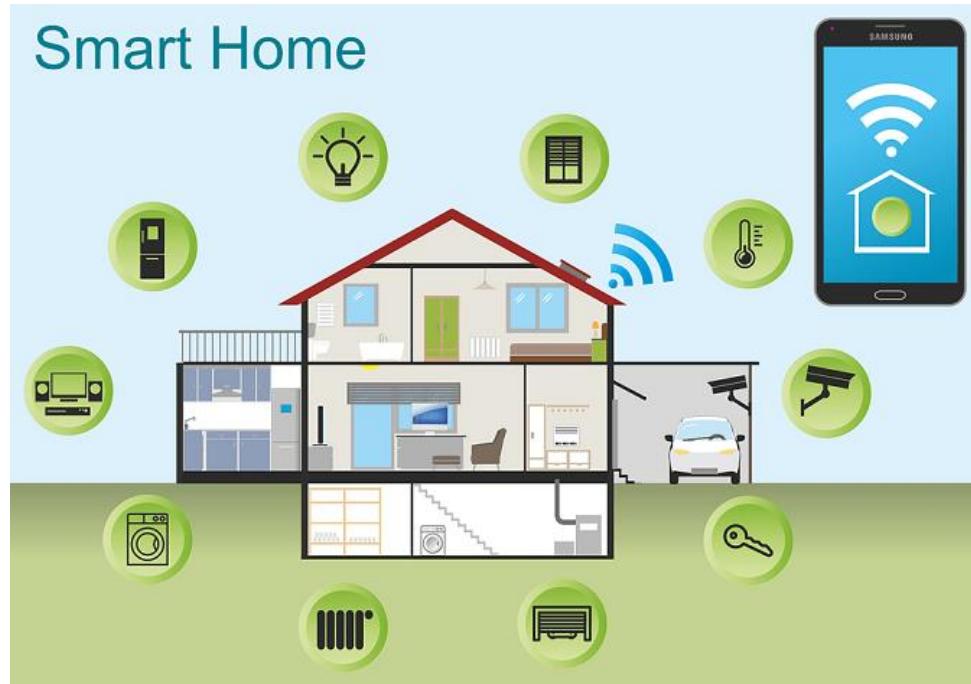


**INSTITUTE OF INDUSTRIAL
ELECTRONICS ENGINEERING,
(PCSIR) KARACHI**



THIRD YEAR SPRING SEMESTER 2024

COMPLEX ENGINEERING PROBLEM



Submitted By:

Mehak Sattar 21019,Mujtaba Abbas 21034

Sameed Siddiqui 21029,Areeb Arfi 21022

TASK 1:

IDENTIFICATION OF THE PROBLEM STATEMENT

PROBLEM STATEMENT:

Devices that can be monitored and controlled via the Internet are part of what is known as the Internet of Things (IoT). Your task is to design and implement a comprehensive smart home system utilizing ESP8266/ESP32 modules. This system should be capable of managing and automating various home functionalities, including lighting, heating, security cameras, and door locks. The project will require you to integrate multiple sensors and actuators, develop a central control algorithm, establish reliable network communication, and design a user-friendly interface. This system should allow homeowners to monitor and control these devices remotely, providing convenience, energy efficiency, and enhanced security.

IDENTIFICATION:

The problem at hand revolves around the increasing demand for smart home systems that enhance convenience, energy efficiency, and security for homeowners. In traditional homes, managing various devices such as lighting, heating, security cameras, and door locks often requires manual intervention, which can be inefficient and lacks the flexibility that modern living demands.

WHY DO WE NEED AN AUTOMATED SYSTEM?

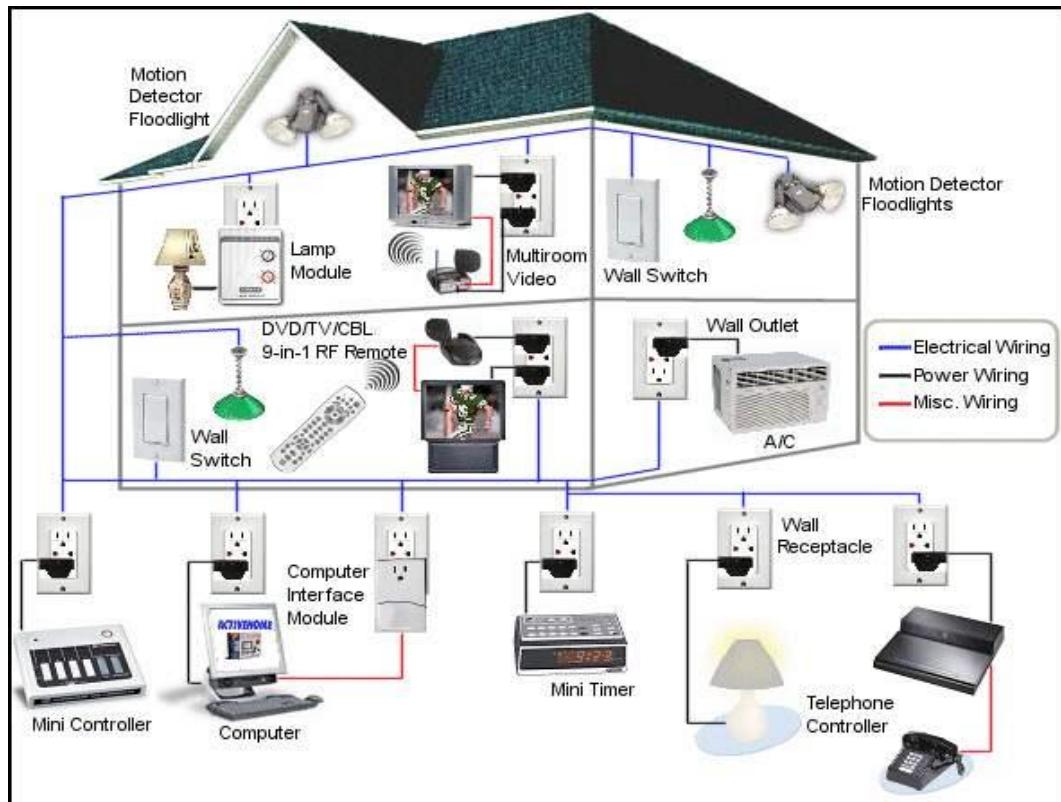
The need for a smart home system arises from the growing desire for greater convenience, efficiency, and security in our daily lives. Here's why such a system is essential:

Convenience:

A smart home system allows homeowners to control various devices and functions—such as lighting, heating, security cameras, and door locks—remotely via a smartphone or computer. This eliminates the need for manual operation, making it easier to manage and customize the home environment to suit individual preferences.

Energy Efficiency:

With smart home systems, devices can be programmed to operate only when needed, reducing unnecessary energy consumption. For example, lights can automatically turn off when no one is in the room, and heating systems can adjust based on the time of day or occupancy. This not only lowers utility bills but also contributes to environmental sustainability.



Enhanced Security:

A smart home system enhances security by allowing real-time monitoring of security cameras, door locks, and sensors. Homeowners can receive alerts about suspicious activities, lock doors remotely, or even simulate occupancy when away, deterring potential intruders.

HOW IOT WILL HELP IN OUR CEP:

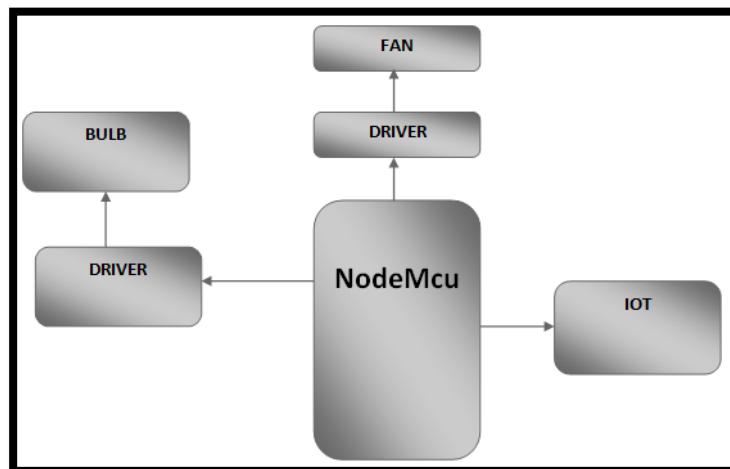
In my complex engineering program, IoT is the backbone that will enable the design and implementation of a comprehensive smart home system.

- Device Connectivity and Communication:**

IoT allows various devices in the smart home, such as lights, thermostats, security cameras, and door locks, to be interconnected. These devices can communicate with each other and with a central control system using the ESP8266/ESP32 modules.

- Automation and Efficiency:**

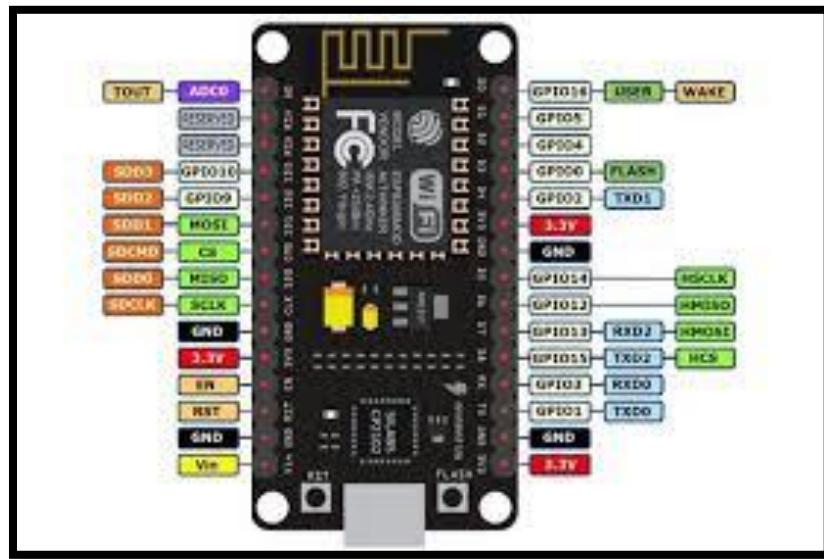
IoT facilitates automation by allowing devices to respond to specific conditions or commands without manual intervention.



ESP8266 AND ESP32 MODULES:

The ESP8266 and ESP32 are widely used microcontroller modules in IoT applications due to their low cost, versatility, and built-in Wi-Fi capabilities

ESP8266 MODULE OVERVIEW:



- **Wi-Fi Connectivity:**

The ESP8266 has an integrated Wi-Fi module that allows it to connect to a local network or the internet, enabling remote control and monitoring of connected devices.

- **GPIO Pins:**

The module provides multiple General Purpose Input/Output (GPIO) pins, which can be used to interface with sensors, relays, and other devices in the smart home system.

KEY FEATURES:

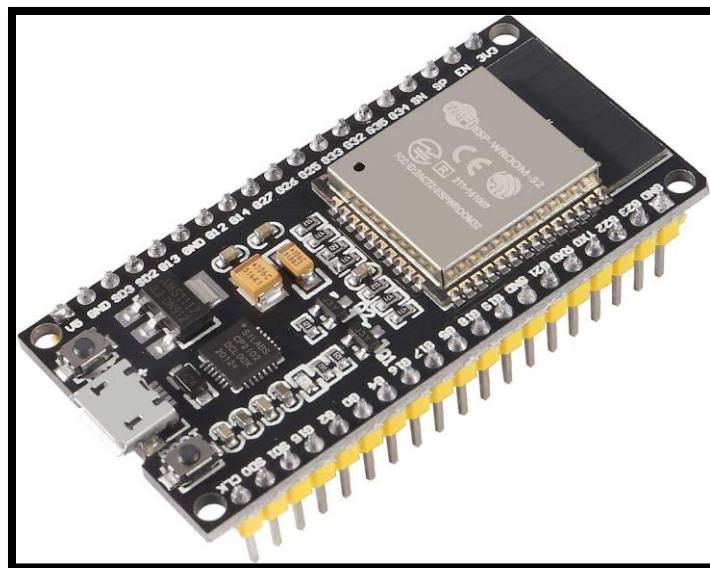
- The ESP8266 is a low-cost Wi-Fi microchip with full TCP/IP stack and microcontroller capability. It has become popular in IoT projects because of its affordability and ease of integration with other components.

- It operates at 80 MHz and has a 32-bit RISC CPU, with a variety of GPIO pins that can be used to connect sensors, actuators, and other peripherals.

USE IN HOME AUTOMATION:

- The ESP8266 can be used to control individual devices like lights, fans, or security sensors. It can send data from sensors to a central controller and receive commands for device operation.
- Its Wi-Fi capability allows these devices to be accessed and controlled remotely through a smartphone app or web interface.

ESP32 MODULE OVERVIEW:



- The ESP32 is an advanced version of the ESP8266, offering more processing power, memory, and additional features such as Bluetooth connectivity alongside Wi-Fi.
- It operates at a higher clock speed (up to 240 MHz) and includes dual-core processing, making it suitable for more complex IoT applications.

KEY FEATURES:

- **Wi-Fi and Bluetooth Connectivity:**

The ESP32 supports both Wi-Fi and Bluetooth (Classic and Low Energy), providing more options for device communication. This is particularly useful for integrating with a broader range of IoT devices and peripherals.

- **GPIO Pins and Peripheral Interfaces:**

The ESP32 has more GPIO pins and additional interfaces such as SPI, I2C, UART, and PWM, allowing for more complex sensor and actuator configurations.

USE IN HOME AUTOMATION:

- The ESP32 can serve as the central controller of the smart home system, managing multiple devices, processing sensor data, and executing control algorithms. Its dual-core processing capability makes it suitable for handling real-time tasks such as security monitoring, device automation, and network communication simultaneously.

COMPARISON IN OUR SMART HOME SYSTEM

- **ESP8266:**

Best suited for individual or simpler IoT tasks, such as controlling a specific device (like a smart light or thermostat) or gathering data from a sensor. It can be used in multiple locations throughout the home, each handling specific tasks and communicating with the central ESP32 module.

- **ESP32:**

Ideal as the central hub or brain of the smart home system, coordinating the activities of various ESP8266 modules and managing complex tasks like security monitoring, data processing, and system-wide automation. Its additional processing power and connectivity options make it crucial for handling more demanding applications and integrating a wider range of devices.

Task 2

LITERATURE REVIEW:

ii. Literature review

- a. Review the research which has been conducted in this regard
- b. Add at least five research papers and add their references

Smart Home System using ESP8266:

<https://journalinstal.cattleyadf.org/index.php/Instal/article/view/224/130>

SUMMARY:

The paper explores energy-saving strategies in smart home automation using IoT. It highlights how ESP8266 modules can be utilized for real-time monitoring and control of home appliances.

Development Home Automation and Safety Circuit Breaker with Esp8266 Microcontroller

https://semarakilmu.com.my/journals/index.php/appl_mech/article/view/7076/5318

SUMMARY:

This paper describes the design of a smart home system utilizing ESP8266 modules, focusing on ease of integration, scalability, and cost-effectiveness.

TASK 3:

DETAILED DOCUMENT:

- iii. A detailed document outlining the project's scope, objectives, and expected outcomes. This should include:
 - a. Project description
 - b. Goals and objectives
 - c. System architecture

PROJECT DESCRIPTION:

The project involves designing and implementing a comprehensive smart home system using ESP8266/ESP32 modules. This system will automate and control various home functionalities, such as lighting, heating, security cameras, and door locks.

KEY COMPONENTS:

- ESP32 Central Controller:
- ESP8266 Modules
- Sensors and Actuator
- User Interface
- Network Communication

PROJECT SCOPE:

- Design and Implementation
- Sensor Integration
- Actuator Control

GOALS AND OBJECTIVES:

Goals:

- Create a Modular and Scalable Smart Home System
- Enhance Home Security and Convenience

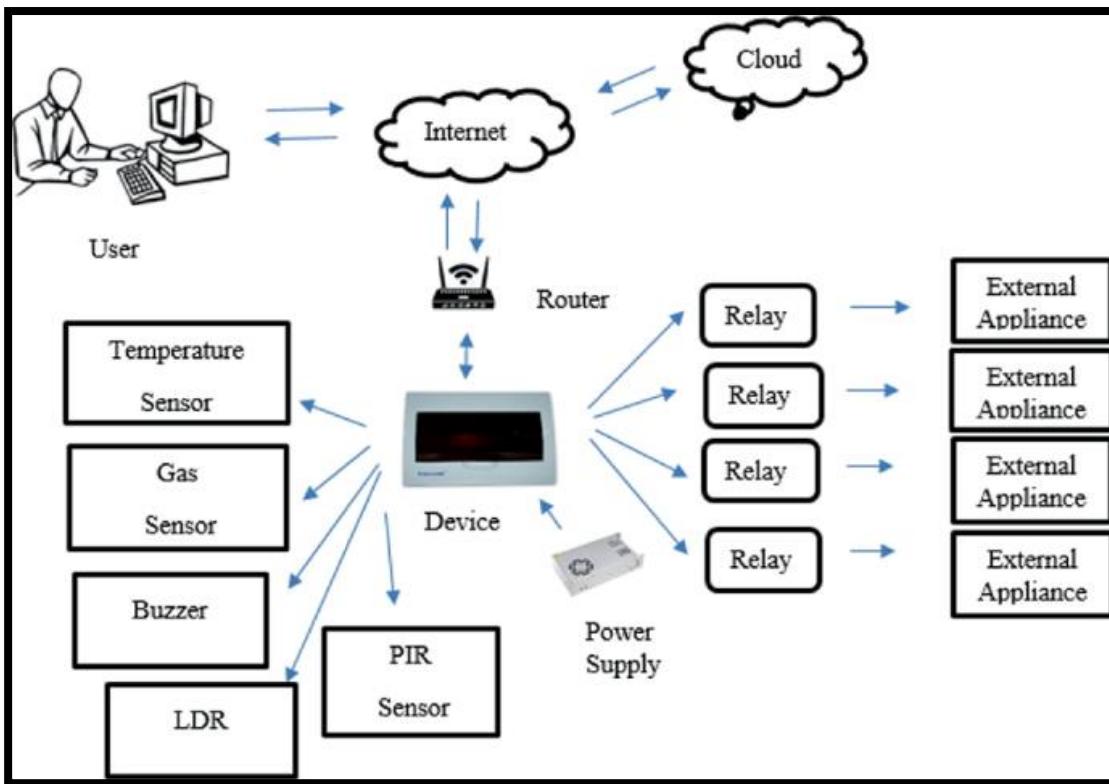
- Improve Energy Efficiency

Objectives:

- Develop a Central Control System
- Implement Wireless Communication
- Design a User Interface
- Integrate Sensors and Actuator
- Test and Optimize the System

SYSTEM ARCHITECTURE:

Architecture Overview: The system architecture is centered around the ESP32 module, which acts as the central controller, coordinating with multiple ESP8266 modules that control individual home functionalities. The ESP32 module communicates with the ESP8266 modules over a Wi-Fi network, which also allows the system to interface with a user's mobile app or web interface for remote access.



Components:

- **ESP32 Central Controller:**
 - Manages communication between different modules.
 - Processes data from sensors and sends commands to actuators.
 - Interfaces with the user interface for remote access.
- **ESP8266 Modules:**
 - Each module is assigned a specific task, such as controlling lights, heating, or security cameras.
 - Communicates with the ESP32 controller via Wi-Fi.
- **Sensors:**
 - Sensors like temperature, motion, and light sensors are connected to ESP8266 modules to provide data.
- **Actuators:**
 - Actuators such as relays, motors, and locks are controlled by the ESP8266 modules based on commands from the ESP32.
- **User Interface:**
 - A mobile app or web-based interface allows users to monitor and control the system remotely.
- **Wi-Fi Network:**
 - Facilitates communication between the ESP32, ESP8266 modules, and the user interface.

Expected Outcomes:

Successful Integration: A fully integrated smart home system that allows remote control and monitoring of home functionalities.

User Accessibility: A user-friendly interface that enables easy interaction with the system.

Enhanced Security and Efficiency: Improved home security and energy efficiency through automation.

Scalability: A system architecture that allows for easy addition of new modules, sensors, and actuators as needed.

Project Scope

Inclusions:

Development of a central control system using ESP32.

Integration of ESP8266 modules with sensors and actuators.

Implementation of a wireless communication protocol (Wi-Fi).

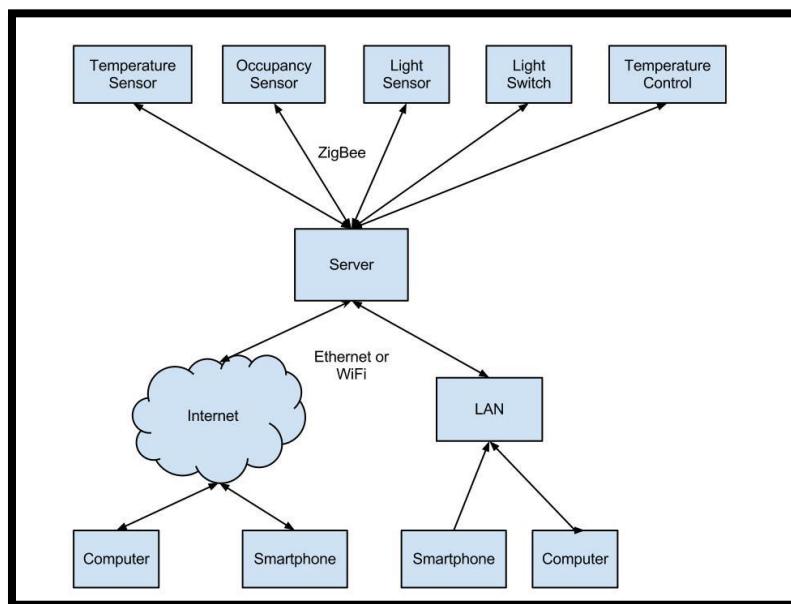
Design of a user-friendly interface for remote access and control.

Testing and optimization of the system for reliability and efficiency.

Exclusions:

The project will not cover the physical installation of sensors and actuators in an actual home environment.

No third-party cloud services will be integrated unless specified.



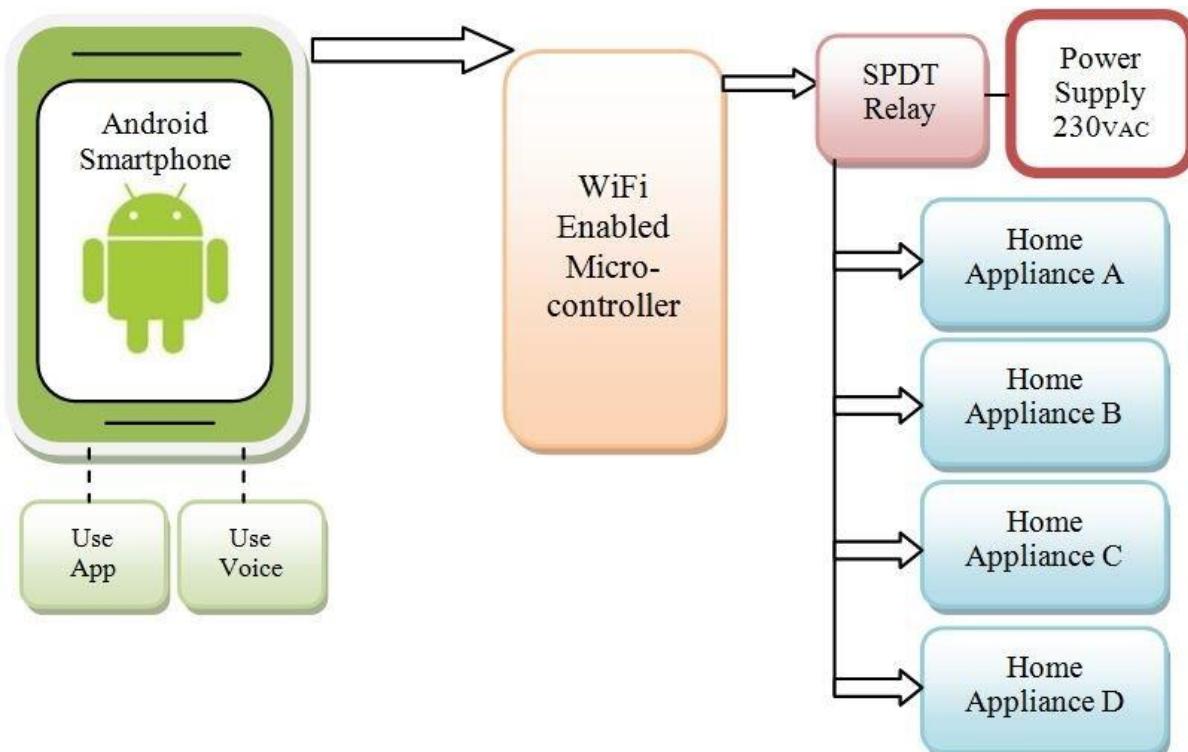
TASK 4:

SOFTWARE AND HARDWARE SCHEMATIC

System Design Documentation: Comprehensive documentation of the system design, including:

- a. Block diagrams and flowcharts
- b. Detailed hardware schematics
- c. Software architecture and module descriptions
- d. Communication protocols and data flow

BLOCK DIAGRAM



SOFTWARE ARCHITECTURE AND MODULE

Main Software Components:

Controller Firmware (ESP32): Manages communication, processing, and control.

Module Firmware (ESP8266): Handles sensor data acquisition and actuator control.

User Interface Software: Provides a web/mobile interface for user interaction.

Wi-Fi Communication Protocols: Ensures secure and reliable data transmission.

Module Descriptions

ESP32 Controller Firmware:

Functions: Initializing modules, processing sensor data, executing user commands.

Modules:

Network Manager: Handles Wi-Fi connectivity.

Data Processor: Analyzes sensor data and triggers appropriate actions.

Command Executor: Executes commands received from the user interface.

ESP8266 Firmware:

Functions: Gathering sensor data, controlling actuators.

Modules:

Sensor Interface: Collects data from connected sensors.

Actuator Controller: Sends signals to actuators based on received commands.

User Interface Software:

Functions: Provides access to monitor and control the smart home system.

Modules:

Dashboard: Displays system status (e.g., temperature, security status).

Success Criteria

Functional Integration:

Criterion: All components of the smart home system, including sensors, actuators, and controllers, are successfully integrated and communicate effectively.

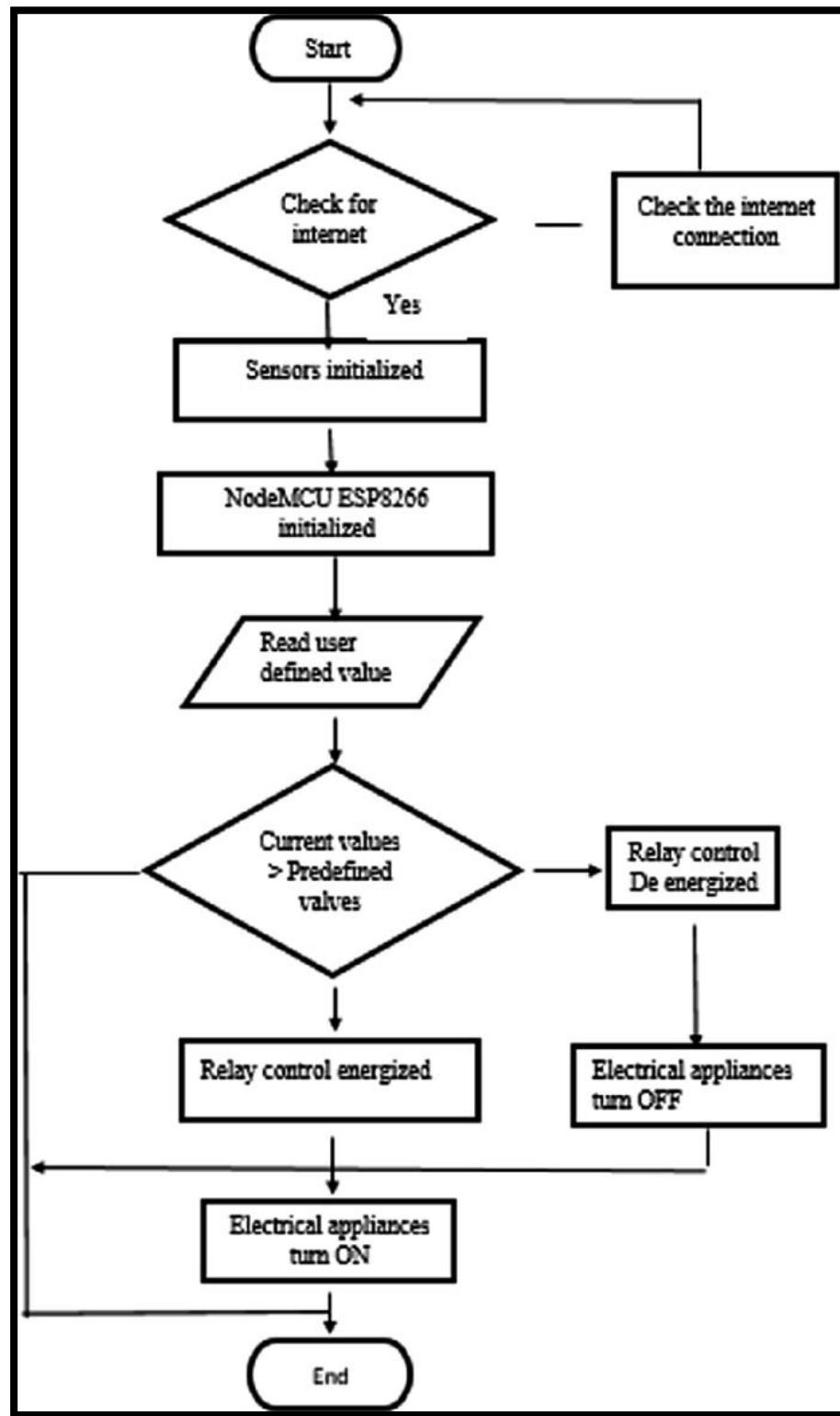
Measure: The system can perform all intended functions (e.g., lighting control, heating management, security monitoring) without errors or interruptions.

Remote Control and Monitoring:

Criterion: The system provides reliable remote access via a mobile app or web interface.

Measure: Users can control and monitor devices from a remote location with minimal latency and no connectivity issues.

Software Architecture Overview

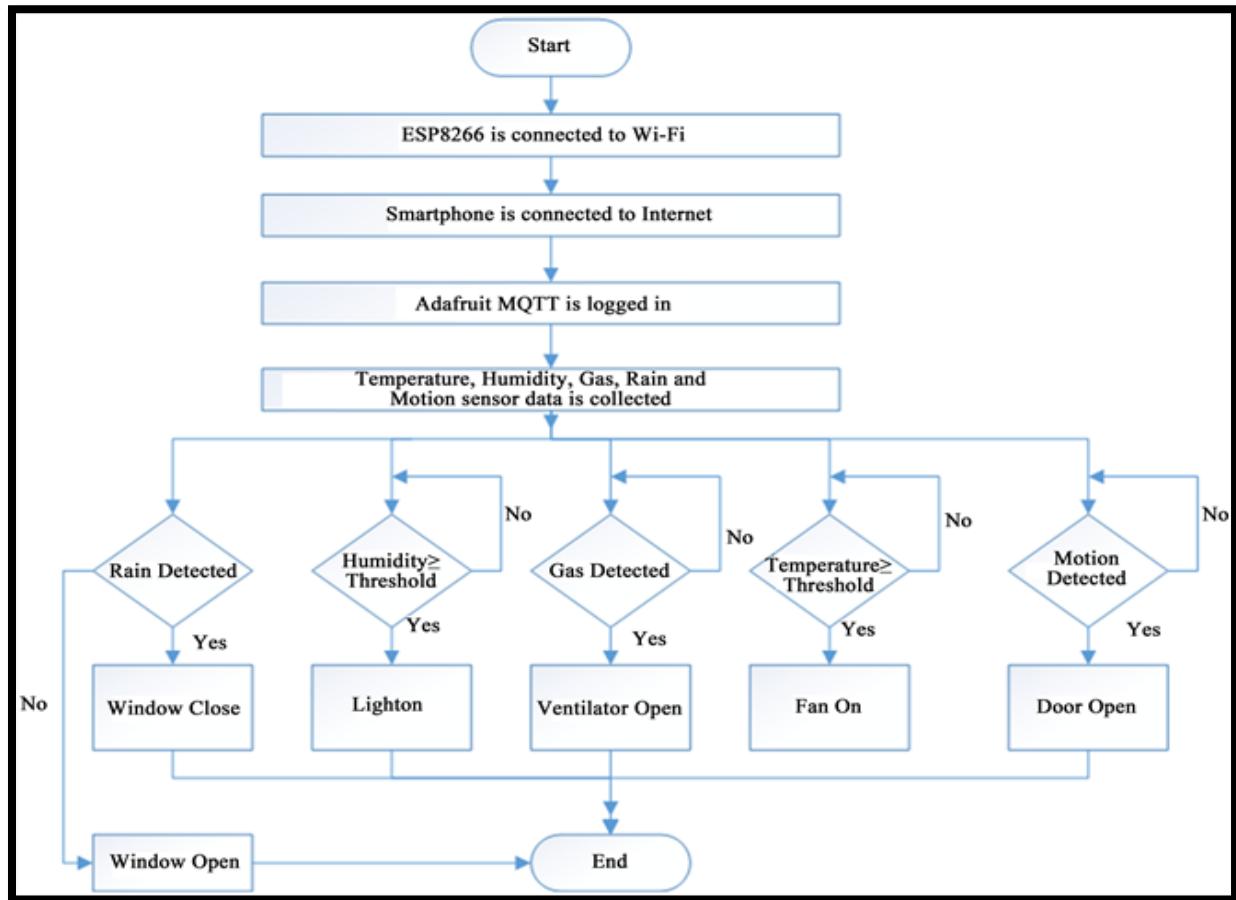


COMMUNICATION PROTOCOLS

Communication Protocols

- **Wi-Fi Communication:**
- **Protocol:** 802.11 b/g/n standard.
- **Data Security:** Use of WPA2 for secure communication.
- **Data Transmission:** JSON format for data exchange between ESP32 and ESP8266 modules.
- **Inter-Module Communication:**
- **Protocol:** MQTT or HTTP for lightweight communication.
- **Data Exchange:** Sensor readings, control commands, and system status updates.

Data Flow Description

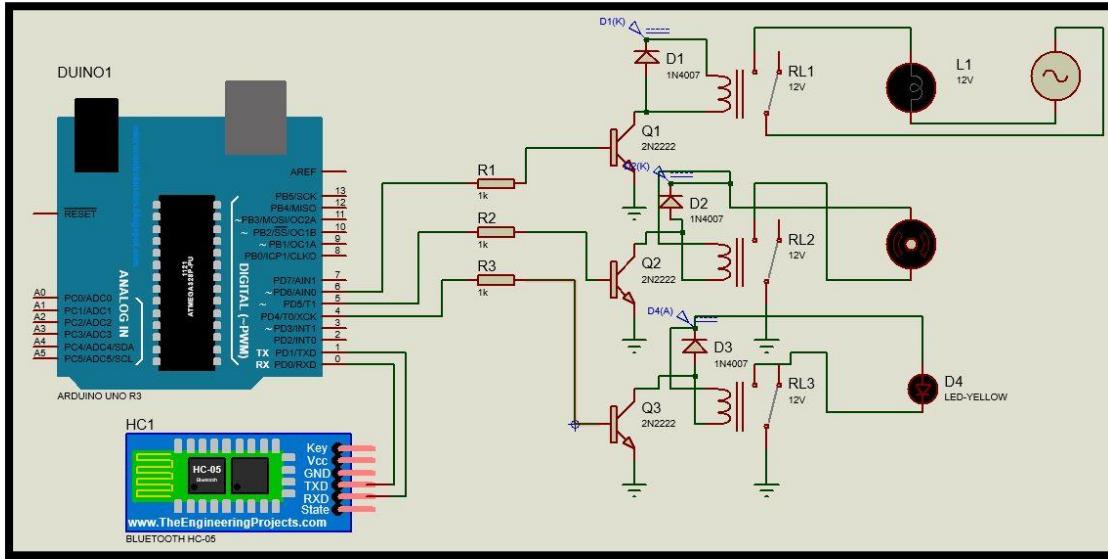


TASK 5:

SOFTWARE CODE

```
#include <WiFi.h>
#include <PubSubClient.h> // MQTT client library
const char* ssid = "Your_SSID";
const char* password = "Your_PASSWORD";
const char* mqtt_server = "broker.hivemq.com";
WiFiClient espClient;
PubSubClient client(espClient);
const int relayLightPin = 5
const int heaterPin = 4;
const int tempPin = A0;
const int motionSensorPin = 14;
int desiredTemp = 22;
void setup() {
    Serial.begin(115200);
    pinMode(relayLightPin, OUTPUT);
    pinMode(heaterPin, OUTPUT);
    pinMode(motionSensorPin, INPUT);
    setup_wifi();
    client.setServer(mqtt_server, 1883);
    client.setCallback(mqttCallback);
    digitalWrite(relayLightPin, LOW);
    digitalWrite(heaterPin, LOW);
}
void setup_wifi() {
    delay(10);
    Serial.println();
    Serial.print("Connecting to ");
    Serial.println(ssid);
    WiFi.begin(ssid, password);
    while (WiFi.status() != WL_CONNECTED) {
        delay(500);
        Serial.print(".");
    }
    Serial.println("");
    Serial.println("WiFi connected");
}
void mqttCallback(char* topic, byte* message, unsigned
int length) {
    String msg;
    for (int i = 0; i < length; i++) {
        msg += (char)message[i];
    if (String(topic) == "home/relay/light") {
        if (msg == "ON") {
            digitalWrite(relayLightPin, HIGH); // Turn on light
        } else if (msg == "OFF") {
            digitalWrite(relayLightPin, LOW); // Turn off light
        }
    }
    if (String(topic) == "home/relay/heater") {
        if (msg == "ON") {
            digitalWrite(heaterPin, HIGH);
        } else if (msg == "OFF") {
            digitalWrite(heaterPin, LOW);
        }
    }
}
void reconnect() {
    while (!client.connected()) {
        Serial.print("Attempting MQTT connection... ");
        if (client.connect("ESP32Client")) {
            Serial.println("connected");
            client.subscribe("home/relay/light");
            client.subscribe("home/relay/heater");
        } else {
            Serial.print("failed, rc=");
            Serial.print(client.state());
            delay(5000);
        }
    }
}
void controlHeating() {
    int currentTemp = analogRead(tempPin); // Read the
temperature sensor
    currentTemp = map(currentTemp, 0, 1023, -40, 125);
    if (currentTemp < desiredTemp) {
        digitalWrite(heaterPin, HIGH);
    } else {
        digitalWrite(heaterPin, LOW)
    }
    String tempPayload = "Temperature: " +
String(currentTemp);
    client.publish("home/sensor/temperature",
tempPayload.c_str());
}
void loop() {
    if (!client.connected()) {
        reconnect();
    }
    client.loop();
    controlHeating();
    if (digitalRead(motionSensorPin) == HIGH) {
        client.publish("home/sensor/motion", "Motion
Detected");
    }
    delay(1000); // Adjust as needed for your system's
response time
}
```

HARDWARE SCHEMATIC:



OUTPUT:

When you run the code, the serial monitor will display various messages related to Wi-Fi connection status, MQTT connection, and other debugging information.

1. Wi-Fi Connection
2. Temperature Reading and Control
3. Motion Detection.