Universidad Nacional de Río Negro, Sede Andina

Carrera:	Materia:	Com.:
Ingeniería en las Telecomunicaciones	**ARQUITECTURA DE COMPUTADORAS Y SISTEMAS EMBEBIDOS**	**1**
Título: **ENCENDIDO DE LUCES POR WI-FI**		

Preparado por
Nombre Aparicio Guillermo
Fecha 21/06/2025

OBSERVACIONES:
Propiedad de título en “Archivo/propiedades/propiedades avanzadas”. Para actualizar los campos automáticos. Presionar “Ctrl + E” y luego “F9”. Hacer doble clic en el pie de página, presionar “Ctrl + E” y luego “F9”. Hacer doble clic fuera del pie de página. Hacer doble clic en encabezado, presionar “Ctrl + E” y luego “F9”. El texto resaltado en amarillo es de orientación borrarlo antes de enviar la versión final. El texto en verde actualizar manualmente.

Nota:

REVISIONES

REV	Fecha	Página Número / Cambios	AUTOR	REVISÓ	APROBÓ
A	dd/mm/aaaa	Original	Alumno 1, N.	Alumno 2, N.	Profesor, N.

INDICE

OBJETIVOS

Implementar un prototipo IoT basado en ESP32 que controle luces de forma remota mediante WiFi, garantizando bajo consumo energético y respuesta en tiempo real, con potencial para escalarse a sistemas de domótica más complejos.

Objetivos Específicos:

1. Programar el ESP32 para conectarse a una red WiFi y recibir comandos de encendido/apagado mediante un protocolo de comunicación (MQTT/HTTP).
2. Crear una interfaz de usuario sencilla (página web o plataforma IoT) para enviar comandos al sistema.
3. Documentar el proceso de desarrollo para permitir su replicación o mejora en futuras versiones.

ALCANCE DEL PROYECTO

Sistema de Control Remoto de Iluminación mediante WiFi con ESP32

Este proyecto tiene como finalidad desarrollar un **prototipo funcional de domótica básica**, utilizando un microcontrolador ESP32s para establecer las bases de un sistema escalable de control de iluminación. Aunque el caso de prueba inicial utiliza un único LED, el diseño contempla principios técnicos que permitirían su expansión a sistemas más complejos.

Componentes principales incluidos:

1. **Módulo de control central**
    - Implementación con ESP32 (con capacidades WiFi integradas)
    - Programación en Arduino IDE, usando librerías específicas para manejo de GPIO y conexión WiFi
2. **Interfaz de usuario**
    - Servidor web embebido en el ESP32 para control desde navegador
    - Protocolo de comunicación HTTP/MQTT para transmisión de comandos
3. **Módulo de potencia (prueba conceptual)**
    - Circuito de control para LED de prueba con resistencia limitadora de 220 ohm.

4. **Arquitectura de red**

- Conexión a redes WiFi 2.4GHz con autenticación WPA2
- Sistema direccionable dentro de red local (posibilidad de implementar DDNS)

Limitaciones (enfoque pedagógico):

El prototipo utiliza **un LED como carga de prueba**, pero el diseño eléctrico considera parámetros para escalar a relés. La interfaz es **local** por ahora, pero la arquitectura permite integración con servicios en la nube. **No incluye** elementos de automatización avanzada (pero sienta las bases para ellos)

Perspectiva de crecimiento:

Este desarrollo constituye la **primera fase** de un sistema más amplio, donde:

- El LED representa la validación conceptual
- La estructura de código permite añadir más dispositivos
- La electrónica demostrada es aplicable a circuitos de mayor potencia

MARCO TEORICO

TECNOLOGÍA WIFI Y PROTOCOLOS DE COMUNICACIÓN

El WiFi (IEEE 802.11) es una tecnología de red inalámbrica ampliamente utilizada para la transmisión de datos en redes locales. En este proyecto se emplea el estándar **2.4 GHz** (protocolos b/g/n), ideal para aplicaciones IoT por su balance entre alcance y consumo energético. Los protocolos clave implementados incluyen:

4. **HTTP**: Para la comunicación cliente-servidor en la interfaz web embebida.
5. **TCP/IP**: Base de la pila de protocolos para garantizar entregas confiables de paquetes.

MICROCONTROLADOR ESP32: CARACTERÍSTICAS TÉCNICAS

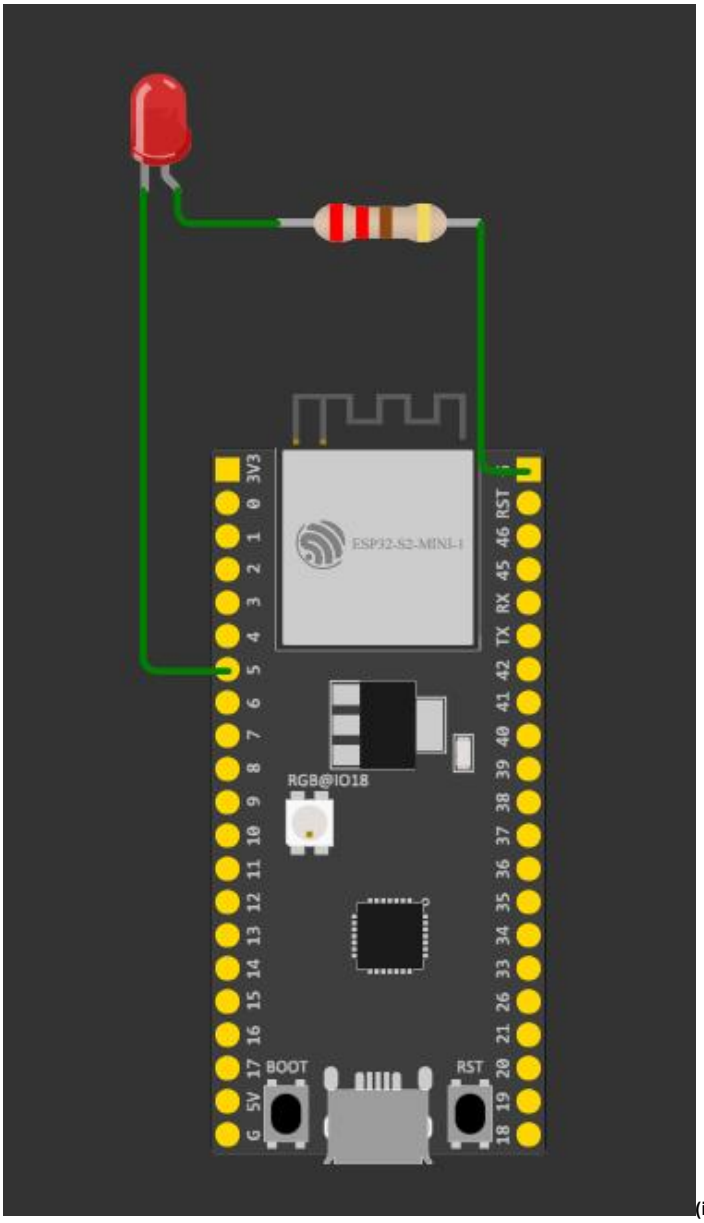
El ESP32 es un sistema en chip (SoC) con doble núcleo Xtensa de 32 bits, destacado por:

- **Conectividad integrada**: WiFi 2.4 GHz y Bluetooth 4.2.
- **E/S versátiles**: 34 pines GPIO, ADC de 12 bits, interfaces SPI/I2C/UART.
- **Bajo consumo**: Múltiples modos de ahorro energético (Deep Sleep: ~10µA).

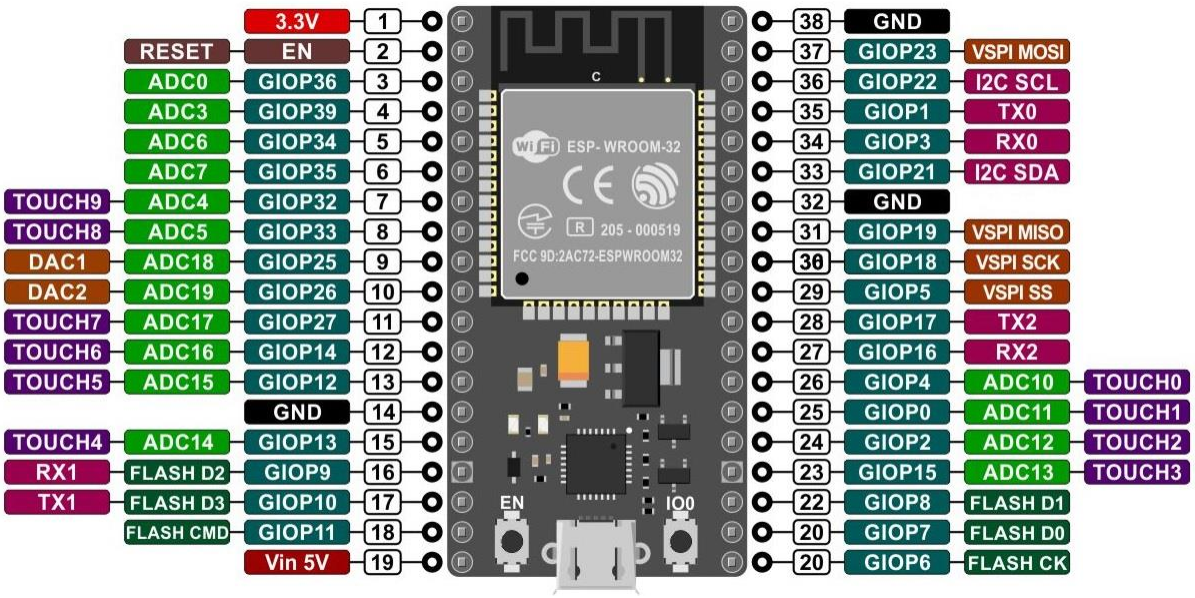
- **Memoria**: 520 KB SRAM, 4MB flash (en la mayoría de modelos).
- **Librerías compatibles**: Arduino Core para ESP32, FreeRTOS para multitarea.

DESARROLLO DEL PROYECTO

Diagrama del circuito eléctrico



(imagen a ejemplo: “GPIO5 lugar supuesto”)



El circuito consta de:

- **ESP32** (pin GPIO5 → Resistencia 220Ω → LED → GND)
- **Alimentación**: USB 5V o fuente externa
- **Protección**: Resistencia limitadora para el LED

Listado de componentes utilizados

componente	Especificaciones
Modulo ESP32s nodeMCU	WiFi 2.4GHz, 38 pines GPIO
LED rojo	2v 20mA
Resistencia	220 Ω
Protoboard	400 puntos
Cable dupoint x2	Macho - Macho

Programación del ESP32

- Inclusión de librerías y declaración de variables

```
1   //Importa la librería para manejar conexiones WiFi
2   #include <WiFi.h>
3
4   //Almacenan las credenciales de la red WiFi
5   const char* ssid = "nombreDeWifi"; //clave wifi
6   const char* password = "contraseñla";  //contradeseña
7
8   //Asigna el pin GPIO5 para controlar el LED
9   const int ledPin = 5;
10
11  //Crea un servidor web en el puerto 80 (HTTP)
12  WiFiServer server(80); //se almacena el servidor en el puerto 80
13
```

El ***nombreDeWifi*** y ***contraseña*** van a modificarse por la respectiva a la red en donde habilitada.

- Inicialización de puerto serial

```
14  void setup() {
15    Serial.begin(115200); //Inicia la comunicación serial a 115200 baudios para poder enviar mensajes de depuración al monitor serial del IDE Arduino.
16
```

- Configuración del pin LED

```
16
17    pinMode(ledPin, OUTPUT);//Establece el pin del LED (definido previamente como GPIO5) como salida digital.
18    digitalWrite(ledPin, LOW);//Pone el pin en estado LOW (0V) para asegurar que el LED comience apagado.
19
```

- Conexión WiFI

```
20    WiFi.begin(ssid, password);//Intenta conectar a la red WiFi usando las credenciales (SSID y contraseña) predefinidas.
21    Serial.print("Conectando a WiFi");//Muestra "Conectando a WiFi" en el monitor serial.
22    while (WiFi.status() != WL_CONNECTED) { //Bucle while que la conexion WiFi se establezca correctamente
23      delay(1000);//Espera 1 segundo (delay(1000))
24      Serial.print(".");//Muestra puntos ".." para indicar progreso
25    }
26
```

- Confirmación de WiFi

```
26
27    Serial.println("\nConectado a WiFi");//Mensaje de confirmación ("Conectado a WiFi")
28    Serial.println("IP local: ");
29    Serial.println(WiFi.localIP());//La dirección IP local asignada al ESP32 en la red
30
```

localIP es una función de WiFi por su librería en donde devuelve el IP asignado.

- Inicio del servidor web

```
31    server.begin();//Inicia el servidor en el puerto 80
32    if (WiFi.status() == WL_CONNECTED) {
33    Serial.println("Servidor iniciado."); //mensaje del servido de incio
34    Serial.print("Accedé desde navegador: http://");mensaje indicacitivo sobre de que va acceder
35    Serial.println(WiFi.localIP());//accede desde el navegador
36    }
37  }
38
```

Se verifica el estado de la conexión, procediendo a mostrar los mensajes de la IP a la que va acceder.

Interfaz de usuario (app/web)

- **Detección de Clientes**

```
39  void loop() {
40    //Verifica si hay un usuario intentando conectarse al servidor web del ESP32s
41    WiFiClient client = server.available();//detecta conexion entrande en el port80
42    if (client) {//si available devolvio un objeto WiFiClient=conexion enlazada
43      Serial.println("Cliente conectado");//se imprime cliente conectado
44
```

En la linea 44 ***available()*** es un método del objeto ***server*** que chequea la conexión entrante en el puerto 80(HTTP). Si detecta conexión crea un objeto ***WiFiClient*** llamado ***client***.

El ***if()*** evalúa si ***client*** contiene un cliente valido, en ese caso se confirma por Monitor Serial del IDE Arduino la conexión entrante que detecto el ESP32.

- Lectura de petición a conexion

```
44
45    String request = ""; //declaracion de variable tipo string para LED=ON/OFF
46    unsigned long timeout = millis();//tiempo maximo de espera
47
```

Se crea una variable vacia (” ”) llamada request del tipo String. Aqui se almacenaran comandos como /LED=ON o /LED=OFF que viene de la URL.

timeout establece un tiempo máximo de espera para evitar que el ESP32 se quede bloqueado si el cliente no envía datos completos, en donde **millis()** devuelve el tiempo actual en miliseguindos desde que el ESP32 se encendió.

- Lectura de la Petición HTTP.

```
47
48    //verifica si user sigue conectador comprobando que haya pasado 1 segundo
49    while (client.connected() && millis() - timeout < 1000) {
50      if (client.available()) {//verifica si hay datos disponibles del cliente
51        request = client.readStringUntil('\r');//lee todo hasta encontrar '/r'
52        Serial.println("Petición: " + request);//Imprime peticion del monitor serial
53        break;//salida del bule
54      }
55    }
56
```

La condición verifica si el cliente sigue conectado (***client.connected()***), mientras que la segunda condición comprueba que no haya pasado más de 1 segundo.

En el ciclo se verifica la disponibilidad de datos del cliente procediendo a leer todos los caracteres hasta encontrar un retorno indicado como '\r'. Cuando la totalidad de datos se recibe procede a imprimirlo por monitor y luego sale del bucle.

- Limpiador del buffer de escritura del cliente WiFi.

```
55      }
56      //limpia el buffer del cliente
57      client.flush();
58
```

Rol critico en el manejo de conexiones HTTP. Antes de responder asegura la linea ante ruidos, ecos, y mezclas de peticiones/respuestas, evitando que se pierdan paquetes de comunicaciones o se fragmente la respuesta HTTP.

- Procesamiento de comandos

```
58
59      // Comandos LED
60      if (request.indexOf("/LED=ON") != -1) {
61        digitalWrite(ledPin, HIGH);//Enciende el LED
62      } else if (request.indexOf("/LED=OFF") != -1) {
63        digitalWrite(ledPin, LOW);//Apaga el LED
64      }
65
```

Busca encontrar en la cadena request del ESP32 "/LED=ON". Si la condición cumple procede a cambiar el estado del pin 5 (GPIO5 del ESP32s) definido previamente como ***ledpin***. Caso contrario si recibie "LED=OFF" apaga el LED.

- Construcción de la respuesta HTTP (Cabecera)

```
65
66      // Respuesta HTTP
67      client.println("HTTP/1.1 200 OK");//indica que la solicitud fue exitosa
68      client.println("Content-Type: text/html");//indica que el contenido será HTML
69      client.println("Connection: close"); //Evita almacenamiento en caché
70      client.println();
71
```

La versión del protocolo usado es HTTP/1.1 y el código 200 OK es el estandar para éxito. Luego indica al navegador que el contenido será ”html”. Cierra la conexión TCP después de enviar la respuesta, y finaliza con una linea en blanco marcando el fin de la cabecera HTTP, obligatorio del protocolo.
Esto hace posible la comunicación entre el ESP32 y el navegador, transformando la placa en un mini servidor web funcional.

- Envío de la Página Web

```
71
72      // Respuesta HTTP con CSS embebido
73      client.println("<!DOCTYPE html><html><head>");//Inicicia el documento HTML
74      client.println("<title>ESP32 LED</title>");//Muestra ESP32 LED"
75
```

Inicia un documento informando el ingreso de codigo HTML. El título que aparece en el navegador es” ESP32 LED”.

- Estilo CSS (botón encendido - apagado)

```
75
76      //Estilos CSS
77      client.println("<style>");
78      client.println("body { font-family: Arial; text-align: center; background-color: #f0f0f0; }");
79      client.println("h1 { color: #333; }");
80      client.println("button { padding: 15px 30px; font-size: 16px; margin: 10px;");
81      client.println("background-color: #4CAF50; color: white; border: none; border-radius: 8px; cursor: pointer; }");
82      client.println("button:hover { background-color: #45a049; }");
83      client.println("a { text-decoration: none; }");
84
```

El cuerpo es un texto entrado con fuente Arial de fondo gris. El título es de color oscuro (#333).
Se añade el botón de color verde (#4CAF50) con texto blanco, sin bordes ni esquinas, y que cambia a color oscuro(#4CAF50) al pasar el mouse.

- Cuerpo de pagina (Body)

```
84
85      //Cuerpo de la pagina
86      client.println("</style>");
87      client.println("</head><body>");
88      client.println("<h1>Control de LED</h1>");//Titulo control de LED
89      client.println("<a href=\"/LED=ON\"><button>Encender</button></a>");
90      client.println("<a href=\"/LED=OFF\"><button>Apagar</button></a>");
91      client.println("</body></html>");
92
```

Título principal " Control de LED “.

Relaciona el botón con la orden de iniciar el envío a la petición /LED=ON o /LED=OFF respectivamente.

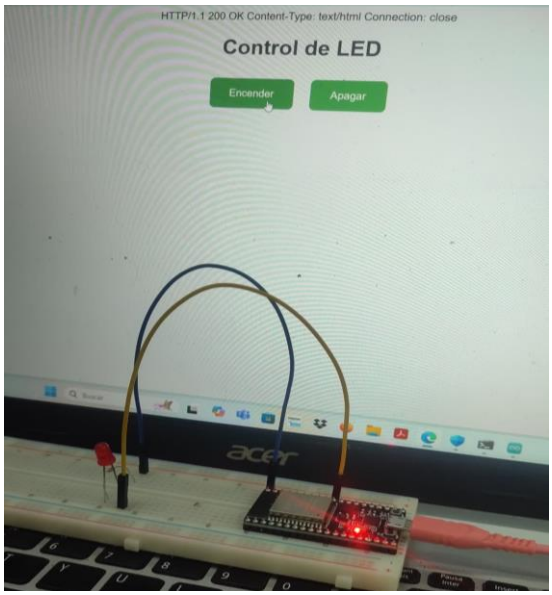
- Fin del programa.

```
92
93        delay(1);
94        client.stop();
95        Serial.println("Cliente desconectado");
96      }
97    }
98
```

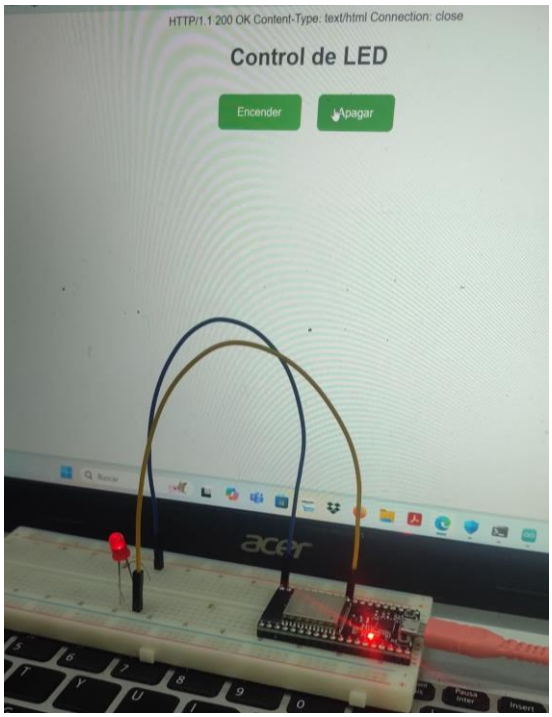
PRUEBAS Y RESULTADOS

Pruebas de conectividad WiFi

- **Led apagado, listo para encender.**



- **LED encendido.**



Pruebas de tiempo de respuesta

CONCLUSIONES

Se logró cumplir con el objetivo principal de conectar un LED, pero más se demostró que el microcontrolador ESP32 desempeña gran conectividad y soporte

superando una conexión Wifi estable y procesamiento HTTP eficiente, obteniendo del proyecto resultados escalables con relés y/o sensores para sistemas automatizados más complejos. Demostrada su versatilidad para un prototipado rápido.