# NAMIT LOHARKAR BT22CIV101

# EXPERIMENT NO - 8

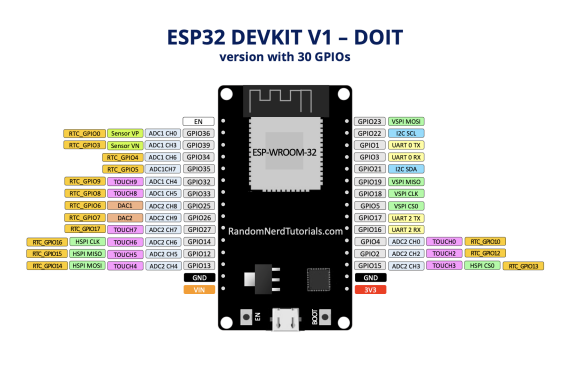
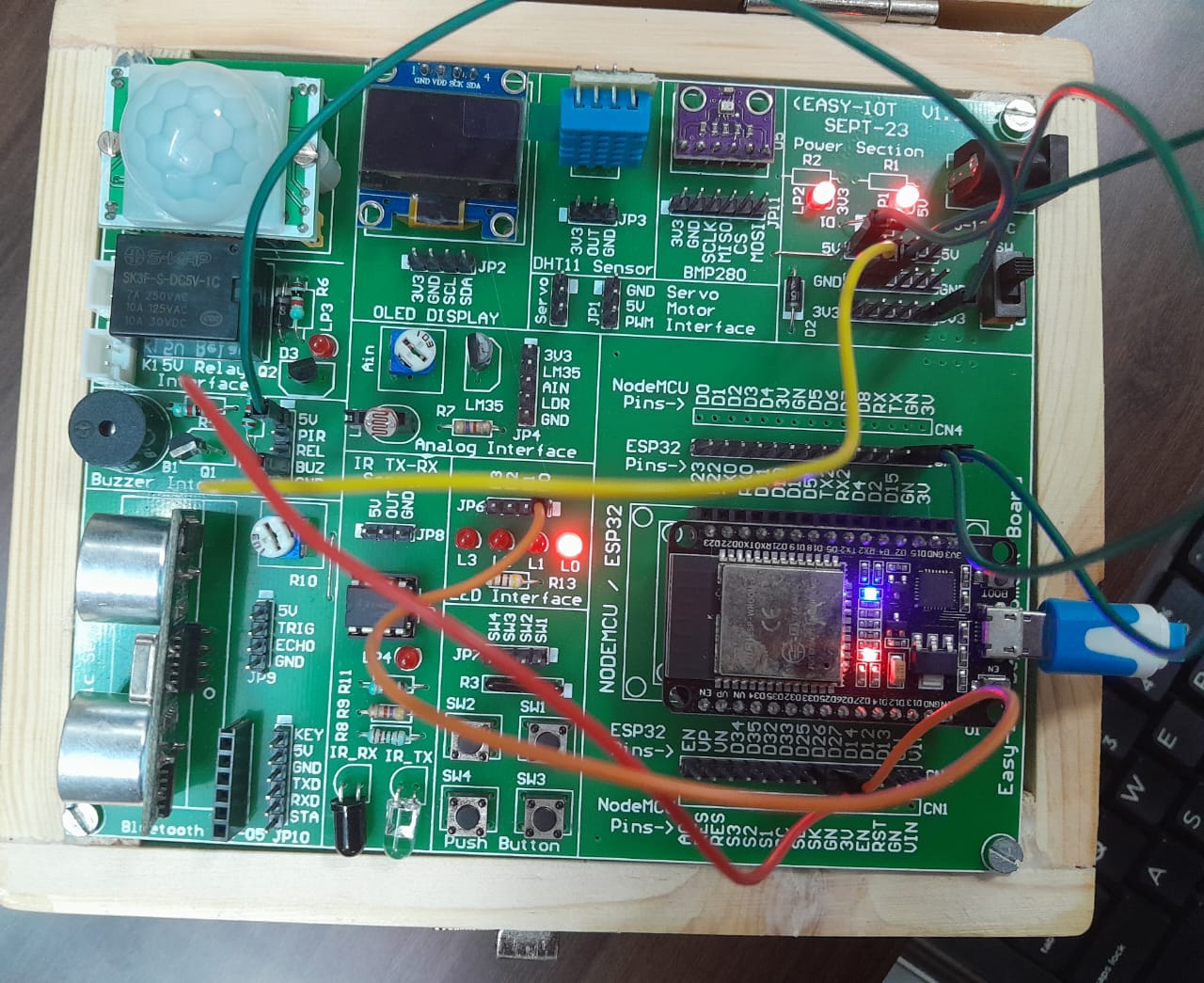
# Aim: Controlling lights and buzzer using an ESP32 Web Server with HTML in the Arduino IDE to control lights and a buzzer

# Apparatus:

- ESP32 Development Board  
- buzzer  
-connecting Jumper Wires  
- Breadboard  
- led  
- USB Cable (for connecting ESP32 to laptop/PC)

**Theory -**The ESP32 microcontroller is a versatile and powerful device often used for IoT applications due to its built-in Wi-Fi and Bluetooth capabilities. It allows the creation of a web server that enables remote control of various connected devices, such as lights and buzzers, using a web-based interface. Through HTML, CSS, and JavaScript, we can develop a simple web page that sends HTTP requests to the ESP32. When a user interacts with the web page, like clicking an “ON” or “OFF” button, these requests are sent to the microcontroller, which interprets them and controls connected devices via GPIO pins. This makes ESP32 a preferred choice for smart home automation, remote monitoring, and control projects.

In this experiment, HTML is embedded within the Arduino IDE, enabling the ESP32 to host a responsive web interface for user interaction. The Arduino code configures specific GPIO pins to control output devices—such as LEDs and buzzers—based on HTTP commands. The ESP32 listens to incoming HTTP requests, processes them, and toggles GPIO pins accordingly. For instance, a command to activate a light or buzzer will set the respective GPIO pin high, supplying power to the device. This approach allows seamless control over appliances without complex wiring or specialized hardware interfaces, highlighting the ESP32's role in IoT and smart homedevelopment.

The application of ESP32 extends far beyond this setup. It opens up potential for IoT networks where multiple ESP32s can communicate over Wi-Fi, offering scalable control and monitoring systems. This project demonstrates the core concepts of remote device management through a simple web server, emphasizing both ESP32’s networking abilities and the practicality of web-based control. Furthermore, the ESP32’s compatibility with various sensors and actuators makes it suitable for real-time IoT applications, such as security systems, energy management, and automation in smart homes and industries.  
  
ESP32 required pin diagram-  
  
Connection diagram-  
  
  
   
  
**Procedure:**

1. **Set Up Arduino IDE**

* Add ESP32 board URL in File > Preferences, install "ESP32 by Espressif Systems" from Board Manager, and select ESP32 Dev Module under Tools > Board.

1. **Connect ESP32 to Wi-Fi**

* Replace ssid and password in the code with your Wi-Fi credentials.

1. **Load and Upload Program**

* Paste the code in Arduino IDE, connect the ESP32, choose the correct COM port, and upload**.**

1. **Monitor Connection**

* Use Serial Monitor to view connection status and IP address once connected.

1. **Access Web Server**

* Open the IP address in a browser on the same Wi-Fi network to control the GPIO pins.

1. **Control Devices**

* Use the web page buttons to toggle GPIO 26 and 27, controlling the connected light and buzzer.

1. **Test and Troubleshoot**

* Confirm that devices respond as expected to the web-based commands.

**CODE-**// Load Wi-Fi library

#include <WiFi.h>

// Replace with your network credentials

const char\* ssid = "Emb\_Lab";

const char\* password = "emb@1234";

// Set web server port number to 80

WiFiServerserver(80);

// Variable to store the HTTP request

String header;

// Auxiliar variables to store the current output state

String output26State = "off";

String output27State = "off";

// Assign output variables to GPIO pins

const int output26 = 26;

const int output27 = 27;

// Current time

unsigned long currentTime = millis();

// Previous time

unsigned long previousTime = 0;

// Define timeout time in milliseconds (example: 2000ms = 2s)

const long timeoutTime = 2000;

void setup() {

Serial.begin(9600);

// Initialize the output variables as outputs

pinMode(output26, OUTPUT);

pinMode(output27, OUTPUT);

// Set outputs to LOW

digitalWrite(output26, LOW);

digitalWrite(output27, LOW);

// Connect to Wi-Fi network with SSID and password

Serial.print("Connecting to ");

Serial.println(ssid);

WiFi.begin(ssid, password);

while (WiFi.status() != WL\_CONNECTED) {

delay(500);

Serial.print(".");

}

// Print local IP address and start web server

Serial.println("");

Serial.println("WiFi connected.");

Serial.println("IP address: ");

Serial.println(WiFi.localIP());

server.begin();

}

void loop(){

WiFiClient client = server.available(); // Listen for incoming clients

if (client) { // If a new client connects,

currentTime = millis();

previousTime = currentTime;

Serial.println("New Client."); // print a message out in the serial port

String currentLine = ""; // make a String to hold incoming data from the client

while (client.connected() &&currentTime - previousTime<= timeoutTime) { // loop while the client's connected

currentTime = millis();

if (client.available()) { // if there's bytes to read from the client,

char c = client.read(); // read a byte, then

Serial.write(c); // print it out the serial monitor

header += c;

if (c == '\n') { // if the byte is a newline character

// if the current line is blank, you got two newline characters in a row.

// that's the end of the client HTTP request, so send a response:

if (currentLine.length() == 0) {

// HTTP headers always start with a response code (e.g. HTTP/1.1 200 OK)

// and a content-type so the client knows what's coming, then a blank line:

client.println("HTTP/1.1 200 OK");

client.println("Content-type:text/html");

client.println("Connection: close");

client.println();

// turns the GPIOs on and off

if (header.indexOf("GET /26/on") >= 0) {

Serial.println("GPIO 26 on");

output26State = "on";

digitalWrite(output26, HIGH);

} else if (header.indexOf("GET /26/off") >= 0) {

Serial.println("GPIO 26 off");

output26State = "off";

digitalWrite(output26, LOW);

} else if (header.indexOf("GET /27/on") >= 0) {

Serial.println("GPIO 27 on");

output27State = "on";

digitalWrite(output27, HIGH);

} else if (header.indexOf("GET /27/off") >= 0) {

Serial.println("GPIO 27 off");

output27State = "off";

digitalWrite(output27, LOW);

}

// Display the HTML web page

client.println("<!DOCTYPE html><html>");

client.println("<head><meta name=\"viewport\" content=\"width=device-width, initial-scale=1\">");

client.println("<link rel=\"icon\" href=\"data:,\">");

// CSS to style the on/off buttons

// Feel free to change the background-color and font-size attributes to fit your preferences

client.println("<style>html { font-family: Helvetica; display: inline-block; margin: 0px auto; text-align: center;}");

client.println(".button { background-color: #4CAF50; border: none; color: white; padding: 16px 40px;");

client.println("text-decoration: none; font-size: 30px; margin: 2px; cursor: pointer;}");

client.println(".button2 {background-color: #555555;}</style></head>");

// Web Page Heading

client.println("<body><h1>ESP32 Web Server</h1>");

// Display current state, and ON/OFF buttons for GPIO 26

client.println("<p>GPIO 26 - State " + output26State + "</p>");

// If the output26State is off, it displays the ON button

if (output26State=="off") {

client.println("<p><a href=\"/26/on\"><button class=\"button\">ON</button></a></p>");

} else {

client.println("<p><a href=\"/26/off\"><button class=\"button button2\">OFF</button></a></p>");

}

// Display current state, and ON/OFF buttons for GPIO 27

client.println("<p>GPIO 27 - State " + output27State + "</p>");

// If the output27State is off, it displays the ON button

if (output27State=="off") {

client.println("<p><a href=\"/27/on\"><button class=\"button\">ON</button></a></p>");

} else {

client.println("<p><a href=\"/27/off\"><button class=\"button button2\">OFF</button></a></p>");

}

client.println("</body></html>");

// The HTTP response ends with another blank line

client.println();

// Break out of the while loop

break;

} else { // if you got a newline, then clear currentLine

currentLine = "";

}

} else if (c != '\r') { // if you got anything else but a carriage return character,

currentLine += c; // add it to the end of the currentLine

}

}

}

// Clear the header variable

header = "";

// Close the connection

client.stop();

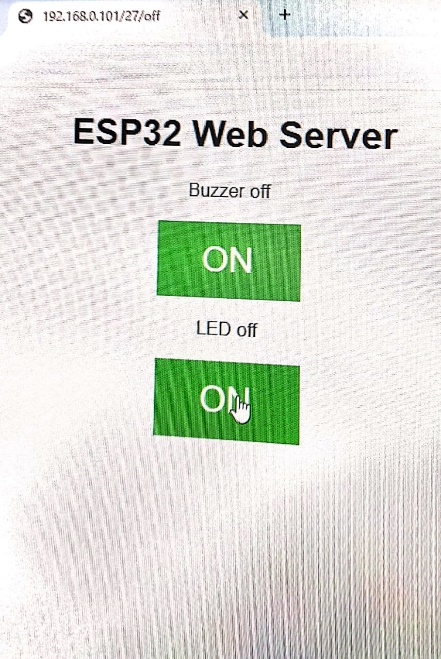
Serial.println("Client disconnected.");

Serial.println("");

}

}

**Photos of results :**

****

**Observations:**

* The ESP32 establishes a stable Wi-Fi connection, allowing real-time control of the light and buzzer via the web interface.
* The response to commands through the web page is quick, with minimal latency, demonstrating reliable connectivity between the ESP32 and the client device.
* The buzzer and LED successfully toggle based on HTTP GET requests, validating the correct mapping of GPIO pins to control outputs.
* Both Serial Monitor and web interface provide feedback on the current state of each device, enhancing user interaction and troubleshooting

# Conclusion:

This experiment demonstrates the practicality of using an ESP32 microcontroller as a web server to control external devices through a web-based interface. The project highlights the ESP32’s capability to integrate with Wi-Fi networks, manage client requests, and control GPIO pins responsively, which makes it ideal for IoT applications. By using simple HTML and HTTP protocols, this setup proves that web-based control of devices is achievable with basic code, laying a foundation for more complex IoT systems and smart home applications.