Aditya Vishe BT2CIV091

Experiment No. 8

Aim

To create a web-based control system for lights and a buzzer using an ESP32. This project uses an ESP32 microcontroller to create a web server that allows users to control connected devices (lights and a buzzer) remotely through an HTML interface.

Apparatus

- ESP32 microcontroller
- LED lights
- Buzzer
- Jumper wires
- Breadboard
- Power source (USB or battery)
- Wi-Fi-enabled device (to access the web server)

Theory

The ESP32 microcontroller has built-in Wi-Fi capabilities, allowing it to host a web server. In this project, the ESP32 creates an HTTP server to handle user requests. Using an HTML interface, users can send commands to turn the lights and buzzer on or off by toggling the GPIO pins. The ESP32 receives these HTTP requests, processes them, and sends the appropriate signals to the GPIO pins, which control the connected devices.

Procedure

1. **Setup**:
    - Connect the LED and buzzer to the ESP32's GPIO pins 26 and 27 respectively.
    - Ensure that the power source is connected to the ESP32.
2. **Code**:
    - Upload the provided code to the ESP32 using the Arduino IDE.
    - This code connects the ESP32 to a specified Wi-Fi network, initializes GPIO pins 26 and 27 as output pins, and creates a web server on port 80.
    - When the ESP32 receives HTTP requests from the user, it parses them to determine if GPIO 26 or GPIO 27 should be turned on or off, then updates the HTML interface to reflect the current state.

    Provided 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
WiFiServer server(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


 String currentLine = ""; // make a String to hold incoming


 while (client.connected() && currentTime - previousTime <= timeoutTime)
{ // loop while the client's connected
 currentTime = millis();
 if (client.available()) { // if there's bytes to read fromthe client,
 char c = client.read(); // read a byte, then
```

```
Serial.write(c); // print it out the serialmonitor
header += c;
if (c == '\n') { // if the byte is a newlinecharacter
// if the current line is blank, you got two newline characters ina 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/1.1200 OK)
// and a content-type so the client knows what's coming, then ablank 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-sizeattributes 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\"><buttonclass=\"button\">ON</button></a></p>");
 } else {
```

```
 client.println("<p><a href=\"/26/off\"><button
class=\"buttonbutton2\">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\"><buttonclass=\"button\">ON</button></a></p>");

 } else {

 client.println("<p><a href=\"/27/off\"><button
class=\"buttonbutton2\">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 carriagereturn 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("");
}
}
```

3. **Operation**:
    - Open a web browser on a device connected to the same Wi-Fi network as the ESP32.
    - Enter the IP address shown in the serial monitor of the Arduino IDE.
    - Control the lights and buzzer by clicking the on/off buttons on the web page.

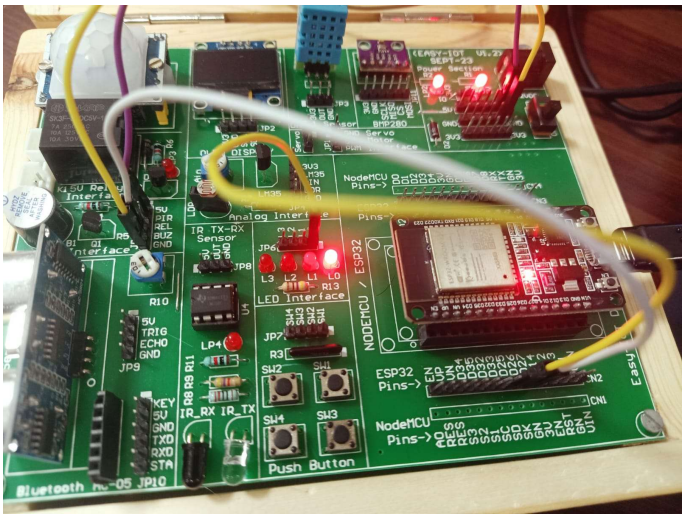
Observations

During the operation of the ESP32 web server control system for the lights and buzzer, several key observations were noted. The response time of the lights and buzzer to commands sent through the web interface was generally quick and consistent, indicating reliable communication between the ESP32 and connected devices. However, occasional delays occurred, likely due to Wi-Fi network variations or processing time within the ESP32 itself. The HTML interface provided clear status indicators for each device, allowing easy monitoring of whether each component was in the "on" or "off" state. This functionality was found to be user-friendly, with button clicks on the web interface accurately reflecting changes in the physical devices connected to the ESP32. Additionally, the web server successfully hosted the control interface accessible via any device on the network, confirming the ESP32's capability to serve web content reliably.

Conclusion

The ESP32-based web server system demonstrated an effective approach to remotely controlling connected devices, such as lights and a buzzer, using a simple HTML interface. This project highlights the ESP32's capabilities for Internet of Things (IoT) applications, where remote control and monitoring are key. The successful implementation of this web-based control system illustrates how embedded systems can be enhanced with wireless capabilities to operate various devices seamlessly from a web browser. Overall, the project is a valuable example of using an ESP32 for practical applications in home automation or remote device management, showcasing both versatility and ease of setup in creating user-friendly control interfaces.

Results:


IR TX-RX Sensor
LED Interface
NODEMCU / ESP32
ESP32 Pins->
NodeMCU Pins->
Push Button
Bluetooth HC-05 JP10