EE-432L Computer Networks CEP

Submitted to:

Dr. Naveed Nawaz

Submitted by:

Areeba Naeem	2021-EE-51
Ayesha Ahmad	2021-EE-52
Saira Taufeeq	2021-EE-56
Aina Shafqat	2021-EE-58

Department of Electrical Engineering

University of Engineering and Technology Lahore

Table of Contents

SMART PLUG

Overview ---------- 3

Softwares/IDEs ---------- 3

Components ---------- 3
Circuit Diagram ---------- 4

Physical Circuit ---------- 5

Python Server / Publisher Code ---------- 6

Handler ---------- 7

Server ---------- 7

MQTT ---------- 8

ESP32 / Subscriber Code ---------- 9

Paho-MQTT AND DNSLIB ---------- 8

EXPLANATION ---------- 11

Webpage ---------- 13

SMART PLUG

OVERVIEW

A **smart plug** is a device that allows you to remotely control electrical appliances by connecting them to a power outlet through the plug. It transforms regular appliances into "smart" devices by enabling remote operation, scheduling, and energy monitoring.

Smart plugs are often integrated into smart home systems and are controlled using mobile apps, voice assistants (like Alexa or Google Assistant), or through automation protocols such as **MQTT**, **Zigbee**, or **Wi-Fi**.

APPLICATIONS OF SMART PLUG

Home Automation: Automate lights, fans, heaters, or air conditioners.

Energy Monitoring: Measure the power usage of devices like refrigerators or washing machines.

Convenience: Control hard-to-reach appliances.

Security: Simulate occupancy by scheduling lights or devices while away.

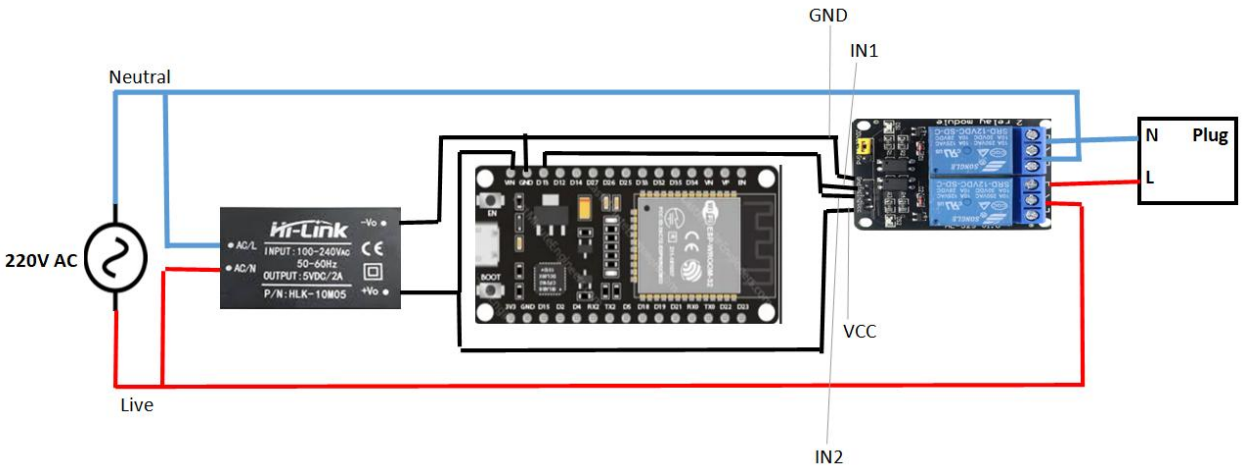
SOFTWARE/IDEs

- C Programming (ESP32 / Subscriber)
- Python (Back-end / Publisher)
- HTML(Webpage)
- JSON

COMPONENTS

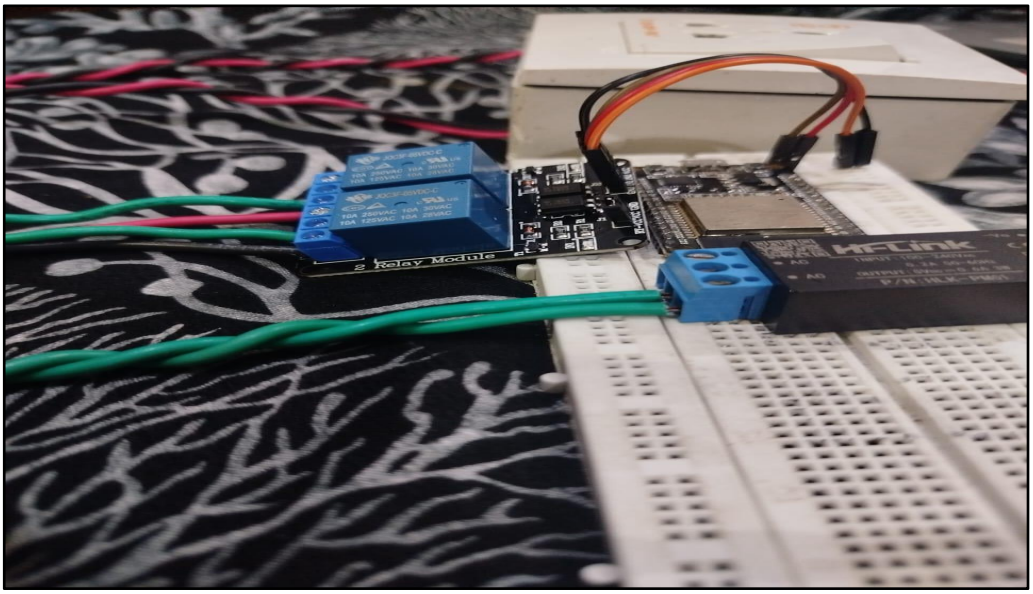
- Hi-Link HLK-PM03 AC DC Converter 220V to 3.3V 3W 1 A Module
- ESP-WROOM-32Wifi module
- Relay Module
- Jumper Wires
- Breadboard

CIRCUIT DIAGRAM

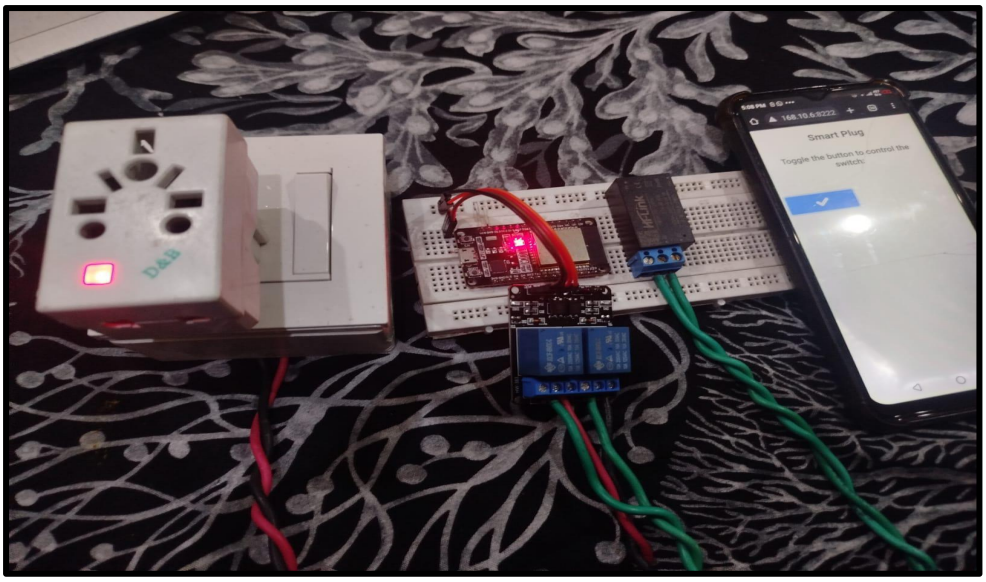


HARDWARE OF ABOVE CIRCUIT

Physical Connections of the Components



WORKING SMART PLUG HARDWARE



EXPLANATION

This setup demonstrates a smart plug system controlled via an ESP32 microcontroller. The ESP32 is connected to a Wi-Fi network and hosts a web interface accessible through a smartphone, allowing the user to toggle the power of the connected appliance. A relay module, connected to the ESP32, acts as a switch to control the flow of electricity to the smart plug. The system includes a power socket (smart plug), ESP32 board, relay module, breadboard for prototyping, and a mobile interface for control. By interacting with the web interface, the user can remotely turn the appliance ON or OFF, showcasing a simple yet effective smart home automation system.

This smart plug setup also includes an AC-DC converter, which is essential for powering the ESP32 and relay module. The AC-DC converter steps down the high-voltage AC power from the mains to a low-voltage DC supply (typically 5V or 3.3V) suitable for the ESP32 and other low-power components.

PYTHON SERVER/PUBLISHER CODE

Publisher

```
from http.server import SimpleHTTPRequestHandler
from dns.resolver import Resolver
from dnslib.server import DNSServer
import json
import paho.mqtt.client as mqtt
import time
import os
```

```
lient()
broker_address, broker_port, 60) # 60s = Keep Alive interval is the duration to wait for ACK before retransmission.

(SimpleHTTPRequestHandler):
self):
.path == '/':
f.send_response(200)
f.send_header('Content-type', 'text/html')
f.end_headers()

pre_state[f'btn1']:
 button = (f"""
        <div class="form-check form-switchmx-auto">
          <input id="btn1" onclick="buttonClicked('btn1')" class="form-check-input" checked type="checkbox" style="width: 9em; height: 3em;">
        </div>
        """)
e:
 button = (f"""
        <div class="form-check form-switchmx-auto">
          <input id="btn1" onclick="buttonClicked('btn1')" class="form-check-input" type="checkbox" style="width: 9em; height:3em;">
        </div>
        """)
```

```
    l_content = """
    OCTYPE html>
    ml lang="en">

    ad>
    meta charset="UTF-8">
    meta name="viewport" content="width=device-width, initial-scale=1.0">
    title>Smart Plug</title>
    link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.1/dist/css/bootstrap.min.css" rel="stylesheet"
    ntegrity="sha384-4bw+/aepP/YC94hEpVNVgiZdgIC5+VKNBQNGCHeKRQN+PtmoHDEXuppvnDJzQIu9" crossorigin="anonymous">
    ead>
    dy>

    v class="container text-center">
    br/>
    h2><strong>Smart Plug</strong></h2>
    div class="text-center">
     <br/>
     <h5>Toggle the button to control the switch:</h5>
     <br/><br/>
     <div>
       """ + button + """
     </div>
    /div>
```

```
  iv>

  ript>
  unction buttonClicked(buttonId) {
   var checkbox = document.getElementById(buttonId);
   var xhr = new XMLHttpRequest();
   xhr.open("GET", "/" + buttonId + "," + checkbox.checked,true);
   xhr.send();

  cript>

  ript
  rc="https://cdn.jsdelivr.net/npm/bootstrap@5.3.1/dist/js/bootstrap.bundle.min.js"
  ntegrity="sha384-HwwvtgBNo3bZJJLYd8oVXjrBZt8cqVSpeBNS5n7C8IVInixGAoxmnlMuBnhbgrkm" crossorigin="anonymous">
  cript>

  ody>
  tml>

  f.wfile.write(html_content.encode())
```

```
ton_id = self.path[1:] # Remove the leading '/'
f.handle_button_click(button_id)

button_click(self, btn_id):
id[0:3] == 'btn':
, state = btn_id.split(',')
 0
_state[btn] = False
state == 'true':
 pre_state[btn] = True
 s = 1
h open('states.json', 'w') as file:
 file.write(json.dumps(pre_state))

ic = "CN/CEP"
sage = f"{btn}{s}"

ent.publish(topic, message)
nt(f"Button {btn} state: {state}")

"__main__":
68.10.6"
```

```
ath.isfile('states.json'): # Make json file to save pre_state of switch, if not exists
te = {"btn1": False} # Switch off by default
en('states.json', 'w') as file:
e.write(json.dumps(pre_state))
ds pre_state from existing json file
en('states.json') as file:
_state = json.loads(file.read())

Resolver()
meservers = ['8.8.8.8']

SServer(resolver,logger=None,tcp=True,
        address=IP,port=PORT,handler=MyHandler)

t_thread()
er.serve_forever()

op()
```

TERMS TO EXPLAIN THAT ARE IN CODE

Handler

The MyHandler class is responsible for managing the webpage, processing button interactions, and handling all webpage-related operations.

Webpage

The HTML content for the webpage is defined in the html_content variable within the do_GET method of the MyHandler class.

Server

A TCP server is launched at the designated address and port, utilizing the custom MyHandler class. The states of the buttons are saved locally in a JSON file named states.json to preserve their previous configurations.

MQTT AND DNS

The **paho-mqtt library**, a third-party Python package, is employed to publish messages to the MQTT broker on a specific topic (CN_CEP in this case). This functionality is implemented in the handle_button_click method of the MyHandler class. We have also imported **dnslib** as it is a lightweight library for building, and handling DNS packets. It is often used to create custom DNS servers and handle DNS queries or responses programmatically the other library is **dnspython** which is a comprehensive library for querying DNS records, managing zone files,

The libraries we import are

```
!pip install paho-mqtt
!pip install dnslib
!pip install dnspython
```

```
Requirement already satisfied: paho-mqtt in ./.venv/lib/python3.12/site-packages (2.1.0)
Requirement already satisfied: dnslib in ./.venv/lib/python3.12/site-packages (0.9.25)
Requirement already satisfied: dnspython in ./.venv/lib/python3.12/site-packages (2.7.0)
```

ESP32/SUBSCRIBER CODE

C: > Users > PMLS > Downloads > main.cpp > ...

```cpp
#include <WiFi.h>
#include <PubSubClient.h>

// Your WiFi credentials
// const char *ssid = "Redmi Note 13";
// const char *password = "katawaredoki6";
const char *ssid = "Ahmad2";
const char *password = "mabutt1957";

// MQTT broker settings
const char *mqtt_broker = "broker.hivemq.com";
const char *topic = "CN_CEP/SmartPlug";
const int mqtt_port = 1883;

int msg[5];
int pin = 13;

WiFiClient espClient;
PubSubClient client(espClient);

void callback(char *m_topic, byte *payload, unsigned int length) {
    for (int i = 0; i < length; i++) {
        msg[i] = (char) payload[i] - '0';
    }

```

```
    if (msg[4]) {
        Serial.println("Turn ON");
        digitalWrite(pin, HIGH);
    } else {
        Serial.println("Turn OFF");
        digitalWrite(pin, LOW);
    }
}

void setup() {
    Serial.begin(115200);
    pinMode(LED_BUILTIN, OUTPUT);
    digitalWrite(LED_BUILTIN, HIGH);

    pinMode(pin, OUTPUT);

    WiFi.begin(ssid, password);
    while (WiFi.status() != WL_CONNECTED) {
        delay(500);
        Serial.println("Connecting to WiFi...");
    }
    Serial.println("Connected to the Wi-Fi network");

    client.setServer(mqtt_broker, mqtt_port);
    client.setCallback(callback);
    digitalWrite(LED_BUILTIN, LOW);

    while (!client.connected()) {
        String client_id = "esp32-client-";
        client_id += String(WiFi.macAddress());
        Serial.printf("The client %s connects to the public MQTT broker\n", client_id.c_str());

        if (client.connect(client_id.c_str())) {
            Serial.println("Public EMQX MQTT broker connected");
            client.subscribe(topic);
        } else {
            Serial.println("Failed connection");
            delay(2000);
        }
    }
}

void loop() {
    client.loop();
}
```

EXPLANATION

Wi-Fi and MQTT Setup

- The code connects the ESP32 to a Wi-Fi network using the provided SSID and password (Ahmad2 and mabutt1957). Once connected, the ESP32 is ready to communicate over the internet.
- It then configures the MQTT client to connect to a public broker (broker.hivemq.com) on port 1883 and subscribe to the topic "CN_CEP/SmartPlug". MQTT enables lightweight communication, allowing the ESP32 to publish or receive messages.

Callback Function for Message Handling

The callback function processes incoming MQTT messages.

It reads the payload of the message, converts it into an integer array (msg), and checks the fifth element (msg[4]) to determine whether to turn the connected device ON or OFF.

If msg[4] is 1, the ESP32 turns ON the device by setting the output pin (pin = 13) HIGH. Otherwise, it turns the device OFF by setting the pin LOW.

ANSWERS TO THE CEP ASSIGNMENT

1- Selection of WIFI chipset module:

We are using the **ESP32-WROOM** for the smart plug for the following reasons:
Size: Compact form factor (typically 25.5 x 18 x 3 mm) and it have Integrated Wi-Fi and Bluetooth capabilities.

Low cost: It is easily accessible at the low price of round about **1500pkr.**
Low-power: the ESP32 consumes very little power compared with other microcontrollers 3.3 to 5 Volts.
Wi-Fi capabilities: In order to operate our smart plug via an online page, the ESP32 can readily establish a Wi-Fi network to connect to the internet or establish its own Wi-Fi wireless network
Compatible with the Arduino: It can be programmed in Arduino style in the Arduino IDE.

2- AC-DC converter:

Hi link5V is used to convert the coming 220 AC voltage to 5 DC voltage. This 5V(DC)is provided as a power supply to the ESP32-WROOM. The range of input voltages of the chipset lies between 3.3 to 5 volts.

3- Select an appropriate software:

We choose publisher on python and subscriber (MQTT client on esp32) and we have used (HTML) for developing the webpage. Bootstrap and CSS for styling the webpage. There is a button on the webpage when changes its state is considered as an event then a TCP server is initiated at the specified address and the PORT using our custom handler MyHandler. The previous states of the buttons are stored locally in a JSON file "states.json". The library paho-mqtt is used for publishing messages on the MQTT broker and the specified topic (CN/CEP in our case), and is done in the handle_button_click method of the handler class.

4- Select an appropriate software for the development of code for the smart plug according to the selected chipset.

The ESP32 code for the smart plug IS developed using the **PlatformIO** in Visual Studio Code. These environments provide libraries like **WiFi.h** and **PubSubClient.h** that simplify the development process for ESP32 and support real-time hardware control and MQTT protocol implementation.

5- Set up DNS network server and webpage(s) for the control of the smart plug according to the real-time instructions given by the user.

The Python script initializes a DNS server using http.server on the specified IP and port (e.g., 192.168.10.6:8222). The server dynamically serves a webpage containing a toggle button for the smart plug, which sends real-time commands to the ESP32. The server uses Bootstrap for styling the webpage and JavaScript for sending GET requests when the button is toggled.

6- Use MQTT protocol to connect between web pages and the hardware of smart plug for transmission of commands and data such as plug name, plug ID etc. and for the display of plug status in real time.

The Python server uses the Paho-MQTT library to publish messages to the MQTT broker on the topic "CN_CEP/SmartPlug". These messages include the button state, which the ESP32 (acting as the subscriber) receives and processes to toggle the plug's relay state. For

example, when the button is toggled, the message format is btn10 or btn11, indicating OFF or ON states.

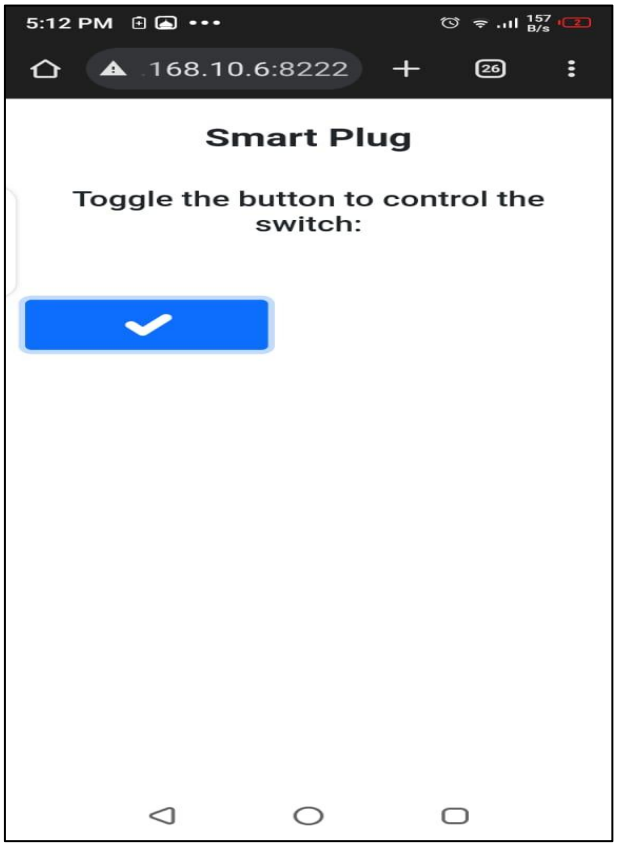
7- **To reduce the size of message in the data transmission process, choose between MySQL and JSON format and justify your choice for environments with low speed of internet.**

JSON is preferred in this implementation for the following reasons:

- Lightweight and Efficient: JSON is used to store and retrieve the button's state locally (states.json), making it faster and more suitable for environments with low-speed internet.
- Simple Integration: JSON allows direct handling of data structures in both Python and ESP32, reducing complexity compared to the overhead of SQL queries required for MySQL.

WEBPAGE

WHEN SWITCH IS "ON"



WHEN SWITCH IS “OFF”