EXP.No.12: WEBSERVER ON ESP8266 TO CONTROL LED

OBJECTIVES:

1. To set up a web server on the ESP8266 NodeMCU board.
2. To control GPIO pins through a web interface.
3. To monitor and manage GPIO pin states remotely via a web browser.

MATERIALS REQUIRED:

- ESP8266 NodeMCU board
- Breadboard
- Jumper wires
- USB cable
- Computer with Arduino IDE installed
- Web browser for testing

THEORY:

The ESP8266 NodeMCU is a low-cost microcontroller with integrated Wi-Fi capabilities, suitable for Internet of Things (IoT) projects. This experiment demonstrates how to set up a simple web server on the ESP8266 to control GPIO pins. The server listens for HTTP requests and responds with HTML, allowing users to control outputs via a web interface.

CIRCUIT DIAGRAM:

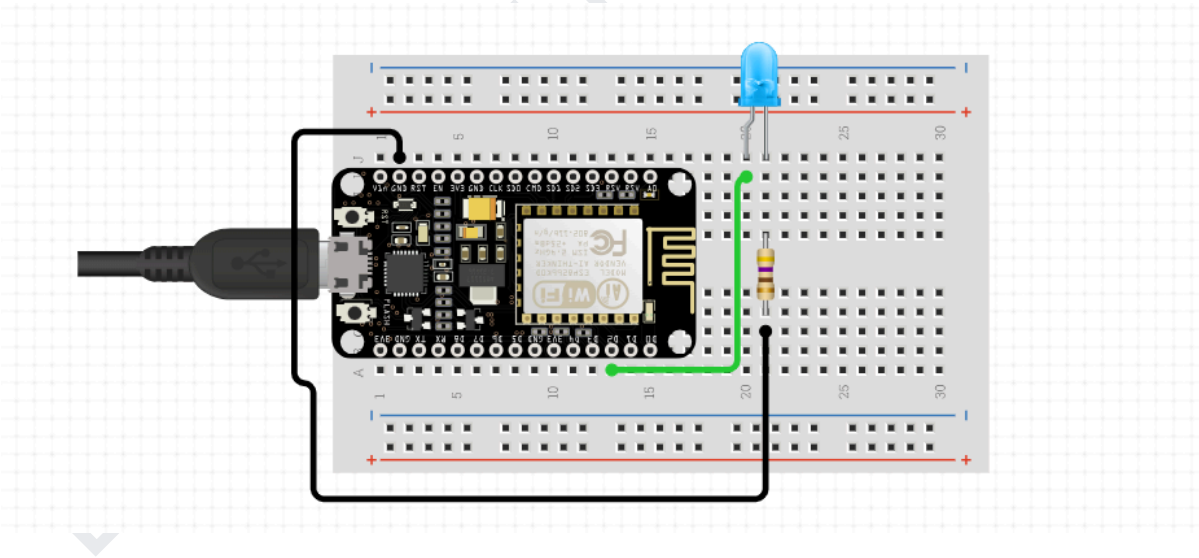

Fig.12.1 Diagram showing the connections for the ESP8266 GPIO pins

GPIO Pin Configuration:

- **GPIO 5**: Connected to a control device or LED.
- **GPIO 4**: Connected to a control device or LED.

PROCEDURE:

1. Hardware Setup:

- Connect the GPIO pins (5 and 4) of the ESP8266 to the devices or LEDs you wish to control.
- Connect the ESP8266 to your computer using a USB cable.

2. Software Setup:

- Open the Arduino IDE on your computer.
- Install the necessary libraries for the ESP8266:
- Navigate to **Sketch** -> **Include Library** -> **Manage Libraries** and search for “ESP8266”.
- Ensure that you have the **ESP8266** board package installed in the Arduino IDE.

3. Programming

- Connect the ESP8266 to your computer using a USB cable.
- In the Arduino IDE, enter the following code:

```
#include <ESP8266WiFi.h>

// Replace with your network credentials
const char* ssid     = "your_ssid";
const char* password = "your_password";

// Set web server port number to 80
WiFiServer server(80);

// Variable to store the HTTP request
String header;

// Auxiliar variables to store the current output state
String output5State = "off";
String output4State = "off";

// Assign output variables to GPIO pins
const int output5 = 5;
const int output4 = 4;

// 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(115200);
  // Initialize the output variables as outputs
  pinMode(output5, OUTPUT);
  pinMode(output4, OUTPUT);
  // Set outputs to LOW
  digitalWrite(output5, LOW);
  digitalWrite(output4, 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,
    Serial.println("New Client.");          // print a message out in the serial port
    String currentLine = "";                // make a String to hold incoming data from the client
    currentTime = millis();
    previousTime = currentTime;
      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 /5/on") >= 0) {
        Serial.println("GPIO 5 on");
        output5State = "on";
        digitalWrite(output5, HIGH);
      } else if (header.indexOf("GET /5/off") >= 0) {
        Serial.println("GPIO 5 off");
        output5State = "off";
        digitalWrite(output5, LOW);
      } else if (header.indexOf("GET /4/on") >= 0) {
        Serial.println("GPIO 4 on");
        output4State = "on";
        digitalWrite(output4, HIGH);
      } else if (header.indexOf("GET /4/off") >= 0) {
        Serial.println("GPIO 4 off");
        output4State = "off";
        digitalWrite(output4, 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: #195B6A; border: none; color: white; padding:
16px 40px;");
          client.println("text-decoration: none; font-size: 30px; margin: 2px; cursor: pointer;}");
          client.println(".button2 {background-color: #77878A;}</style></head>");

          // Web Page Heading
          client.println("<body><h1>ESP8266 Web Server</h1>");

          // Display current state, and ON/OFF buttons for GPIO 5
          client.println("<p>GPIO 5 - State " + output5State + "</p>");
          // If the output5State is off, it displays the ON button
          if (output5State == "off") {
            client.println("<p><a href=\"/5/on\"><button class=\"button\">ON</button></a></p>");
          } else {
                                client.println("<p><a    href=\"/5/off\"><button    class=\"button
button2\">OFF</button></a></p>");
          }

          // Display current state, and ON/OFF buttons for GPIO 4
          client.println("<p>GPIO 4 - State " + output4State + "</p>");
          // If the output4State is off, it displays the ON button
          if (output4State == "off") {
            client.println("<p><a href=\"/4/on\"><button class=\"button\">ON</button></a></p>");
          } else {
                                  client.println("<p><a    href=\"/4/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("");
  }
}
```

- Replace "your_ssid" and "your_password" with your actual network credentials.
- Upload the code to the ESP8266.

4. Testing the Web Server:

- Open the Serial Monitor in the Arduino IDE to observe the connection status and IP address of the ESP8266.

- Enter the IP address of the ESP8266 into a web browser.

- You should see a web page with ON/OFF buttons to control the GPIO pins.

Observations:

Record the state changes of GPIO pins by interacting with the web page. Note the following:

- State of GPIO 5 (ON/OFF)
- State of GPIO 4 (ON/OFF)

Result:

The ESP8266 successfully serves a web page that allows remote control of GPIO pins. The ON/OFF buttons on the web page change the states of GPIO 5 and GPIO 4 accordingly.

Conclusion:

This experiment demonstrates how to create a basic web server with the ESP8266 to control GPIO pins via

D. Reference link with QR code

https://www.youtube.com/watch?v=bFJRQOjdAkE

This format provides a clear and comprehensive guide for conducting the experiment, ensuring students can follow along and achieve the desired outcomes.