor de brillo del LED2.

```

void handleRoot() {
  server.send(200, "text/html", html);
}

void handleLED() {
  ledState = server.arg("state").toInt();
  digitalWrite(led1Pin, ledState);
  server.send(200, "text/plain", String(ledState));
}

void handleBrightness() {
  ledBrightness = server.arg("value").toInt();
  analogWrite(led2Pin, ledBrightness);
  server.send(200, "text/plain", String(ledBrightness));
}

```

Luego, en la función `de configuración ()` : configuramos e inicializamos el WiFi y el WebServer. E imprimimos el mensaje de conexión WiFi exitosa, así como la dirección IP del WebServer. De esta manera, ya sabes la dirección IP a la que necesitas acceder en tu navegador web para interactuar con el WebServer ESP32.

```

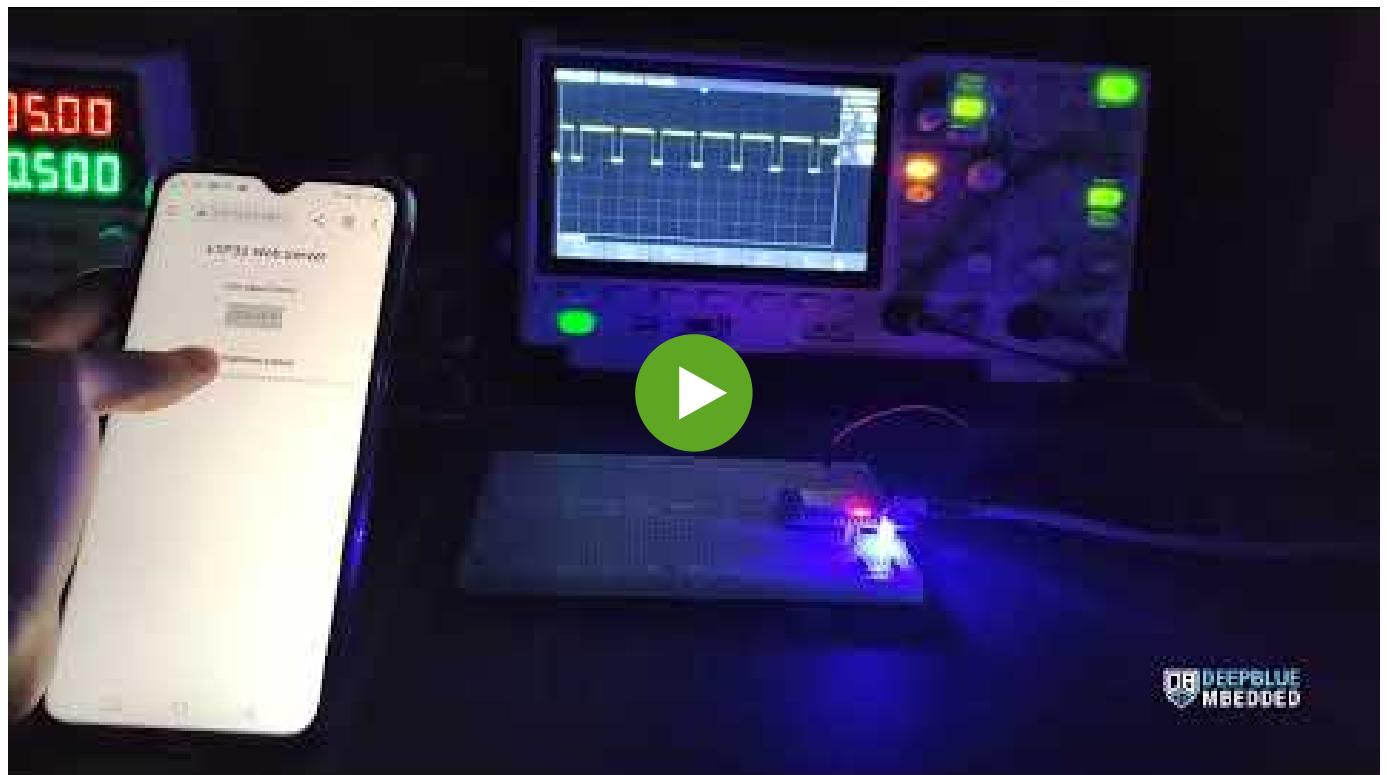
void setup() {
  Serial.begin(115200);
  pinMode(led1Pin, OUTPUT);
  pinMode(led2Pin, OUTPUT);
  digitalWrite(led1Pin, ledState);
  analogWrite(led2Pin, ledBrightness);
  // WiFi Configuration & Init
  WiFi.mode(WIFI_STA);
  WiFi.begin(ssid, password);
  Serial.print("Connecting To WiFi Network .");
  while (WiFi.status() != WL_CONNECTED) {
    Serial.print(".");
    delay(100);
  }
  // Connected Successfully
  Serial.println("\nConnected To The WiFi Network");
  Serial.print("Local ESP32 IP: ");
  Serial.println(WiFi.localIP());
  // Attach Server Handler Functions & Start WebServer
  server.on("/", handleRoot);
  server.on("/led", handleLED);
  server.on("/brightness", handleBrightness);
  server.begin();
}

```

Resultados de la prueba

Después de ejecutar este proyecto de ejemplo en mi ESP32, abrí el monitor serial para obtener la dirección IP del servidor web. Y quedó como se muestra a continuación.

Luego, fui al navegador web de mi computadora y escribí la dirección IP (**192.168.1.104**) en la barra de URL. La página web del servidor web ESP32 se cargó y pude controlar LED1 y LED2. También pude lograr lo mismo a través del navegador web del teléfono inteligente, como puede ver en la breve demostración a continuación.



2. Cree un servidor web WiFi (punto de acceso) ESP32

En este ejemplo, crearemos un servidor web ESP32 (en **modo de punto de acceso "AP"**) que servirá una página web que proporciona a los clientes las siguientes funciones:

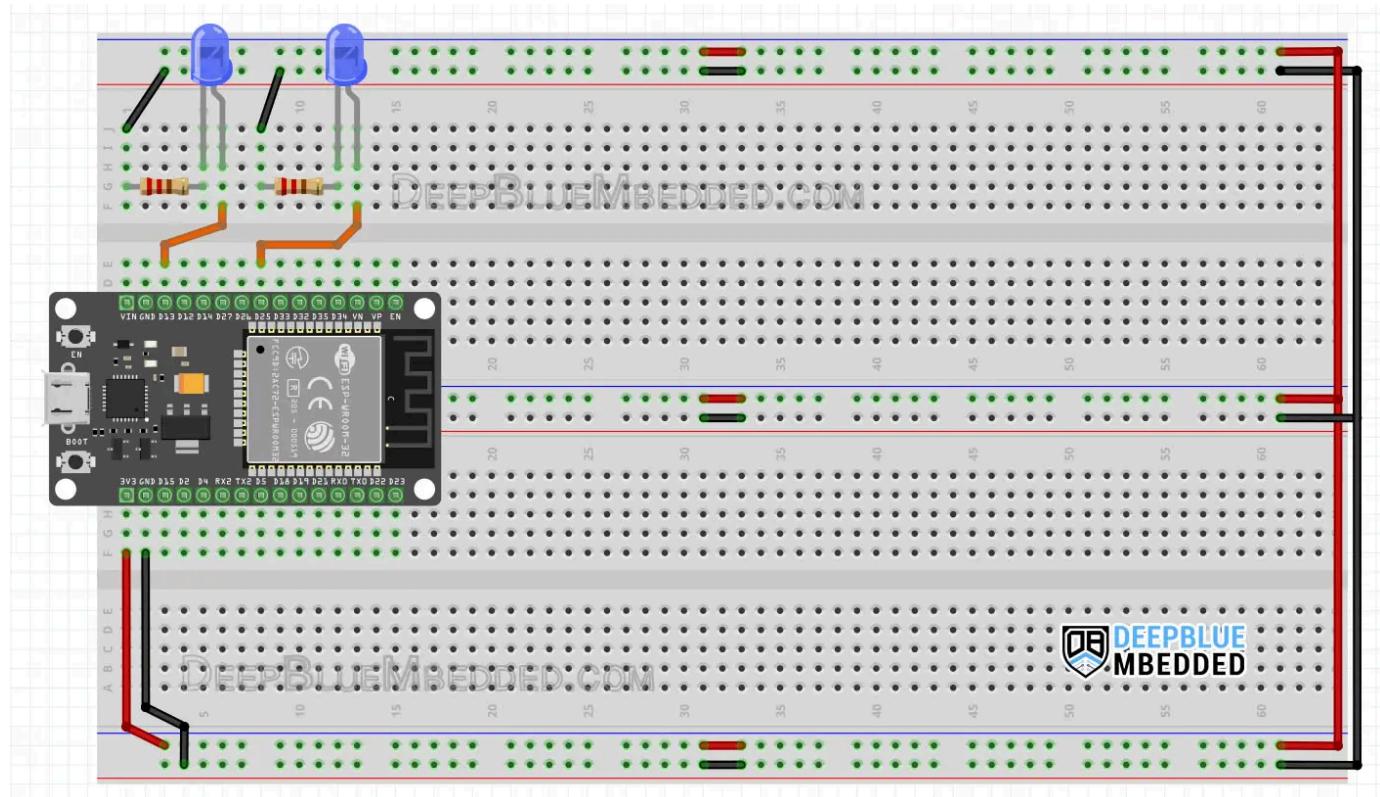
1. Botón para controlar el estado de un LED (LED1)
2. Barra deslizante para controlar el brillo de un segundo LED (LED2)

teléfono inteligente, etc.) después de conectarse a la red WiFi del ESP32.

Este es exactamente el mismo ejemplo anterior con una única diferencia, que es el modo de funcionamiento WiFi del ESP32. Es AP (SoftAP Access Point Mode) el que permite al ESP32 crear su propia red WiFi y nos conectaremos a ella con un teléfono inteligente para probar la aplicación.

Alambrado

A continuación se explica cómo conectar las señales de salida a los LED. (Igual que en el ejemplo anterior)



Código de ejemplo

Aquí está la lista completa de códigos para este ejemplo (**modo de punto de acceso del servidor web ESP32**)

```
/*
 *  ESP32 WebServer (Access Point "AP" Mode) Example
 *  Full Tutorial @ https://deepbluembedded.com/esp32-wifi-library-examples-tutorial-arduino/
 */
#include <WiFi.h>
#include <WebServer.h>

// Replace With Your Desired SSID & Password
const char* ssid = "ESP32-LED-AP";
const char* password = "01234567";

WebServer server(80);

const int led1Pin = 25;
const int led2Pin = 13;
int ledState = LOW;
int ledBrightness = 0;

const char html[] PROGMEM = R"rawliteral(
<!DOCTYPE html>
<html>
<head>
  <title>ESP32 Web Server</title>
  <meta name="viewport" content="width=device-width, initial-scale=1" />
  <style>
    body { font-family: Arial, sans-serif; text-align: center; }
    h1 { font-size: 2em; margin-bottom: 10px; }
    p { font-size: 1.2em; margin-bottom: 10px; }
    .button { font-size: 1em; padding: 5px 10px; border: 1px solid black; border-radius: 5px; margin: 10px 0; }
    .button:active { background-color: #ccc; }
  </style>
)</head>
<body>
  <h1>ESP32 Web Server</h1>
  <p>This is a simple web server running on an ESP32. It has two LEDs controlled by buttons on the board. You can change the brightness of the LEDs and toggle them on/off using the buttons and the slider on the left.</p>
  <div>
    <input type="button" value="LED 1 On" class="button" onclick="document.getElementById('led1').style.display = 'block'; document.getElementById('led2').style.display = 'block';" />
    <input type="button" value="LED 1 Off" class="button" onclick="document.getElementById('led1').style.display = 'none'; document.getElementById('led2').style.display = 'block';" />
    <input type="button" value="LED 2 On" class="button" onclick="document.getElementById('led1').style.display = 'block'; document.getElementById('led2').style.display = 'none';" />
    <input type="button" value="LED 2 Off" class="button" onclick="document.getElementById('led1').style.display = 'none'; document.getElementById('led2').style.display = 'none';" />
    <input type="range" min="0" max="100" value="50" class="button" onclick="document.getElementById('led1').style.display = 'block'; document.getElementById('led2').style.display = 'block';" />
  </div>
  <div>
    <div>LED 1</div>
    <div id="led1" style="display: none; width: 10px; height: 10px; background-color: red; border-radius: 50%; margin: 10px auto; border: 1px solid black; position: relative; z-index: 1;></div>
    <div>LED 2</div>
    <div id="led2" style="display: none; width: 10px; height: 10px; background-color: red; border-radius: 50%; margin: 10px auto; border: 1px solid black; position: relative; z-index: 1;></div>
  </div>
</body>
)
```

```

        font-family: sans-serif;
    }
    .container {
        display: flex;
        flex-direction: column;
        align-items: center;
    }
    button {
        font-size: 1.5em;
        margin: 10px;
        padding: 10px;
        background-color: lightgray; /* Set the initial background color */
    }
    button.on {
        background-color: green; /* Set the background color when the LED is ON */
    }
    input[type=range] {
        width: 50%;
    }
</style>
</head>
<body>
    <div class="container">
        <h1>ESP32 Web Server</h1>
        <br>
        <h3>LED1 State Control</h3>
        <button id="button" onclick="toggleLED()">LED: OFF</button>
        <br>
        <br>
        <h3>LED2 Brightness Control</h3>
        <input type="range" id="trackbar" min="0" max="255" step="1" value="0" oninput="setBrightness()">
    </div>
    <script>
        var ledState = false;
        var ledBrightness = 0;
        var button = document.getElementById('button');
        var trackbar = document.getElementById('trackbar');
        function toggleLED() {
            ledState = !ledState;
            var xhr = new XMLHttpRequest();
            xhr.open('GET', '/led?state=' + (ledState ? '1' : '0'), true);
            xhr.send();
            button.innerHTML = 'LED: ' + (ledState ? 'ON' : 'OFF');
            button.classList.toggle('on', ledState); // Add or remove the 'on' class based on the LED state
        }
        function setBrightness() {
            ledBrightness = trackbar.value;
            var xhr = new XMLHttpRequest();
            xhr.open('GET', '/brightness?value=' + ledBrightness, true);
            xhr.send();
        }
    </script>
</body>
</html>
)rawliteral";
}

void handleRoot() {
    server.send(200, "text/html", html);
}

void handleLED() {
    ledState = server.arg("state").toInt();
    digitalWrite(led1Pin, ledState);
    server.send(200, "text/plain", String(ledState));
}

void handleBrightness() {
    ledBrightness = server.arg("value").toInt();
    analogWrite(led2Pin, ledBrightness);
}

```

```

void setup() {
  Serial.begin(115200);
  pinMode(led1Pin, OUTPUT);
  pinMode(led2Pin, OUTPUT);
  digitalWrite(led1Pin, ledState);
  analogWrite(led2Pin, ledBrightness);
  // WiFi AP Configuration & Init
  Serial.println("Setting Up ESP32 AP Network..");
  WiFi.softAP(ssid, password);
  Serial.print("ESP32 AP IP: ");
  Serial.println(WiFi.softAPIP());
  // Attach Server Handler Functions & Start WebServer
  server.on("/", handleRoot);
  server.on("/led", handleLED);
  server.on("/brightness", handleBrightness);
  server.begin();
}

void loop() {
  server.handleClient();
}

```

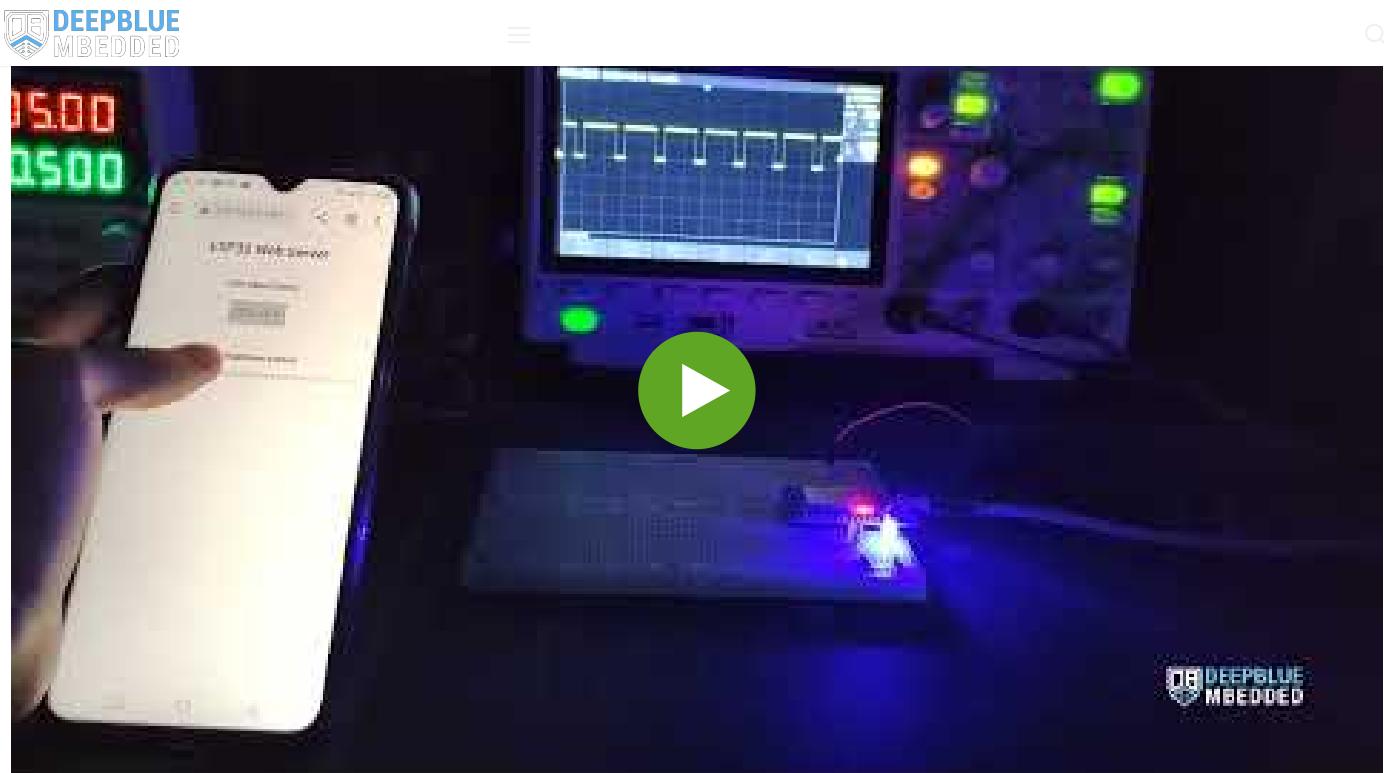
El código para este ejemplo es exactamente el mismo que el del ejemplo anterior (modo estación). Sin embargo, la única diferencia es la configuración del modo WiFi del ESP32 para que sea AP (punto de acceso) en lugar de STA en el proyecto de ejemplo anterior. Y las credenciales de red WiFi para nuestro AP ESP32 serán:

- **SSID** : **ESP32 - LED - AP**
- **Contraseña WiFi** : **01234567**

Y después de ejecutar este proyecto de ejemplo en mi placa ESP32, pude obtener el siguiente mensaje en el monitor serial.

Luego, utilicé mi teléfono inteligente para buscar la red WiFi "**ESP32 - LED - AP**" y proporcioné su contraseña "**01234567**". Y pude conectarme a la red WiFi ESP32.

El siguiente paso fue probar la aplicación visitando la dirección IP del servidor web ESP32 (**192.168.4.1**) en mi navegador. El resultado de la prueba fue exactamente el mismo que el del ejemplo anterior.



Protocolos WiFi ESP32

El ESP32 admite varios protocolos de comunicación a través de WiFi y otras tecnologías de comunicación inalámbrica como BLE, LoRa y GSM. En esta sección, nos centraremos únicamente en obtener una visión general de los **protocolos WiFi del ESP32** que necesita conocer para tener una idea clara de las capacidades WiFi del ESP32 y también para tomar una decisión guiada sobre si utilizar un protocolo específico para una aplicación específica suya o no.

Esta será una lista exhaustiva hasta cierto punto, pero resumida para mantener la extensión de este tutorial manejable y aún así brindarle suficientes detalles para obtener una visión general de qué protocolos se pueden usar con ESP32 WiFi en general.

HTTP ESP32



HTTP (**Hypertext Transfer Protocol**) es un protocolo utilizado para la comunicación de datos a través de Internet. El ESP32 proporciona una pila WiFi integrada que nos permite conectarnos a una red WiFi y enviar solicitudes HTTP. Las solicitudes HTTP pueden ser de dos tipos: GET y POST.

Las solicitudes HTTP GET se utilizan para recuperar datos de un servidor web. En las solicitudes **GET** , los datos se envían en la URL de la solicitud. Por otro lado, las solicitudes **HTTP POST se utilizan para enviar datos a un servidor web**. En las solicitudes **POST** , los datos se envían en el cuerpo de la solicitud.

Podemos utilizar la biblioteca `HTTPClient.h` para enviar solicitudes HTTP con ESP32. La función `http.begin()` se utiliza para especificar la URL de la solicitud. Luego podemos utilizar las funciones `la solicitud es exitosa` , podemos `obtener la respuesta` utilizando la función `http.getString()` .

ESP32 HTTPS



HTTPS es una versión segura de HTTP, lo que significa que todos los datos que se envían entre el cliente y el servidor están cifrados. Esto es importante para proteger datos confidenciales, como contraseñas, números de tarjetas de crédito y otra información personal.

Para realizar una solicitud HTTPS con el ESP32, primero debe asegurarse de tener los certificados necesarios instalados en su dispositivo. Puede obtener estos certificados de una autoridad de certificación (CA) de confianza o generarlos usted mismo mediante OpenSSL.

Una vez que tenga los certificados, puede usar la biblioteca `WiFiClientSecure` . [h](#) incorporada del ESP32 para realizar solicitudes HTTPS. Tenga en cuenta que las solicitudes HTTPS pueden tardar más que las solicitudes HTTP debido a la sobrecarga de cifrado. Además, asegúrese de usar el número de puerto correcto (443 para HTTPS) al conectarse al servidor.

ESP32-UDP



UDP (**User Datagram Protocol**) es un protocolo sin conexión que permite a los dispositivos enviar y recibir datos a través de una red sin establecer una conexión dedicada. Esto lo hace adecuado para aplicaciones donde la comunicación de baja latencia es importante, como sistemas de control en tiempo real o redes de sensores.

Pero es menos confiable que otros protocolos de comunicación, ya que no ofrece ninguna garantía de que los datos que estás enviando sean bien recibidos en el otro extremo. Simplemente envía los datos lo más rápido posible y se olvida de ellos. Para usar UDP con ESP32, puedes usar la biblioteca `WiFiUDP` . [h](#) incorporada o la biblioteca `AsyncUDP` para operación asincrónica.

El protocolo UDP es una forma sencilla y eficaz de enviar y recibir datos a través de una red, pero no ofrece ninguna verificación de errores ni control de flujo. Para aplicaciones más avanzadas, es posible que desee considerar el uso de un protocolo que proporcione estas funciones, como TCP o MQTT.

ESP32 TCP



TCP (**Transmission Control Protocol**) es un protocolo orientado a la conexión que permite la entrega ordenada y confiable de un flujo de bytes entre aplicaciones que se ejecutan en hosts. ESP32 puede actuar como cliente TCP o servidor TCP.

Para utilizar el servidor TCP ESP32, debe crear un socket TCP y luego conectarse al servidor remoto. Una vez que se establece la conexión, puede enviar y recibir datos a través del socket. El servidor TCP ESP32 funciona de manera similar. Crea un socket de servidor TCP y luego espera a que los clientes se conecten. Una vez que un cliente se conecta, puede enviar y recibir datos a través del socket.

TCP es un protocolo confiable y ordenado, lo que significa que los datos se entregan en el orden en que fueron enviados y los paquetes perdidos se retransmiten. Esto hace que TCP sea una buena opción para aplicaciones que requieren transmisiones de datos confiables, como transferencias de archivos o control remoto de dispositivos. Sin embargo, TCP tiene una sobrecarga mucho mayor en comparación con UDP, lo que significa que puede ser mucho más lento.

WebSocket es un protocolo que proporciona comunicación full-duplex a través de una conexión TCP. Permite la comunicación bidireccional entre el cliente y el servidor en tiempo real.

Una de las principales ventajas de utilizar WebSocket es que permite crear un canal de comunicación en tiempo real que puede enviar datos desde el servidor al cliente sin necesidad de que el cliente consulte constantemente al servidor para obtener actualizaciones. Esto puede suponer una mejora significativa de la eficiencia y la capacidad de respuesta de la aplicación.

Dado que el protocolo WebSocket está basado en TCP, necesitarás instalar una biblioteca TCP como AsyncTCP.h para ESP32. De esta manera, podrás crear aplicaciones de servidor web basadas en WebSocket. En futuros tutoriales, veremos algunos proyectos de ejemplo basados en este protocolo.

SMTP ESP32



SMTP (**Simple Mail Transfer Protocol**) es un protocolo que se utiliza para enviar correos electrónicos. Este protocolo se utiliza ampliamente para enviar correos electrónicos por parte de aplicaciones y dispositivos, como sistemas de seguridad, sistemas de monitoreo y más. Al utilizar SMTP, el ESP32 puede enviar correos electrónicos a una dirección de correo electrónico específica cuando ocurre un determinado evento o cuando se cumplen condiciones específicas.

Para poder utilizar SMTP con ESP32, necesitamos tener acceso a un servidor SMTP. El servidor SMTP puede ser un servidor local o un servidor remoto. Una vez que tengamos acceso a un servidor SMTP, podremos utilizar el WiFi del ESP32 para conectarnos al servidor y enviar correos electrónicos.

Para enviar un correo electrónico utilizando el protocolo ESP32 y SMTP, debemos seguir estos pasos:

1. Conéctese a la red WiFi utilizando la biblioteca WiFi.
2. Conéctese al servidor SMTP utilizando la pila TCP/IP.
3. Autenticarse con el servidor SMTP si es necesario.
4. Envíe un correo electrónico enviando el mensaje de correo electrónico al servidor SMTP.

El protocolo SMTP se puede utilizar para diversas aplicaciones, como el envío de notificaciones por correo electrónico o alertas desde dispositivos IoT basados en ESP32. Sin duda, dedicaremos un par de tutoriales futuros a explorar la funcionalidad y las aplicaciones de este protocolo.

ESP32 MQTT



MQTT (**Message Queuing Telemetry Transport**) es un protocolo de mensajería ligero que se utiliza en aplicaciones de IoT para intercambiar mensajes entre dispositivos. Es un protocolo de publicación/suscripción diseñado para usarse con redes poco confiables y de bajo ancho de banda. MQTT es compatible con ESP32 y se puede utilizar para la comunicación entre dispositivos ESP32 y otros dispositivos o servidores habilitados para MQTT.

MQTT funciona con el concepto de un **intermediario** , que actúa como un centro neurálgico para todos los dispositivos conectados. Los dispositivos pueden publicar mensajes en un tema específico del intermediario, y otros dispositivos pueden suscribirse a ese tema para recibir los mensajes. MQTT admite tres niveles de calidad de servicio (QoS) para garantizar una entrega confiable de los mensajes.

Es importante tener en cuenta que MQTT es un protocolo cliente-servidor y requiere un agente para funcionar. Hay muchos agentes MQTT públicos disponibles que se pueden usar para pruebas y desarrollo, pero para uso en producción, se recomienda configurar un agente MQTT privado.

Existe una biblioteca ESP32 para el protocolo MQTT llamada “ **PubSubClient** ”. Se puede usar para conectarse a intermediarios MQTT y publicar o suscribirse a temas en sus aplicaciones de IoT. Además, realizaremos algunos proyectos para practicar el uso del protocolo MQTT en diferentes aplicaciones en futuros tutoriales.

ESP-NOW

ESP-NOW es un protocolo de comunicación peer-to-peer de bajo consumo desarrollado por Espressif, el fabricante de microcontroladores ESP32. Permite una comunicación sencilla, rápida y fiable entre dos o más módulos ESP32 sin necesidad de una red Wi-Fi.

ESP-NOW opera en la capa MAC y se puede utilizar para enviar datos entre dispositivos ESP32 directamente, sin necesidad de un enrutador o punto de acceso. Esto lo hace ideal para aplicaciones donde se requiere un bajo consumo de energía y una comunicación simple y directa.

ESP-NOW es particularmente útil en aplicaciones como la automatización del hogar, donde varios dispositivos necesitan comunicarse entre sí de manera descentralizada. Se puede utilizar para construir una red de dispositivos que puedan comunicarse entre sí utilizando muy poca energía, lo que lo hace ideal para dispositivos alimentados por batería. Las características de ESP-NOW incluyen:

- Protocolo de comunicación rápida
- Comunicación energéticamente eficiente
- Hasta 250 bytes de carga útil en cada paquete
- La comunicación puede ser cifrada o no
- ESP-NOW puede lograr demasiadas topologías de red
- Alcance anunciado de hasta 1 km (los usuarios informan alcances entre 200 y 500 metros en la práctica)

Para utilizar ESP-NOW, primero debe obtener la biblioteca `esp_now.h` e inicializar el protocolo en cada dispositivo llamando `a esp_now_init()`. Luego, debe registrar cada dispositivo con el protocolo llamando a `esp_now_add_peer()`. Esto le indica al ESP32 con qué dispositivos se puede comunicar.

Una vez que se hayan registrado los dispositivos, puede utilizar `esp_now_send()` para enviar datos de un dispositivo a otro. Los datos se envían en paquetes y cada paquete puede contener hasta 250 bytes de datos.

ESP-MESH con ESP32

ESP-MESH

ESP-MESH es un protocolo de red desarrollado por Espressif Systems, el fabricante del microcontrolador ESP32. Permite la creación de redes en malla que pueden conectar múltiples nodos ESP32, lo que permite una comunicación fiable y robusta entre ellos.

Las redes en malla son particularmente útiles en situaciones en las que no es posible realizar conexiones punto a punto tradicionales, como en edificios grandes o áreas al aire libre. Con ESP-MESH, los nodos pueden comunicarse entre sí de manera descentralizada, lo que permite una topología de red dinámica y una fácil escalabilidad.

Algunas de las características clave de ESP-MESH incluyen:

- Formación automática de topología: los nodos ESP-MESH pueden descubrirse y conectarse entre sí automáticamente, formando una red en malla sin necesidad de configuración manual.
- Autocuración: si un nodo de la red falla o se desconecta, los demás nodos pueden redirigir automáticamente su comunicación para mantener la conectividad.
- Selección de padre: los nodos ESP-MESH pueden seleccionar dinámicamente el nodo padre más adecuado para conectarse, en función de factores como la intensidad de la señal y la carga de la red.
- Seguridad: ESP-MESH admite varias funciones de seguridad, como cifrado, autenticación y control de acceso, para garantizar que solo los dispositivos autorizados puedan unirse a la red.

Con ESP-MESH, se pueden crear una amplia gama de aplicaciones, como la automatización del hogar, la monitorización industrial y la agricultura inteligente. El protocolo también es compatible con otros protocolos de redes en malla estándar, como Zigbee y Thread, lo que permite la interoperabilidad con otros dispositivos y sistemas.

ESP-TOUCH

El protocolo ESP-TOUCH es una tecnología desarrollada por Espressif que simplifica el proceso de configuración de ESP32 y otros dispositivos con ESP8266EX integrado para conectarse a una red Wi-Fi. Está diseñado para que el proceso de configuración sea lo más sencillo posible para los usuarios, permitiéndoles conectar sus dispositivos con tan solo unos sencillos pasos mediante un teléfono inteligente.

El protocolo funciona enviando un paquete de configuración de Wi-Fi desde un teléfono inteligente al dispositivo ESP32 o ESP8266EX integrado. El dispositivo recibe el paquete y utiliza la información para conectarse a la red Wi-Fi especificada.

Una de las principales ventajas de ESP-TOUCH es que elimina la necesidad de que el usuario introduzca manualmente el nombre y la contraseña de la red Wi-Fi en el dispositivo. En su lugar, el protocolo gestiona todo el proceso de configuración de forma automática, lo que ahorra tiempo y esfuerzo al usuario.

El protocolo ESP-TOUCH también es muy seguro, ya que utiliza algoritmos de cifrado avanzados para proteger el nombre y la contraseña de la red Wi-Fi durante la transmisión. Esto garantiza que la información se mantenga a salvo de accesos no autorizados o interceptaciones.

ESP32 ESP-Modbus



Modbus es un protocolo de comunicación ampliamente utilizado en sistemas de control industrial para conectar dispositivos electrónicos. Modbus es un protocolo de solicitud-respuesta que opera sobre diferentes capas de comunicación física, como RS-232, RS-485, WiFi, Ethernet y otras. El protocolo Modbus incluye varios códigos de función, cada uno con un propósito específico, como leer y escribir datos, controlar dispositivos y más.

Existe una biblioteca oficial de Espressif llamada ESP-Modbus que agrega soporte para este protocolo en microcontroladores ESP32 para aplicaciones industriales.

ESP32 mDNS (DNS de multidifusión)

mDNS (**Multicast DNS**) es un servicio UDP de multidifusión que permite que los dispositivos de una red se detecten y se comuniquen entre sí mediante nombres de dominio en lugar de direcciones IP. Es particularmente útil en situaciones en las que las direcciones IP de los dispositivos de la red cambian constantemente o son desconocidas.

mDNS funciona mediante el uso de paquetes DNS de multidifusión para transmitir consultas y respuestas en la red local. Cuando un dispositivo desea descubrir otro dispositivo en la red, envía una consulta mDNS para obtener el nombre de host del dispositivo. El dispositivo con el nombre de host coincidente responde con su dirección IP, lo que permite que el dispositivo que realiza la consulta establezca una conexión.

E1 ESP32 admite mDNS de fábrica a través de la biblioteca incorporada “ `ESPMDNS.h` ” . Para usar mDNS en el ESP32, simplemente incluya la biblioteca en su código e inicialícela con un nombre de host único. Los ejemplos provistos en la biblioteca Arduino-Core del ESP32 deberían ser suficientes para ayudarlo a comenzar.

ESP32 WiFi Otras aplicaciones

El WiFi del ESP32 no solo permite muchas aplicaciones de IoT, diferentes protocolos de comunicación y varias topologías de red, sino que también nos permite crear aplicaciones más interesantes. En esta sección, repasaremos una breve descripción de varias aplicaciones adicionales que también son posibles gracias a las capacidades WiFi del ESP32.

Una de las grandes ventajas del ESP32 es su compatibilidad con actualizaciones por aire (OTA), lo que le permite actualizar su código de forma remota, sin tener que conectar físicamente su ESP32 a una computadora. Esto puede ser particularmente útil si su ESP32 está instalado en una ubicación remota o si tiene muchos ESP32 que necesitan una actualización de firmware.

Existen varias formas de realizar actualizaciones OTA en el ESP32, pero el método más común es utilizar la biblioteca [ArduinoOTA](#). Esta biblioteca está incluida en el Arduino Core del ESP32 y proporciona una forma sencilla y cómoda de realizar actualizaciones OTA para el ESP32.

Sincronización horaria NTP ESP32

NTP (**Network Time Protocol**) es un protocolo popular que se utiliza para la sincronización horaria en redes. El ESP32 puede obtener fácilmente la hora de un servidor NTP mediante la biblioteca de hora incorporada.

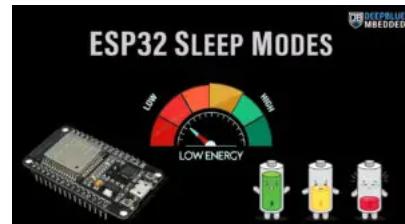
Una vez que el ESP32 obtiene la hora del servidor NTP, se puede utilizar para sincronizar el reloj interno del dispositivo. Esto también se puede hacer utilizando la biblioteca de hora incorporada.

Ahorro de energía WiFi ESP32

El ESP32 ofrece distintos modos de ahorro de energía entre los que se puede alternar para alcanzar un determinado consumo energético general. La mayoría de los dispositivos IoT consideran que la energía de la batería es un recurso fundamental y es necesario maximizar su vida útil. Puede resultar tedioso tener que ir al campo donde está instalado el nodo ESP32 y recargar la batería manualmente con mucha frecuencia.

Consulte el siguiente tutorial para obtener más información sobre los modos de ahorro de energía ESP32 y las posibles aplicaciones de cada modo.

???? Lea también



Modos de suspensión del ESP32 (modos de ahorro de energía)

Este artículo le brindará información más detallada sobre los modos de ahorro de energía del ESP32, los modos de suspensión, su consumo de energía y las posibles aplicaciones de cada modo.

Solución de problemas de WiFi del ESP32

Es posible que surjan algunos problemas al probar ejemplos de WiFi ESP32 y que resulte muy frustrante. La mayoría de los problemas de conexión/desconexión de WiFi y sus soluciones se encuentran en el artículo vinculado a continuación. Seguiré agregando más problemas comunes y sus soluciones en función de sus comentarios y experiencias.

- **Conexión WiFi ESP32, posibles problemas y soluciones**

Lista de piezas

Aquí se incluye la lista completa de componentes para todas las piezas que necesitarás para realizar los laboratorios prácticos mencionados en este artículo y para toda la serie de tutoriales de ESP32 que se encuentran aquí en DeepBlueMbedded. Ten en cuenta