**Leveraging the ESP32's Inbuilt Wi-Fi Hotspot**

**for an IoT-Based Smart Lighting Control System**

# -Swarnim Dahal, Shristi Bastola, Paras Bista, Hari Bashyal

# Abstract

In this project, we designed and implemented an IoT-based smart lighting control system using the ESP32-WROOM-32 microcontroller and a 4-channel relay module. The system enables users to remotely control a light bulb via Wi-Fi, eliminating the need for traditional physical switches and providing a modern, energy-efficient solution for home automation. The ESP32 microcontroller operates in Access Point (AP) mode, creating a local Wi-Fi network that users can connect to using their smartphones, tablets, or computers. A lightweight web server hosted on the ESP32 processes user requests, allowing seamless control of the light bulb through a simple web interface.

The system is designed with scalability in mind, capable of integrating additional sensors and actuators for enhanced functionality. For instance, motion sensors and light sensors can be added to automate lighting based on occupancy and ambient light levels, further improving energy efficiency. The use of the ESP32's inbuilt Wi-Fi hotspot ensures reliable communication within the local network, while its low power consumption makes it an ideal choice for IoT applications.

Testing demonstrated the system's ability to provide real-time control with minimal latency, ensuring a responsive user experience. The project highlights the potential of IoT in transforming traditional home automation systems into smart, interconnected ecosystems. By leveraging the ESP32's capabilities, this project serves as a foundation for developing scalable smart home solutions, with potential future expansions such as internet-based control, integration with voice assistants, and support for multiple devices.

This project not only showcases the practical application of IoT in everyday life but also emphasizes the importance of energy efficiency and user convenience in modern smart home systems. It provides a cost-effective and accessible solution for homeowners and businesses looking to adopt smart lighting technologies, paving the way for more advanced and interconnected IoT-based automation systems.

**Table of Contents**

1. Introduction
2. Objectives
3. System Overview
4. Hardware Components
5. Implementation
6. Testing and Results
7. Challenges
8. Applications and Future Scope
9. Conclusion
10. Appendices

Introduction

The Internet of Things (IoT) revolutionized how we interact with technology, with systems and devices interacting smoothly. IoT integrates sensors, software, and connectivity into regular objects so that they can share and collect information. The technology finds uses in various fields, including healthcare, agriculture, transport, and home automation.

**Smart lighting systems** are a key application of IoT, offering significant advantages over traditional lighting solutions. These systems provide:

1. **Energy Efficiency**: By automating lighting based on occupancy, time of day, or ambient light levels, smart lighting reduces energy consumption.
2. **Convenience**: Users can control lights remotely via smartphones or voice assistants, eliminating the need for physical switches.
3. **Customization**: Smart lighting allows users to adjust brightness, color, and scheduling to suit their preferences.
4. **Cost Savings**: Reduced energy usage translates to lower electricity bills.
5. **Scalability**: Smart lighting systems can be expanded to include other IoT devices, creating a fully interconnected smart home ecosystem.

This project addresses the limitations of traditional and existing smart lighting systems by utilizing the ESP32 microcontroller's inbuilt Wi-Fi hotspot capability. The ESP32 is a low-cost, low-power microcontroller with integrated Wi-Fi and Bluetooth, making it an ideal choice for IoT applications.

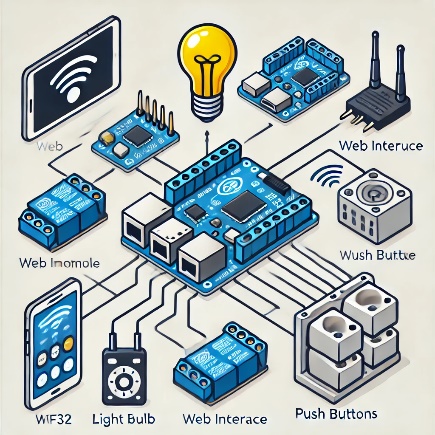
Objective

The primary objective of this project is to design and implement a cost-effective, energy-efficient, and scalable IoT-based smart lighting controlsystem using the ESP32 microcontroller. Specifically, the project aims to:

* + Utilize the ESP32's Access Point (AP) mode to create a local Wi-Fi network, enabling remote control of lighting without dependency on external internet connectivity.
  + Develop a web-based interface hosted on the ESP32 to allow users to control lights remotely via smartphones or computers.
  + Integrate sensors (e.g., motion and light sensors) to automate lighting based on occupancy and ambient light levels, reducing energy consumption.
  + Design the system to be easily expandable for controlling multiple devices or integrating with other IoT applications.
  + Showcase the potential of IoT in home automation by providing a simple, affordable, and user-friendly smart lighting solution.

System Overview

The system enables remote control of electrical appliances using an ESP32 microcontroller, a 4-channel relay module, and a web interface. Users can interact with the system via a web interface or physical push buttons to toggle connected appliances. The ESP32 acts as a central controller, receiving user commands and switching relays accordingly.



**Architecture**

* User Interface: A web-based interface hosted by the ESP32 allows users to send commands.
* ESP32 Microcontroller: Acts as the main controller, receiving user inputs and controlling the relays.
* Relay Module: Connected to the ESP32, responsible for switching electrical loads on and off.
* Electrical Appliances: Devices (e.g., light bulbs) connected to the relay module.
* Push Buttons: Allow manual control of the relays in addition to the web interface.

**Workflow**

The system architecture consists of several key components working together to enable remote and manual control of electrical appliances. At the core of the system is the ESP32 microcontroller, which acts as the main controller, receiving user inputs and managing the relay module. The relay module is directly connected to the ESP32 and is responsible for switching electrical loads on and off. Users can interact with the system through a web-based interface hosted by the ESP32, allowing them to send commands remotely via a smartphone or computer. Additionally, push buttons provide an alternative manual control method, enabling users to toggle the relays without needing internet access. The electrical appliances, such as light bulbs or other devices, are connected to the relay module and respond to commands sent through either the web interface or physical buttons, ensuring flexible and reliable control over the system.

Hardware Components

1. ESP32-WROOM-32

The ESP32-WROOM-32 is a powerful microcontroller module featuring built-in Wi-Fi and Bluetooth, making it ideal for IoT applications. It operates on a dual-core Tensilica LX6 processor, running at up to 240 MHz, with 520 KB of SRAM and support for external flash memory. The ESP32 is known for its low power consumption, offering various sleep modes to optimize energy usage. It supports multiple GPIOs, PWM, ADC, and I2C, allowing easy integration with sensors, relays, and other peripherals.

1. 4-Channel Relay Module

The 4-channel relay module acts as an electrically operated switch, enabling the ESP32 to control high-power electrical loads. Each relay is electromagnetically operated and can handle AC (110V–250V) or DC (up to 30V, 10A per channel). The ESP32 sends low-voltage control signals (3.3V or 5V) to trigger the relays, which in turn switch ON or OFF connected appliances such as light bulbs, fans, or other household devices.

1. Light Bulb

The system uses an LED bulb for demonstration, as LED bulbs are energy-efficient, durable, and generate less heat compared to incandescent or CFL bulbs. A standard 9W or 12W LED bulb operating at 220V AC or 110V AC can be connected through the relay module.

1. Power Supply

The ESP32 requires a 5V DC power supply, which can be provided via a USB adapter, Li-ion battery, or a 5V regulated power source. The relay module often requires 5V or 12V DC, depending on its specifications, so an appropriate power adapter or step-down converter (e.g., buck converter) may be needed to power both the ESP32 and relays.

Implementation

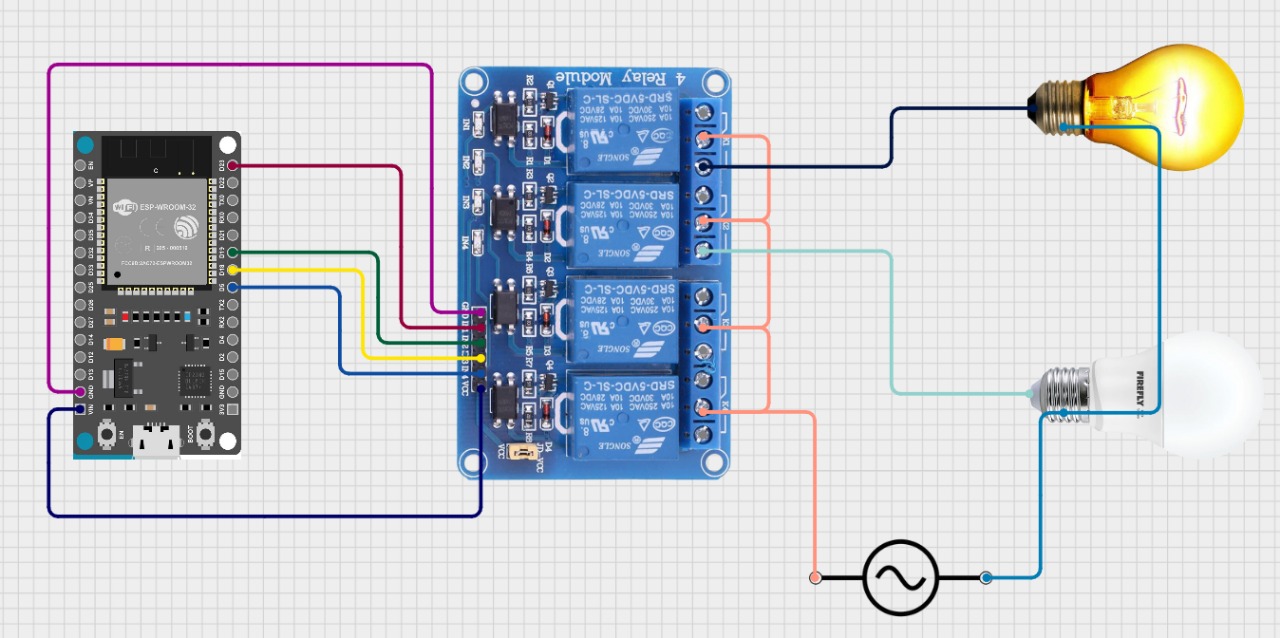


Fig: Circuit Diagram

Setup:

ESP32 to Relay Module Wiring

The ESP32 is connected to the relay module via GPIO pins to control the relays:

VCC (Relay Module) → 5V (ESP32)

GND (Relay Module) → GND (ESP32)

IN1 (Relay Module) → GPIO (ESP32)

IN2 (Relay Module) → GPIO (ESP32)

IN3 (Relay Module) → GPIO (ESP32)

IN4 (Relay Module) → GPIO (ESP32)

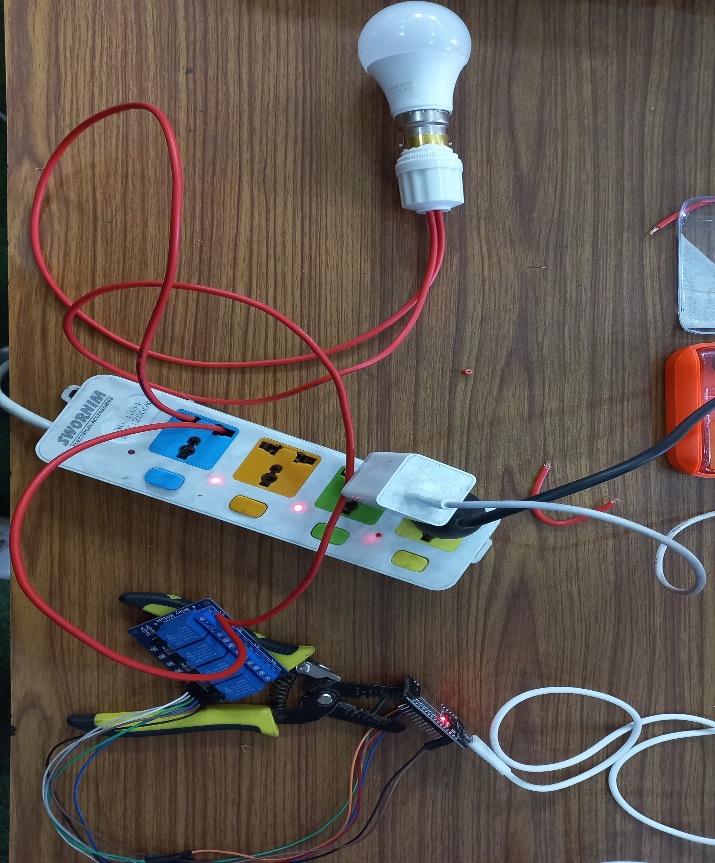
Each INx pin on the relay module corresponds to a specific relay. The ESP32 sends HIGH/LOW signals to these pins to switch the relays ON/OFF.

Relay Module to Light Bulb Wiring

Each relay acts as an electrical switch controlling the live wire of the AC connection:

* Common Terminal (COM): Connected to the AC live wire.
* Normally Open (NO) Terminal: Connected to the live wire of the light bulb.
* Neutral (AC Source) → Neutral (Light Bulb)

When the relay is OFF, the circuit is open, and the bulb remains OFF. When the ESP32 activates the relay, it closes the connection between COM and NO, allowing current to flow and turning the bulb ON.



Testing and Results

To ensure the proper functionality of the ESP32-based relay control system, various tests were conducted. These tests focused on connectivity, response time, and power consumption.

* The ESP32 was configured as both a Wi-Fi access point and a station to check if users could connect seamlessly.
* A smartphone and laptop were used to access the web-based control panel.
* The delay between pressing a button on the web interface and the relay activation was measured.

Challenges

Challenges faced:

* Limited range when using the ESP32’s Access Point (AP) mode.
* Web interface sometimes failed to load or became unresponsive.
* Noticeable delay when switching relays via the web interface.
* Voltage fluctuations led to unexpected relay triggers.

Applications and Future Scope

* Smart Home Automation: Remotely control lights, fans, and appliances using a web interface.
* Office & Commercial Spaces: Automate lights, air conditioning, and security systems to save power and integrate with motion sensors for smart lighting.
* Smart Cities & Infrastructure: Remotely control streetlights, traffic signals, or public utilities.
* Agriculture & Farming: Control water pumps and irrigation systems based on real-time data and automate greenhouse temperature and humidity control.
* Industrial Automation: Manage machinery and production lines remotely and monitor and control HVAC systems in factories.

Future Enhancement:

**Internet-Based Control**

* Integrate IoT cloud platforms (e.g., Blynk, MQTT, Firebase) for remote access from anywhere.

**Voice Assistant Integration**

* Connect with Google Assistant, Alexa, or Siri to control devices using voice commands.

**Mobile App Support**

* Develop an Android/iOS app for a more user-friendly experience.

**Scalability for Multiple Devices**

* Add support for multiple ESP32 modules, allowing control of several appliances.
* Implement centralized control via a master ESP32.

**Energy Monitoring & Smart Scheduling**

* Add current sensors to track power usage.
* Use AI-based scheduling to reduce energy consumption.

Conclusion

This project successfully implemented an ESP32-based relay control system for smart lighting and automation. It allowed users to remotely control appliances via a web interface, while also supporting manual push-button operation.

The ESP32’s inbuilt Wi-Fi hotspot played a crucial role in enabling wireless control, even without an internet connection. This feature made the system versatile and accessible, ideal for homes, offices, and industrial automation.

By integrating IoT technology, this project demonstrates the potential of smart lighting systems in improving energy efficiency, convenience, and automation. Future enhancements, such as cloud connectivity and voice control, can further transform it into a fully scalable IoT solution.

Appendices

Source code:

#include <WiFi.h>

#include <WebServer.h>

// Wi-Fi access point credentials

const char\* ssid = "New Summit college Autocontrol Hotspot";

const char\* password = "12345678";

// Static IP configuration

IPAddress local\_IP(192, 168, 1, 100);  // Change to 192.168.1.100

IPAddress gateway(192, 168, 1, 1);     // Typical router IP

IPAddress subnet(255, 255, 255, 0);

// GPIO pins for relays (active low)

const int relayPins[] = {23, 19, 18, 5};

// GPIO pins for push buttons

const int buttonPins[] = {13, 12, 14, 27};

// Relay states

bool relayStates[] = {HIGH, HIGH, HIGH, HIGH}; // Start with relays off

// Create a WebServer object on port 80

WebServer server(80);

// Toggle relay function

void toggleRelay(int relayIndex) {

  relayStates[relayIndex] = !relayStates[relayIndex];

  digitalWrite(relayPins[relayIndex], relayStates[relayIndex]);

}

// HTML content generation function with feedback and styling

String generateHTML() {

  String html = "<!DOCTYPE html><html><head><meta name=\"viewport\" content=\"width=device-width, initial-scale=1.0\">";

  html += "<style>";

  html += "body { font-family: Arial, sans-serif; max-width: 400px; margin: auto; text-align: center; }";

  html += "h1 { color: #333; }";

  html += ".button { display: inline-block; width: 80%; padding: 15px; margin: 10px; font-size: 18px; color: white; background-color: #4CAF50; border: none; border-radius: 5px; text-decoration: none; }";

  html += ".button.off { background-color: #f44336; }";

  html += ".refresh { background-color: #2196F3; padding: 10px 20px; margin: 15px; color: white; font-size: 18px; border: none; border-radius: 5px; }";

  html += ".footer { margin-top: 20px; color: #666; font-size: 14px; }";

  html += "</style></head><body>";

  html += "<h1>ESP32 Relay Control</h1>";

  // Relay control buttons

  for (int i = 0; i < 4; i++) {

    html += "<p>Relay " + String(i+1) + ": " + (relayStates[i] == LOW ? "ON" : "OFF") + "</p>";

    html += "<form action=\"/toggle\_relay\_" + String(i) + "\" method=\"GET\">";

    html += "<button class=\"button " + String(relayStates[i] == LOW ? "off" : "") + "\" type=\"submit\">"

            + String(relayStates[i] == LOW ? "Turn OFF" : "Turn ON") + " Relay " + String(i+1) + "</button>";

    html += "</form><br>";

  }

  // Refresh button

  html += "<form action=\"/refresh\" method=\"GET\">";

  html += "<button class=\"refresh\" type=\"submit\">Refresh</button>";

  html += "</form>";

  // Footer text

  html += "<div class=\"footer\">Tech StudyCell</div>";

  html += "</body></html>";

  return html;

}

// Setup server routes

void setupServerRoutes() {

  // Route for the main page

  server.on("/", HTTP\_GET, []() {

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

  });

  // Route for relay toggle control

  for (int i = 0; i < 4; i++) {

    int relayIndex = i;

    server.on(("/toggle\_relay\_" + String(i)).c\_str(), HTTP\_GET, [relayIndex]() {

      toggleRelay(relayIndex);

      server.send(200, "text/html", generateHTML()); // Update the page with current states

    });

  }

  // Route for refreshing the page without toggling relays

  server.on("/refresh", HTTP\_GET, []() {

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

  });

}

void setup() {

  Serial.begin(115200);

  // Initialize relay pins as output and set them to HIGH (inactive for active-low relays)

  for (int i = 0; i < 4; i++) {

    pinMode(relayPins[i], OUTPUT);

    digitalWrite(relayPins[i], HIGH); // Start with relays off

  }

  // Initialize button pins as input with pull-up resistors

  for (int i = 0; i < 4; i++) {

    pinMode(buttonPins[i], INPUT\_PULLUP);

  }

  // Configure Wi-Fi with a fixed IP and set up as an access point

  WiFi.softAPConfig(local\_IP, gateway, subnet);

  WiFi.softAP(ssid, password);

  Serial.print("Access Point IP: ");

  Serial.println(WiFi.softAPIP());

  // Set up web server routes

  setupServerRoutes();

  // Start the server

  server.begin();

  Serial.println("Web server started.");

}

void loop() {

  // Handle incoming client requests

  server.handleClient();

  // Manual button control

  for (int i = 0; i < 4; i++) {

    if (digitalRead(buttonPins[i]) == LOW) { // Button pressed (active low)

      delay(100); // Debounce delay

      toggleRelay(i); // Toggle the corresponding relay

      while (digitalRead(buttonPins[i]) == LOW); // Wait until button is released

      delay(100); // Debounce delay

    }

  }

}