# IoT BASED MULTICONTROLLED SMART HOME AUTOMATION AND PROTECTION SYSTEM




## SONARGAON UNIVERSITY (SU)

**Supervised By**

**Md. Ferdous Khan**
Assistant Professor
Department of EEE
Sonargaon University, SU

**Submitted By**

| | |
|---|---|
| **Md. Shakib Mia** | **EEE-2202026022** |
| **Md. Naeim Hasan** | **EEE-2202026069** |
| **Md. Sabbir Hossain** | **EEE-2202026047** |
| **Md. Habibur Rahman** | **EEE-2202026044** |
| **Md. Khirul Islam Fahim** | **EEE-2202026050** |

**Department Of Electrical And Electronic Engineering (EEE)**
Sonargaon University, Dhaka, Bangladesh
Date of Submission: September-2025

# DECLARATION

This is to certify that the project titled "IoT Based Multicontrolled Smart Home Automation And Protection System" is the result of our study in partial fulfillment of the B.Sc. Engineering degree under the supervision of Md. Ferdous Khan, Assistant Professor, Department of Electrical and Electronic Engineering (EEE), Sonargaon University, Bangladesh. It is also hereby declared that this project or any part of it has not been submitted elsewhere for the award of any degree.

## Signature of the Candidates

---

Md. Shakib Mia
EEE-2202026022

---

Md. Sabbir Hossain
EEE-2202026047

---

Md. Naeim Hasan
EEE-2202026069

---

Md. Habibur Rahman
EEE-2202026044

---

Md. Khirul Islam Fahim
EEE-2202026050

# CERTIFICATE OF APPROVAL

The project titled **"IoT Based Multicontrolled Smart Home Automation And Protection System"** submitted by **Md. Shakib Mia**, ID No: EEE-2202026022, **Md. Sabbir Hossain**, ID No: EEE 2202026047, **Md. Naeim Hasan**, ID No: EEE-2202026069, **Md. Habibur Rahman**, ID No: EEE-2202026044, and **Md. Khirul Islam Fahim**, ID No: EEE-2202026050, has been accepted as satisfactory in partial fulfillment of the requirement for the degree of Bachelor of Science in Electrical and Electronic Engineering on September 2025.

**Supervised By**

---

**Md. Ferdous Khan**

Assistant Professor

Department of EEE

Sonargaon University, SU

# ACKNOWLEDGEMENT

First of all, we would like to express our deepest gratitude to almighty "ALLAH" for giving us the strength, patience, and ability to successfully complete our project as a partial fulfillment of the requirements for the Degree of Bachelor of Science in Electrical and Electronic Engineering.

We would like to express our sincere gratitude and appreciation to our supervisor, Md. Ferdous Khan, Assistant Professor, Department of Electrical and Electronic Engineering, Sonargaon University, Bangladesh, for his continuous guidance, valuable advice, and encouragement throughout the progress of this work.

We are also grateful to all the honorable faculties of the Department of Electrical and Electronic Engineering of Sonargaon University, Bangladesh for giving us the opportunity to complete this work and for their necessary support during the entire period of our study.

This thesis work would not have been possible without the encouragement, constructive feedback, and logical help from our friends and well-wishers. We are truly thankful to them.

Finally, we must acknowledge with due respect the constant inspiration, sacrifice, and patience of our parents. Their support has been a continuous source of motivation in completing this work.

# ABSTRACT

The IoT Based Multicontrolled Smart Home and Protection System is designed to provide complete monitoring, automation, and protection of household electrical loads over the internet. The system allows users to operate electrical and electronic appliances using smartphones, computers, infrared remote, manual switches, and voice commands through AI assistants such as Google Home.

The project is implemented using ESP32 as the central controller, which integrates various sensors and modules. A DHT11 sensor is used for temperature and humidity monitoring, while an MQ-2 smoke sensor ensures safety by activating a buzzer during fire or smoke detection. Electrical loads such as lights and fans are controlled through a relay module. The system supports multi-control operation, ensuring both flexibility and user convenience.

Overall, this smart home system demonstrates a low-cost, flexible, and efficient IoT-based solution for modern household automation and safety. It reduces energy consumption by enabling intelligent monitoring and switching of loads, while ensuring fire protection and AI-based control. This work highlights the potential of IoT and AI integration in creating safer and smarter living environments.

Keywords: IoT, Smart Home, ESP32, Load Control, AI Assistant, Smoke Detection, Arduino Cloud, Protection System.

# TABLE OF CONTENTS

**DECLARATION** ..... **ii**

**CERTIFICATE OF APPROVAL** ..... **iii**

**ACKNOWLEDGEMENT** ..... **iv**

**ABSTRACT** ..... **v**

**TABLE OF CONTENTS** ..... **vi**

**List Of Figure** ..... **x**

**List of Table** ..... **xi**

**CHAPTER 1** ..... **1**

**INTRODUCTION** ..... **1**

1.1 Introduction ..... 1

1.2 Internet of Things (IoT) ..... 1

1.3 AI and Smart Home Automation ..... 2

1.4 Load Monitoring and Controlling ..... 2

1.5 Problem Statement ..... 2

1.6 Motivation ..... 3

1.7 Objectives ..... 3

1.8 Research Methodology ..... 4

1.9 Conclusion ..... 4

1.10 Project Outline ..... 4

**CHAPTER 2** ..... **6**

**ANALYSIS OF THE SYSTEM COMPONENT** ..... **6**

2.1 Introduction ..... 6

2.2 Required Components ..... 6

2.3 ESP32 Development Board ..... 7

2.3.1 Specification of ESP32 ..... 8

2.3.2 Pin Functions of ESP32 ..... 9

2.4 ESP32 Terminal Adapter ..... 11
2.4.1 Purpose and Usage in Our Project ..... 11
2.5 DHT11 Temperature & Humidity Sensor ..... 12
2.5.1 Specification of DHT11 ..... 13
2.5.2 Pin Functions of DHT11 Sensor Module ..... 13
2.5.3 Working Principle of DH11 ..... 14
2.6 MQ-2 Smoke & Gas Sensor ..... 15
2.6.1 Specification of MQ-2 ..... 16
2.6.2 Pin Functions of MQ-2 ..... 16
2.6.3 Working Principle of MQ-2 ..... 18
2.7 IR Receiver (TSOP1838) ..... 19
2.7.1 Specifications of IR Receiver (TSOP1838) ..... 20
2.7.2 Pin Functions of IR Receiver (TSOP1838) ..... 20
2.7.3 Working Process of IR Receiver (TSOP1838) ..... 21
2.7.4 Block Diagram of TSOP1838 IR Reciver ..... 22
2.8 IR Remote ..... 22
2.8.1 Features of IR Remote ..... 23
2.8.2 HEX Code Decoding Process ..... 23
2.8.3 Functions of Remote Keys ..... 25
2.9 Manual Switches ..... 26
2.9.1 Switching Mechanism in Project ..... 26
2.9.2 Pin Mapping with ESP32 ..... 26
2.10 Relay Module (4-channel) ..... 27
2.10.1 Specification of Relay Module (12V) ..... 28
2.10.2 Pin Functions of Relay Module (12V) ..... 28
2.11 Buzzer Module ..... 29
2.11.1 Specifications of Buzzer Module ..... 30
2.11.2 Pin Functions of Buzzer Module ..... 30
2.11.3 Working Process of Buzzer Module ..... 31
2.12 SMPS (Switch Mode Power Supply) ..... 31
2.12.1 Circuit Diagram of smps ..... 32

2.12.2 Working Principle ..... 32
2.12.3 Application in Our Project ..... 32
2.13 Power Module Multi Output Voltage Conversion ..... 33
2.13.1 Specifications of Power Module Multi Output Voltage ..... 34
2.13.2 Purpose in Our Project ..... 34
2.14 USB Output (5V Supply for ESP32 & Sensors) ..... 34
2.14.1 Specifications of USB Output ..... 35
2.15 Circuit Breaker ..... 36
2.16 Electrical Dual Row Terminal Block ..... 37
2.17 Connecting Wire and Jumper ..... 37
2.18 Lugs (Wire Connectors/Crimping Lugs) ..... 38
2.19 Summary ..... 39

**Chapter 3 ..... 40**

**HARDWARE DEVELOPMENT ..... 40**
3.1 Introduction ..... 40
3.2 Block Diagram ..... 40
3.2.1 Block Diagram of Power Section ..... 40
3.2.2 Block Diagram of Controller Unit ..... 41
3.3 Designing Software (Arduino IDE, Arduino IoT Cloud, Google Home Integration) ..... 42
3.4 Circuit Diagram ..... 43
3.5 Construction of the Main Circuit ..... 44
3.6 Working Principle ..... 44
3.7 Flow Chart ..... 45
3.7.1 Flowchart Step by Step Operation ..... 46
3.8 Summary ..... 46

**Chapter 4 ..... 47**

**RESULT AND DISCUSSION ..... 47**
4.1 Introduction ..... 47
4.2 Result ..... 47

4.3 Advantages of the Monitoring System ..... 47
4.4 Limitations of the System ..... 48
4.5 Cost Estimation of the project..... 48
4.6 Applications of the System ..... 49
4.7 Summary ..... 49

**Chapter 5 .....50**

**CONCLUSION ..... 50**
5.1 Conclusion ..... 50
5.2 Future Scope of Modification ..... 50

**REFERENCES..... 51**

**APPENDIX..... 52**

# List Of Figure

Figure 2. 1: ESP32 Development Board ..... 8
Figure 2. 2: Pin Functions of ESP32 ..... 9
Figure 2. 3: ESP32 Terminal Adapter ..... 11
Figure 2. 4: DHT11 Temperature & Humidity Sensor ..... 13
Figure 2. 5: Pin Functions of DHT11 Sensor Module ..... 14
Figure 2. 6: Working Principle of DH11 ..... 14
Figure 2. 7: MQ-2 Smoke & Gas Sensor ..... 15
Figure 2. 8: Pin Functions of MQ-2 ..... 17
Figure 2. 9: Pin Functions of MQ-2 Module ..... 17
Figure 2. 10: Current flow process in MQ2 ..... 18
Figure 2. 11: Output voltyage in MQ2 Sensor Module ..... 18
Figure 2. 12: IR Receiver (TSOP1838) ..... 19
Figure 2. 13: Pin Functions of IR Receiver (TSOP1838) ..... 20
Figure 2. 14: Block Diagram of TSOP1838 IR Reciver ..... 22
Figure 2. 15: IR Remote ..... 22
Figure 2. 16: Wiring Connections  Decoding HEX Code ..... 23
Figure 2. 17: Reading HEX Codes ..... 25
Figure 2. 18: Manual Switches ..... 26
Figure 2. 19: Relay ..... 27
Figure 2. 20: (4 chanel Relay Module) ..... 27
Figure 2. 21: Pin Functions of Relay Module (12V) ..... 29
Figure 2. 22: Buzzer and Buzzer Module ..... 30
Figure 2. 23: Pin Functions of Buzzer Module ..... 31
Figure 2. 24: SMPS (Switch Mode Power Supply) ..... 31
Figure 2. 25: 12V 10A Circuit Diagram of SMPS ..... 32
Figure 2. 26: Power Module Multi Output Voltage Conversion ..... 33
Figure 2. 27:  Multi Voltage  Output Power Module ..... 33
Figure 2. 28: 5V USB Output ..... 35
Figure 2. 29: Miniature  Circuit Breaker (MCB) ..... 36
Figure 2. 30: Electrical Dual Row Terminal Block ..... 37
Figure 2. 31: Connecting Wire ..... 38
Figure 2. 32: Jumper Wire ..... 38
Figure 2. 33: Lugs (Wire Connectors/Crimping Lugs) ..... 39

Figure 3. 1: Block Diagram of Power Section ..... 41
Figure 3. 2: Block Diagram of Controller Unit ..... 41
Figure 3. 3: Logo of Arduino IDE, Arduino IoT Cloud, Google Home ..... 42
Figure 3. 4: Circuit Diagram ..... 43
Figure 3. 5: Flow Chart ..... 45

# List of Table

Table 2. 1: Required Components ..... 6
Table 2. 2: Specification of ESP32 ..... 8
Table 2. 3: Pin Functions of ESP32 ..... 10
Table 2. 4: Specification of DHT11 ..... 13
Table 2. 5: Pin Functions of DHT11 Sensor Module ..... 13
Table 2. 6: Specification of MQ-2 ..... 16
Table 2. 7: Pin Functions of MQ-2 Sensor ..... 16
Table 2. 8: Pin Functions of MQ-2 Module ..... 17
Table 2. 9: Specifications of IR Receiver (TSOP1838) ..... 20
Table 2. 10: Pin Functions of IR Receiver (TSOP1838) ..... 20
Table 2. 11: Features of IR Remote ..... 23
Table 2. 12: Function Key of Remote ..... 25
Table 2. 13: Pin Mapping with ESP32 ..... 26
Table 2. 14: Specification of Relay Module (12V) ..... 28
Table 2. 15: Pin Functions of Relay Module (12V) ..... 28
Table 2. 16: Specifications of Buzzer Module ..... 30
Table 2. 17: Pin Functions of Buzzer Module ..... 30
Table 2. 18: Specifications of Multi Voltage Output Power Module ..... 34
Table 2. 19: Specifications of USB Output ..... 35

Table 4. 1: Cost Estimation of the project ..... 48

# CHAPTER 1

# INTRODUCTION

## 1.1 Introduction

In the modern world, technology is advancing rapidly and one of the most impactful areas is **smart home automation**. With the growing population and increasing demand for energy, the need for intelligent systems that can monitor and control electrical loads has become essential. A smart home is no longer a luxury but a necessity for reducing energy wastage, ensuring safety, and improving the quality of life.

The **IoT Based Multicontrolled Smart Home and Protection System** is designed to meet this demand by integrating Artificial Intelligence (AI) and the Internet of Things (IoT). This system enables users to monitor and control household appliances not only through smartphones and computers but also by infrared (IR) remote, manual switches, and voice assistants such as Google Home.

In addition to automation and control, the project also addresses safety concerns. A smoke and gas sensor is incorporated to detect hazardous conditions, and a buzzer alerts the occupants immediately. Thus, the system provides a comprehensive solution that combines **load control, monitoring, and protection** under a single platform.

## 1.2 Internet of Things (IoT)

The Internet of Things (IoT) is a revolutionary concept where everyday objects are connected to the internet and can communicate with each other without human intervention. IoT devices are equipped with sensors, processors, and communication modules that allow real-time monitoring and control. This makes IoT one of the fastest-growing technologies in automation, healthcare, agriculture, transportation, and smart homes.

In the context of smart homes, IoT allows household appliances such as lights, fans, security systems, and temperature sensors to be connected to the cloud. Users can access and control these devices remotely from anywhere in the world. For example, turning off a light in the living room while being at work, or monitoring energy usage in real-time.

This project uses the **ESP32 microcontroller** as the core IoT device. With built-in Wi-Fi and Bluetooth, ESP32 efficiently communicates with the Arduino IoT Cloud, ensuring smooth and reliable data transfer. This enables the system to check load conditions, energy consumption, and provide users with the flexibility of real-time monitoring.

## 1.3 AI and Smart Home Automation

Artificial Intelligence (AI) plays a key role in advancing smart home technology. AI enables devices to learn, adapt, and make intelligent decisions based on user behavior. One of the most popular uses of AI in homes is **voice-based automation** through assistants like Google Home or Amazon Alexa. By simply giving voice commands, users can turn on lights, operate fans, or check the condition of sensors in their homes.

In this project, AI integration ensures that the system is not just limited to manual or remote control but also provides a **hands-free user experience**. For example, instead of manually pressing a switch, a user can simply say, *"Hey Google, turn off the lights"*, and the system will execute the command through the IoT platform.

Moreover, AI can help optimize energy consumption by identifying unnecessary loads and suggesting power-saving actions. The combination of AI and IoT provides a futuristic approach to home automation, making homes smarter, safer, and more efficient.

## 1.4 Load Monitoring and Controlling

The system provides real-time load monitoring and multi-mode controlling. Using IoT integration, users can check the status of connected loads, track energy consumption, and remotely switch devices on or off. Local control is supported by manual switches and IR remote, while cloud and AI integration allow global monitoring and AI-assisted control. This ensures both **energy efficiency** and **operational flexibility**.

## 1.5 Problem Statement

In traditional households, offices, and industries, many appliances are often left running unnecessarily, causing energy wastage, higher costs, and safety risks. Manual control systems lack flexibility, remote monitoring, and intelligent automation. Moreover, conventional systems rarely include protective features such as smoke detection, which

increases vulnerability to hazards. Therefore, there is a strong need for a smart system that combines IoT, AI, and protection mechanisms to provide safe, efficient, and intelligent load management.

## 1.6 Motivation

The motivation for this project comes from the rising importance of **energy conservation and safety** in daily life. While exploring existing literature and research, it became clear that most IoT-based systems focus only on load monitoring or controlling but do not provide multiple modes of operation or safety features.

By integrating IoT with AI and protection mechanisms, our project introduces a system that is **multi-functional, user-friendly, and secure**. It reduces power wastage by allowing remote control, increases convenience with voice commands, and ensures safety with smoke detection and buzzer alarms.

Our vision is to design a system that not only benefits households but can also be extended to offices, factories, and industries. The project demonstrates how modern technology can create smarter and safer environments, motivating us to contribute toward the future of sustainable living.

## 1.7 Objectives

The goals and objectives of our designed system are as follows:

i. To design an **IoT based multicontrolled smart home system** that reduces energy wastage and ensures safety.

ii. To implement a circuit that allows **multi-mode control** (manual, IR remote, IoT Cloud, and AI voice control).

iii. To integrate sensors for **temperature, humidity, and smoke detection**, providing protection against hazards.

iv. To study and apply the concept of **IoT-based communication** between ESP32 and Arduino IoT Cloud.

To design a system that helps prevent **electrical hazards such as short circuits or fire** by adding safety features like buzzer alerts.

## 1.8 Research Methodology

In this project, we have designed a system that can check the status of loads (ON/OFF), monitor temperature and humidity, detect smoke, and trigger safety alarms. The ESP32 collects all the data from different sensors such as DHT11 and MQ-2. This information, including load condition and power consumption, is processed and then sent to the Arduino IoT Cloud via Wi-Fi.

Through the IoT platform, we can monitor and control the loads from anywhere using a smartphone, laptop, or computer. The system supports multiple controlling methods: manual switches, IR remote, IoT mobile application, and Google Home AI assistant.

During the development of this system, we studied different research papers, online resources, and technical documentation to understand IoT communication, cloud integration, and safety mechanisms. Our methodology ensures that the system is not only functional but also reliable, user-friendly, and protective.

## 1.9 Conclusion

In this chapter, we introduced the **IoT Based Multicontrolled Smart Home and Protection System**. The system combines IoT and AI to provide real-time monitoring, multi-mode control, and protection features such as smoke detection with buzzer alerts.

We discussed the problem statement, motivation, objectives, and research methodology that guided the project. This system is designed to reduce energy wastage, improve user convenience, and ensure safety. The following chapters will present the components, hardware development, and implementation in detail.

## 1.10 Project Outline

This project is organized as follows:

Chapter 1: Introduction

Discusses the background, problem statement, objectives, and motivation of the project.

Chapter 2: Analysis of the System Components

Describes the major hardware components used such as ESP32, sensors, relay module, buzzer, and power supply units.

Chapter 3: Hardware Development

Explains the circuit design, block diagrams, construction, and working principle of the system.

Chapter 4: Results and Discussion

Presents the outcomes of the project, system performance, advantages, limitations, and applications.

Chapter 5: Conclusion

Summarizes the project and provides possible future improvements.

# CHAPTER 2

# ANALYSIS OF THE SYSTEM COMPONENT

## 2.1 Introduction

The proposed system is the AI Based Multicontrolled Smart Home and Protection System, which combines IoT and AI technologies to provide intelligent monitoring, multi-mode controlling, and safety features. The system reduces unnecessary power consumption by monitoring load conditions and controlling appliances through multiple modes such as smartphone, computer, IR remote, manual switches, and Google Home voice assistant.

The system performs the following key functionalities:

- ✓ It monitors all load statuses (ON/OFF).
- ✓ It measures temperature, humidity, and detects smoke or gas leakage.
- ✓ It activates a buzzer alarm in case of smoke detection for safety.
- ✓ It allows multi-mode control (manual, IR remote, smartphone app, and AI assistant).

This makes the system a complete solution for both smart home automation and household protection.

## 2.2 Required Components

The **IoT Based Multicontrolled Smart Home and Protection System** has the following main components:

*Table 2. 1: Required Components*

| Hardware Components | Software Tools |
|---|---|
| ESP32 Development Board | Arduino IDE |
| ESP32 Terminal Adapter | Arduino IoT Cloud |
| DHT11 Temperature & Humidity Sensor | Google Home |
| MQ-2 Smoke & Gas Sensor | Google Assistant |
| IR Receiver Module | Gemini AI |
| IR Remote | - |
| Manual Switches (4 pcs) | - |

| | |
|---|---|
| Relay Module (4-channel) | - |
| Buzzer | - |
| SMPS (Switch Mode Power Supply) | - |
| DC to DC Buck Converter (12V → 5V) | - |
| USB Output Module | - |
| Circuit Breaker | - |
| Electrical Dual Row Terminal Block | - |
| Connecting Wires/ Jumper | - |
| Lugs (Wire Connectors) | - |
| Acrylic Plastic Sheets | - |

## 2.3 ESP32 Development Board

The ESP32 is a powerful and low-cost family of wireless microcontroller chips developed by **Espressif Systems**. It is widely used in **IoT-based applications** due to its integrated **Wi-Fi** and **Bluetooth** capabilities, dual-core processor, and versatile peripheral support. The ESP32 is considered the successor to the ESP8266, which gained massive popularity in the maker community. With its release in 2016, the ESP32 brought enhanced features including dual-core processing, Bluetooth connectivity, and improved performance, while keeping the price affordable.

The development board used in this project is a **Type-C ESP32 module** equipped with a **CP2102 USB-to-serial port chip** for stable communication and fast data transmission. This feature allows the board to be directly connected to a computer for debugging and programming. The board also includes two indicator LEDs: a **red power indicator** and a **blue working status indicator**, which provide real-time information about the operating condition of the module.

There are also **two touch buttons**:

- **Reset Button:** Used to restart the module.
- **Boot/Download Button:** Used to enter programming mode when uploading new firmware.

The ESP32 development board has pins aligned on both sides, which simplifies hardware connections during prototyping. It supports multiple development environments including **Arduino IDE**, **PlatformIO**, and **ESP-IDF**, making it accessible for beginners as well as professionals.

Due to its **dual Wi-Fi and Bluetooth functionality**, the ESP32 is capable of supporting a wide variety of IoT applications such as smart home automation, wireless industrial control, and positioning systems. In our project, the ESP32 is used as the **central controller** that manages all sensors and actuators, communicates with the Arduino IoT Cloud, and allows multi-mode control including manual switches, IR remote, smartphone, and Google Home assistant.

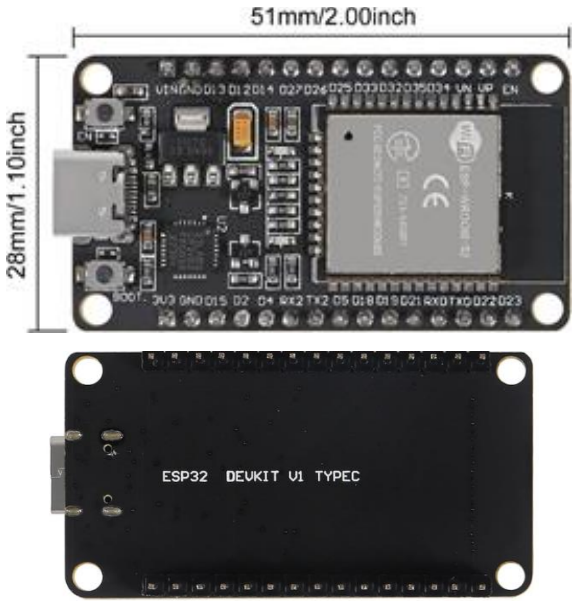



*Figure 2. 1: ESP32 Development Board*

### 2.3.1 Specification of ESP32

*Table 2. 2: Specification of ESP32*

| Specifications | Details |
|---|---|
| **Microprocessor** | Dual-core 32-bit Xtensa® LX6 CPU |
| **Clock Frequency** | 80 MHz to 240 MHz |
| **RAM** | 520 KB SRAM |
| **ROM** | 448 KB (bootloader and system firmware) |
| **Flash Memory** | 4 MB (default), expandable via SPI |

| | |
|---|---|
| **PSRAM (Optional)** | Supported in some variants |
| **Wi-Fi Standards** | IEEE 802.11 b/g/n (2.4 GHz) |
| **Bluetooth** | v4.2 BR/EDR and BLE |
| **Input Voltage** | 5V (USB/VIN) |
| **Operating Voltage** | 2.7 V – 3.6 V (typical 3.3V) |
| **Operating Temperature** | -40°C to +85°C |
| **GPIO Pins** | 34 Programmable |
| **Integrated Sensors** | Hall sensor, temperature sensor, capacitive touch sensor |
| **Peripheral Interfaces** | SPI, I2C, I2S, UART, SDIO, CAN, Ethernet, IR, PWM |
| **DAC Pins** | 2 (GPIO25, GPIO26) |
| **ADC Pins** | 12-bit resolution (up to 18 channels) |
| **RTC Support** | Real-Time Clock with ULP coprocessor |

### 2.3.2 Pin Functions of ESP32

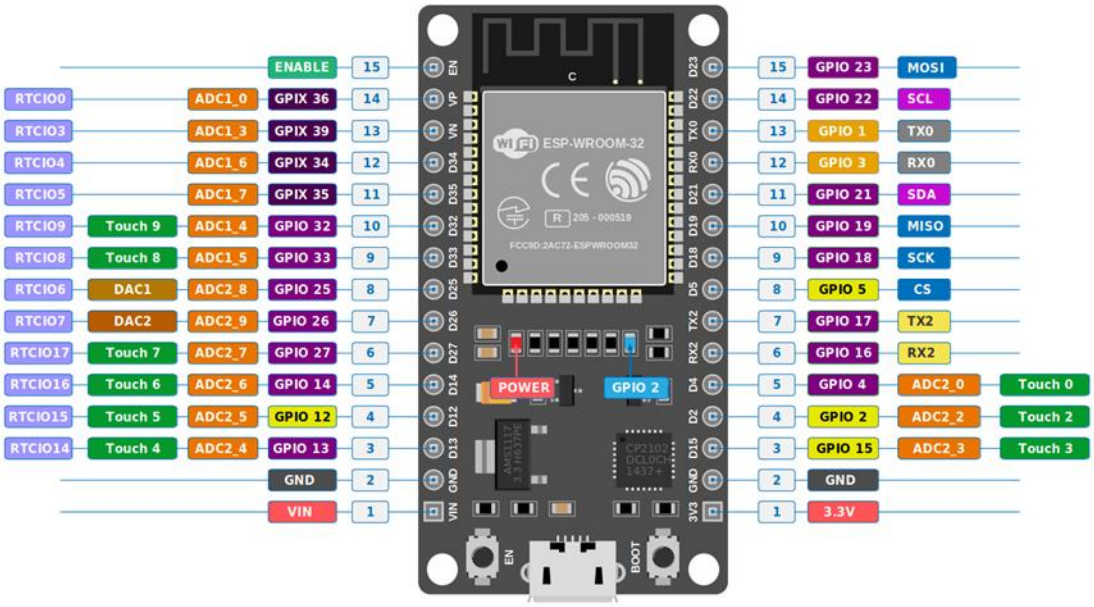



*Figure 2. 2: Pin Functions of ESP32*

*Table 2. 3: Pin Functions of ESP32*

| Pin No. | Pin Name | GPIO | Function / Notes |
|---|---|---|---|
| 1 | 3V3 | – | 3.3V regulated power output |
| 2 | GND | – | Ground |
| 3 | IO15 | GPIO15 | ADC2_CH3, HSPI_CS, Touch 3, PWM |
| 4 | IO2 | GPIO2 | ADC2_CH2, Touch 2, Boot (default LOW = flash mode), PWM |
| 5 | IO4 | GPIO4 | ADC2_CH0, Touch 0, PWM |
| 6 | IO16 | GPIO16 | UART2 RX, general-purpose I/O |
| 7 | IO17 | GPIO17 | UART2 TX, general-purpose I/O |
| 8 | IO5 | GPIO5 | VSPI_CS, PWM |
| 9 | IO18 | GPIO18 | VSPI_CLK, PWM |
| 10 | IO19 | GPIO19 | VSPI_MISO, PWM |
| 11 | IO21 | GPIO21 | I²C SDA |
| 12 | RX0 | GPIO3 | UART0 RX (default serial input) |
| 13 | TX0 | GPIO1 | UART0 TX (default serial output) |
| 14 | IO22 | GPIO22 | I²C SCL |
| 15 | IO23 | GPIO23 | VSPI_MOSI, PWM |
| 16 | EN | – | Enable / Reset pin (active HIGH) |
| 17 | IO36 (VP) | GPIO36 | ADC1_CH0, Input only |
| 18 | IO39 (VN) | GPIO39 | ADC1_CH3, Input only |
| 19 | IO34 | GPIO34 | ADC1_CH6, Input only |
| 20 | IO35 | GPIO35 | ADC1_CH7, Input only |
| 21 | IO32 | GPIO32 | ADC1_CH4, Touch 9, PWM |
| 22 | IO33 | GPIO33 | ADC1_CH5, Touch 8, PWM |
| 23 | IO25 | GPIO25 | DAC1, ADC2_CH8, PWM |
| 24 | IO26 | GPIO26 | DAC2, ADC2_CH9, PWM |
| 25 | IO27 | GPIO27 | ADC2_CH7, Touch 7, PWM |
| 26 | IO14 | GPIO14 | HSPI_CLK, ADC2_CH6, Touch 6, PWM |

| Pin No. | Pin Name | GPIO | Function / Notes |
|---|---|---|---|
| 27 | IO12 | GPIO12 | HSPI_MISO, ADC2_CH5, Touch 5, PWM |
| 28 | GND | – | Ground |
| 29 | IO13 | GPIO13 | HSPI_MOSI, ADC2_CH4, Touch 4, PWM |
| 30 | VIN | – | Input voltage (5V via USB or external supply) |

## 2.4 ESP32 Terminal Adapter

The ESP32 Terminal Adapter is a specially designed breakout board that allows easy interfacing of the ESP32 development board with external modules and sensors. It provides screw terminals for each GPIO pin of the ESP32, which makes wiring simple, secure, and reliable during hardware development.

Normally, connecting multiple sensors, relays, or external circuits to the ESP32 using jumper wires becomes messy and error-prone. The terminal adapter solves this issue by providing a well-organized structure where each ESP32 pin is extended to a terminal block. This makes it ideal for prototyping, testing, and permanent installations in IoT and smart home applications.

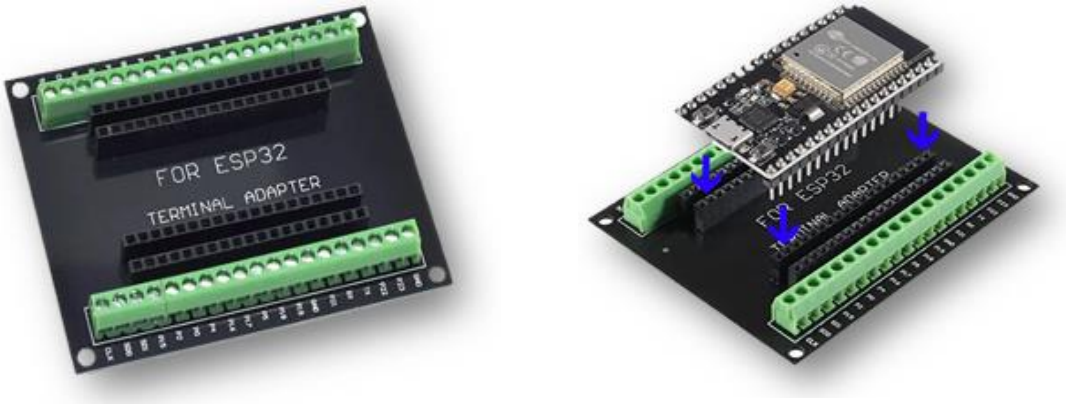



*Figure 2. 3: ESP32 Terminal Adapter*

### 2.4.1 Purpose and Usage in Our Project

In our project, the ESP32 Terminal Adapter was used to simplify the connection of the ESP32 development board with multiple sensors and modules. Since the system integrates various input and output devices such as: (DHT11, MQ-2, IR receiver module and IR

remote, Manual switches, Relay module Buzzer for alarm system, Power supply connections etc. )

Managing these connections directly on the ESP32 board would have been difficult and prone to wiring errors. The terminal adapter provided an organized and secure wiring interface by breaking out each pin into a labeled screw terminal.

This ensured:

- ✓ **Neat Wiring:** All wires from sensors, relays, and switches were connected neatly without loose jumper wires.
- ✓ **Error-Free Connections:** Labeled terminals reduced the chances of wrong pin connections.
- ✓ **Easy Debugging:** During testing and troubleshooting, modules could be connected or disconnected quickly.
- ✓ **Reliable Power Distribution:** Multiple devices could share stable 3.3V, 5V, and GND terminals without loose wiring.

Thus, the ESP32 terminal adapter played a crucial role in building a stable, reliable, and easily maintainable hardware system for the AI Based Multicontrolled Smart Home and Protection System.

## 2.5 DHT11 Temperature & Humidity Sensor

The **DHT11** is a low-cost digital sensor used for measuring **temperature and humidity**. It can be easily interfaced with microcontrollers such as **Arduino, ESP32, and Raspberry Pi**, and provides reliable environmental data in real-time.

The DHT11 is available in two forms:

- **As a raw sensor**, and
- **As a module** (which includes a pull-up resistor and power LED).

The DHT11 sensor is available in two forms: as a raw sensor and as a module. In both cases, the performance remains the same. The raw sensor version comes in a 4-pin package, but only three pins (VCC, GND, and Data) are actually used in operation. On the other hand, the module version is designed more conveniently with three pins only, since it already includes the required pull-up resistor and sometimes a power indicator LED.

For our project, using the module form made the connection with the ESP32 much easier and reliable.

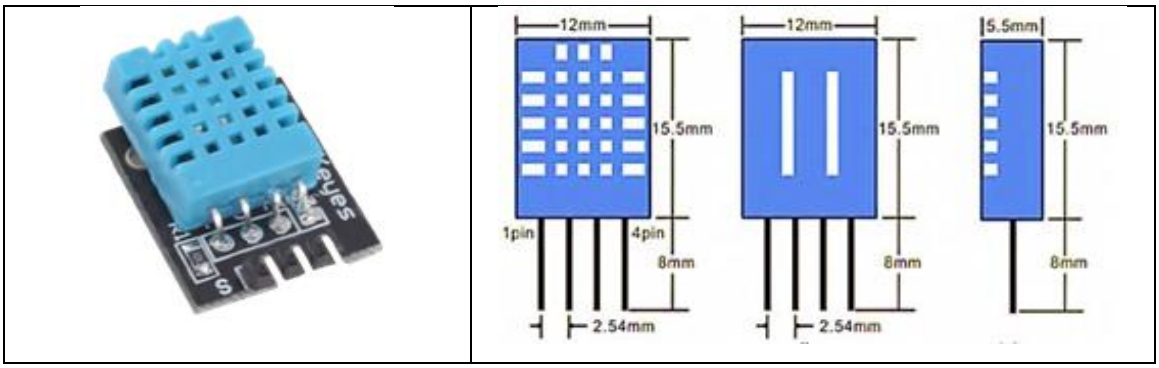



*Figure 2. 4: DHT11 Temperature & Humidity Sensor*

### 2.5.1 Specification of DHT11

*Table 2. 4: Specification of DHT11*

| Parameter | Specification |
|---|---|
| Operating Voltage | 3.5V – 5.5V |
| Operating Current | 0.3 mA (measuring), 60 µA (standby) |
| Output Type | Digital serial data |
| Temperature Range | 0°C to 50°C |
| Humidity Range | 20% – 90% RH |
| Accuracy | ±1°C (Temperature), ±1% RH (Humidity) |
| Sampling Rate | 1 Hz (1 reading per second) |
| Number of Pins | 3 (Module) / 4 (Raw sensor) |

### 2.5.2 Pin Functions of DHT11 Sensor Module

*Table 2. 5: Pin Functions of DHT11 Sensor Module*

| Pin No. | Pin Name | Description |
|---|---|---|
| 1 | VCC | Power supply (3.5V – 5.5V) |
| 2 | Data | Digital output – provides both temperature and humidity data through serial communication |
| 3 | GND | Ground connection of the circuit |

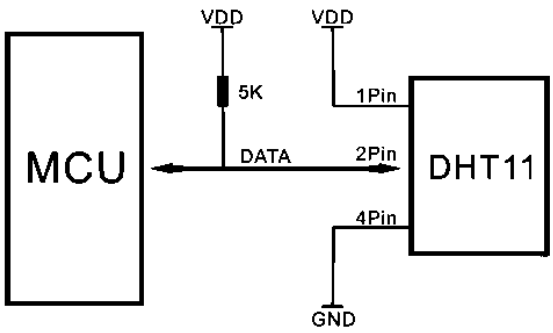



*Figure 2. 5: Pin Functions of DHT11 Sensor Module*

### 2.5.3 Working Principle of DH11

In our project, the **data pin** of the DHT11 sensor is connected to one of the I/O pins of the ESP32 through a **5kΩ pull-up resistor**. This single pin sends both **temperature** and **humidity** values in serial digital format, which makes the interfacing very simple.

Since we used an ESP32 board, the process became easier because ready-made **Arduino libraries** are available for DHT11. These libraries helped us to read the temperature and humidity data quickly without implementing the entire protocol manually.

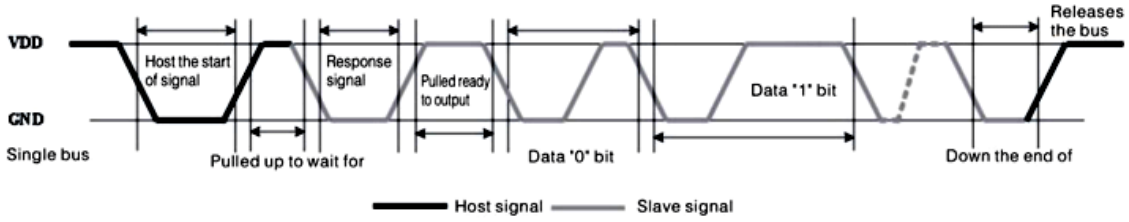



*Figure 2. 6: Working Principle of DH11*

The data received from the sensor always follows a fixed **40-bit format**:

- 8-bit humidity integer data
- 8-bit humidity decimal data
- 8-bit temperature integer data
- 8-bit temperature decimal (fractional) data
- 8-bit parity bit (checksum)

To request this data, the ESP32 pulls the data line **LOW for a short duration** and then keeps it **HIGH**, after which the DHT11 responds and transmits the data sequence.

This feature allowed our project to continuously monitor **room temperature and humidity** and update the values to the **Arduino IoT Cloud**, so that we could view them in real time on a smartphone or computer.

## 2.6 MQ-2 Smoke & Gas Sensor

The MQ-2 is a low-cost sensor module widely used for detecting combustible gases such as **LPG, propane, methane, alcohol, hydrogen, and smoke**. The sensor works on the principle that its electrical conductivity changes when exposed to gas molecules. This variation in conductivity can be converted into a measurable signal that represents the concentration of the gas**.**

The module comes with two types of outputs:

- **Analog Output (A0):** Provides variable voltage proportional to gas concentration.
- **Digital Output (D0):** Provides a HIGH/LOW signal based on a user-defined threshold (set using the onboard potentiometer).

It also features an adjustable potentiometer for setting sensitivity levels, making it flexible for different applications. With a detection range of 300–10,000 ppm, the MQ-2 is suitable for home and industrial use.

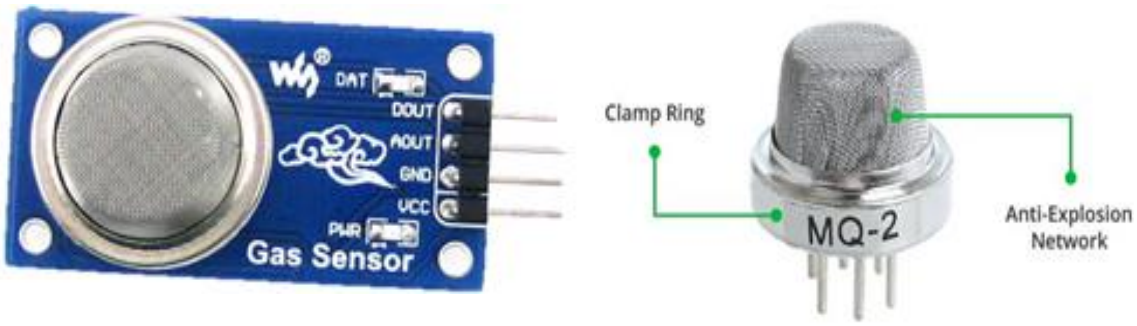



*Figure 2. 7: MQ-2 Smoke & Gas Sensor*

In our **IoT Based Multicontrolled Smart Home and Protection System**, the MQ-2 sensor was integrated to enhance safety protection. When harmful gases or smoke are detected, the sensor immediately triggers the buzzer alarm and sends real-time data to the ESP32 and Arduino IoT Cloud, allowing users to monitor the situation remotely.

### 2.6.1 Specification of MQ-2

*Table 2. 6: Specification of MQ-2*

| Parameter | Specification |
|---|---|
| **Sensor Type** | Semiconductor |
| **Standard Encapsulation** | Black Bakelite |
| **Detection Gas** | Combustible gases and smoke (LPG, Propane, Hydrogen, Methane, Alcohol, CO, etc.) |
| **Concentration Range** | 300 – 10,000 ppm |
| **Loop Voltage** | ≤ 24V DC |
| **Heater Voltage** | 5.0V ± 0.2V (AC or DC) |
| **Heater Resistance** | 31Ω ± 3Ω (at Room Temperature) |
| **Heater Consumption** | ≤ 900 mW |
| **Sensing Resistance** | 2 kΩ – 20 kΩ (in 2000 ppm $C_3H_8$) |
| **Sensitivity** | Rs (in air) / Rs (1000 ppm isobutane) ≥ 5 |
| **Operating Condition** | 20°C ± 2°C ; 65% ± 5% RH |
| **Load Resistance** | Adjustable |
| **Output** | Analog (A0) and Digital (D0) |
| **Special Features** | High sensitivity, low cost, simple drive circuit, long lifespan |

### 2.6.2 Pin Functions of MQ-2

*Table 2. 7: Pin Functions of MQ-2 Sensor*

| Pin No: | Pin Name: | Description (MQ-2 Sensor) |
|---|---|---|
| 1 | H-Pins | Out of the two H pins, one pin is connected to supply and the other to ground |
| 2 | A-Pins | The A pins and B pins are interchangeable. These pins will be tied to the Supply voltage. |
| 3 | B-Pins | The A pins and B pins are interchangeable. One pin will act as output while the other will be pulled to ground. |

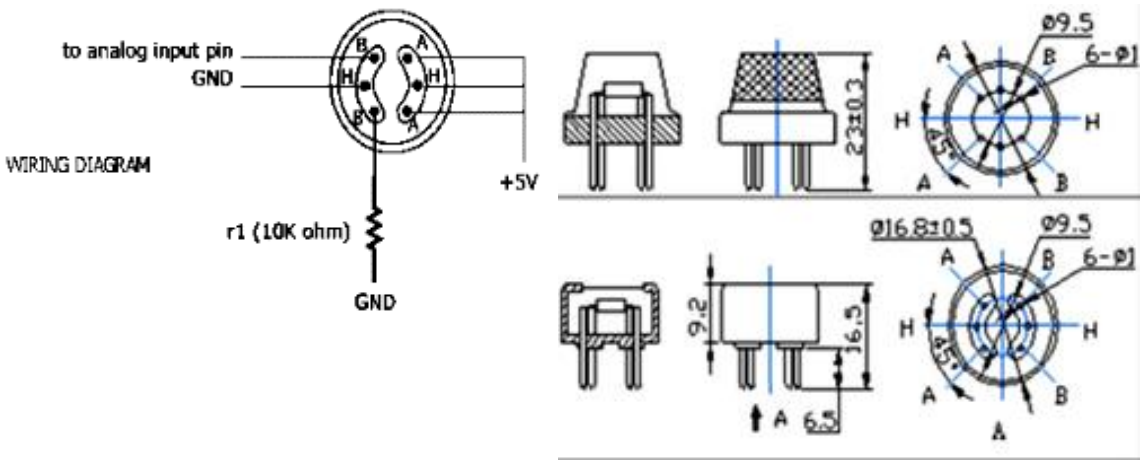



*Figure 2. 8: Pin Functions of MQ-2*

*Table 2. 8: Pin Functions of MQ-2 Module*

| Pin No: | Pin Name: | Description (MQ-2 Sensor Module) |
|---|---|---|
| 1 | Vcc | This pin powers the module, typically the operating voltage is +5V |
| 2 | Ground | Used to connect the module to system ground |
| 3 | Digital Out | You can also use this sensor to get digital output from this pin, by setting a threshold value using the potentiometer |
| 4 | Analog Out | This pin outputs 0–5V analog voltage based on the intensity of the gas |

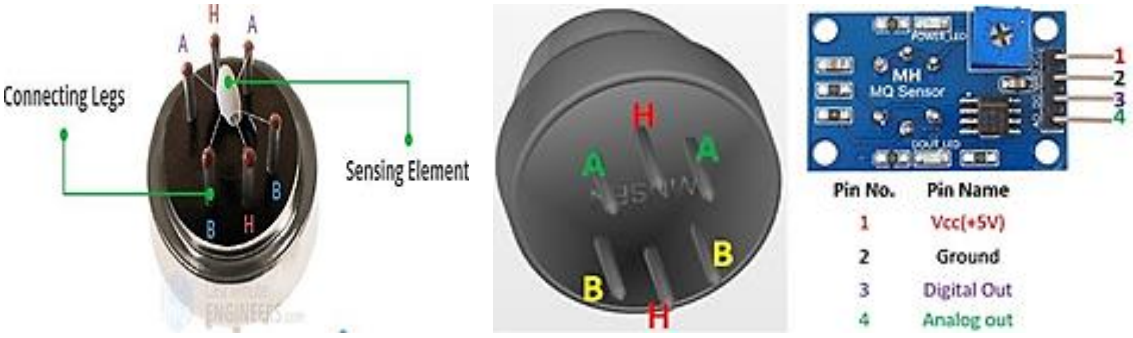



*Figure 2. 9: Pin Functions of MQ-2 Module*

### 2.6.3 Working Principle of MQ-2

When the MQ-2 gas sensor is powered on, it first **heats up its sensing layer (Tin Dioxide, $SnO_2$)** using the built-in heater coil. At this high temperature, **oxygen molecules from the air** attach to the surface of the $SnO_2$. These oxygen molecules trap free electrons, which creates a region of **high resistance** inside the sensor. As a result, only a very small current can flow when the air is clean.

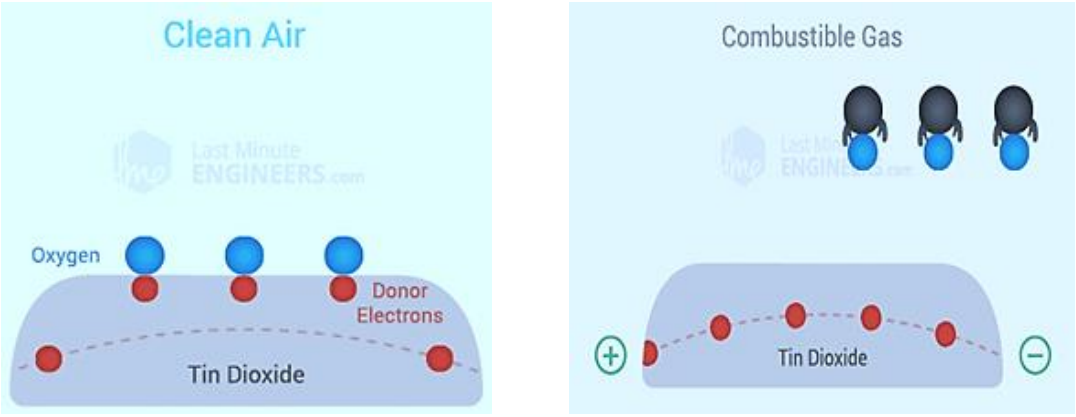



*Figure 2. 10: Current flow process in MQ2*

When **smoke or combustible gases** (like LPG, methane) enter the sensor, they react with the oxygen molecules on the $SnO_2$ surface. This reaction **releases the trapped electrons back into the material**, which reduces resistance and allows more current to flow.

The MQ-2 module converts this change into an output signal. In our project, we only use the **D0 pin (Digital Output)**. This pin gives a **HIGH signal (5V)** whenever the gas level goes above the preset limit (set by the onboard potentiometer), and stays **LOW (0V)** when the air is clean.

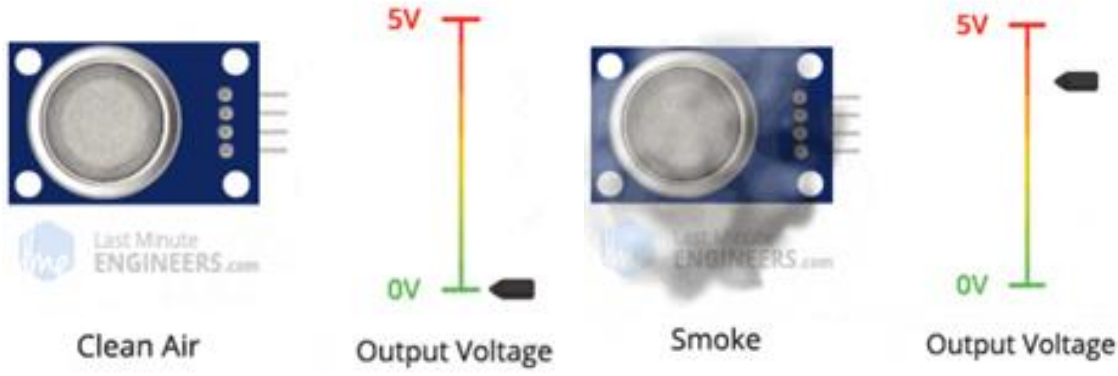



*Figure 2. 11: Output voltyage in MQ2 Sensor Module*

- **Step 1: Preheating**
- **Step 2: Gas Detection**
- **Step 3: Output Signal (D0 Pin)**
- **Step 4: Usage in Our Project**

In our **AI Based Multicontrolled Smart Home Automation and Protection System**:

- The **D0 pin** of MQ-2 is directly connected to the ESP32.
- Whenever smoke or gas leakage is detected above the set limit, the ESP32 immediately **activates the buzzer alarm**.
- This ensures quick **local safety response** for fire or gas leakage.

## 2.7 IR Receiver (TSOP1838)

In our project **IoT Based Multicontrolled Smart Home Automation and Protection System**, we used an **IR Receiver Module (TSOP1838)** as one of the key input devices. This small component is designed to detect the infrared (IR) signals sent by a remote control and convert them into a form that can be processed by the microcontroller.

The working idea is simple – whenever we press a button on the IR remote, it sends out a unique infrared light pattern. The IR receiver picks up this signal, filters it, and outputs a clean digital pulse. With the help of the Arduino IRremote library, we can decode these signals into **HEX codes**. Each HEX code corresponds to a specific remote button, and in our project, these codes are directly mapped to different load-controlling functions.

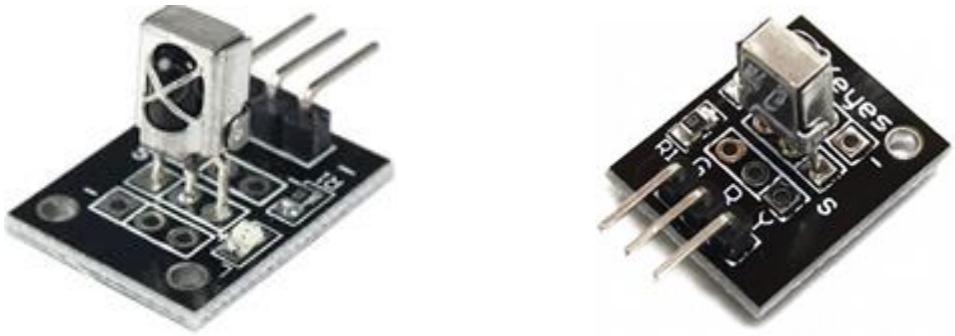


*Figure 2. 12: IR Receiver (TSOP1838)*

We selected this sensor because it is **cheap, easily available, and reliable** for indoor control. Also, it has a built-in filter for noise reduction, which makes it more accurate. This makes the IR receiver a perfect choice for adding **wireless manual control** alongside other control methods like IoT Cloud and manual switches.

### 2.7.1 Specifications of IR Receiver (TSOP1838)

*Table 2. 9: Specifications of IR Receiver (TSOP1838)*

| Parameter | Value / Description |
|---|---|
| **Model** | TSOP1838 IR Receiver |
| **Operating Voltage** | 2.7V – 5.5V (typically 5V in our project) |
| **Current Consumption** | 0.4 mA – 1.5 mA |
| **Carrier Frequency** | 38 kHz |
| **Reception Distance** | Up to 8–10 meters |
| **Viewing Angle** | ±45° |
| **Output Signal** | Active LOW (Digital) |
| **Response Time** | < 200 µs |
| **Ambient Light Immunity** | Works reliably even under sunlight/fluorescent light |
| **Package** | Compact 3-pin epoxy package (5 mm) |

### 2.7.2 Pin Functions of IR Receiver (TSOP1838)

*Table 2. 10: Pin Functions of IR Receiver (TSOP1838)*

| Pin No. | Pin Name | Description |
|---|---|---|
| 1 | **OUT** | Digital output pin. Goes **LOW** when IR signal is received.<br>This pin is connected to ESP32 for decoding remote signals. |
| 2 | **GND** | Ground pin.<br>Connected to the common ground of the circuit. |
| 3 | **VCC** | Power supply pin.<br>Connected to +5V in our project. |




*Figure 2. 13: Pin Functions of IR Receiver (TSOP1838)*

### 2.7.3 Working Process of IR Receiver (TSOP1838)

The IR Receiver (TSOP1838) is the main component that allows our **AI Based Multicontrolled Smart Home Automation and Protection System** to receive commands from the remote control. Its job is quite simple but very effective—it takes the invisible infrared light signals coming from the remote and converts them into digital signals that the ESP32 can understand.

Here's how it actually works step by step:

1. **Receiving the IR Signal:** When I press any button on the IR remote, it sends out a series of infrared light pulses, usually modulated at 38 kHz. The small photodiode inside the TSOP1838 catches this IR light and immediately converts it into a tiny electrical signal.
2. **Noise Reduction and Amplification:** The signal at this stage is very weak and can easily be disturbed by sunlight, fluorescent lamps, or other light sources. To handle this, the module has an **Automatic Gain Control (AGC)** circuit. It strengthens the weak remote signal while filtering out the unnecessary background noise.
3. **Bandpass Filtering:** After amplification, the signal passes through a **bandpass filter** that only allows signals close to 38 kHz to pass. This step ensures that the receiver only responds to the remote and not to random IR light around the room.
4. **Demodulation**: The actual command information is hidden inside the 38 kHz pulses. The **demodulator circuit** inside the TSOP1838 removes the carrier frequency and extracts only the useful data. In simple words, it separates the "message" from the "carrier wave."
5. **Output to Microcontroller:** Finally, the clean digital signal comes out through the **OUT pin** of the receiver. Normally, the output pin stays HIGH, but when a valid signal arrives, it goes LOW in a specific sequence of pulses. Each button on the remote generates a unique sequence, which the ESP32 reads and converts into a HEX code.

### 2.7.4 Block Diagram of TSOP1838 IR Reciver

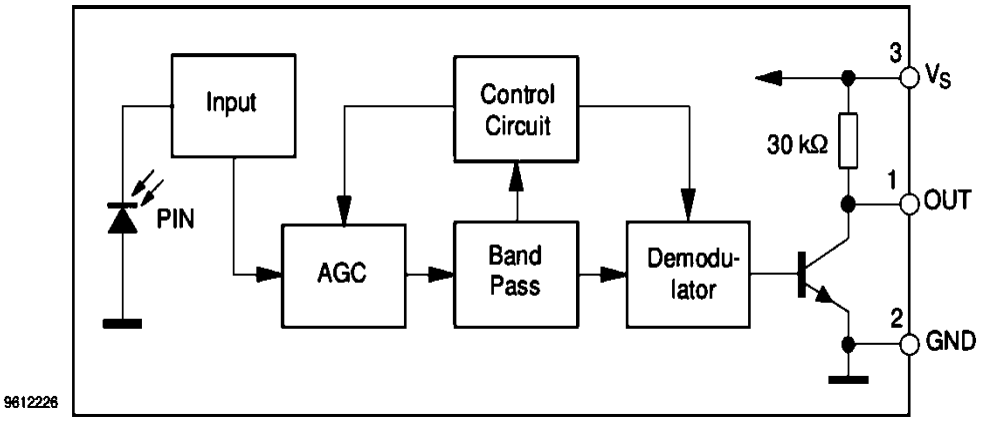



*Figure 2. 14: Block Diagram of TSOP1838 IR Reciver*

## 2.8 IR Remote

An **Infrared (IR) Remote** is a compact wireless controller that transmits data using modulated infrared light signals (typically 38 kHz). Each button on the remote generates a unique **HEX code**, which is received by an IR Receiver module (like TSOP1838). The microcontroller then decodes this signal and performs the assigned task.



*Figure 2. 15: IR Remote*

In our project **"AI Based Multicontrolled Smart Home Automation and Protection System"**, the IR remote is used to provide **offline device control**, allowing users to operate loads (lights, fans, etc.) without internet dependency. This makes the system **low-cost, reliable, and easy to implement** while maintaining technical accuracy in command transmission.

### 2.8.1 Features of IR Remote

*Table 2. 11: Features of IR Remote*

| Feature | Description |
|---|---|
| **Communication Type** | Infrared light, modulated at 38 kHz |
| **Number of Keys** | 21 buttons, each with a unique HEX code |
| **Operating Distance** | Greater than 8 meters |
| **Dimensions** | 84 mm × 39 mm (compact size) |
| **Power Supply** | 3V button cell battery |
| **Reliability** | Stable communication with IR receivers like TSOP1838 |
| **Cost** | Low-cost and widely available |
| **Application** | Used for wireless control in Arduino/ESP32 based projects |

### 2.8.2 HEX Code Decoding Process

To integrate the **IR remote** into our *AI Based Multicontrolled Smart Home and Protection System*, we first needed to decode the unique HEX codes generated by each button. These codes act as identifiers and are later mapped to specific relay operations in our project.

***Step 1: Components Used***

- TSOP 1838 IR Receiver Module
- Arduino Uno
- IR Remote
- USB cable

***Step 2: Wiring Connections***

- *VCC → 5V*
- GND → GND
- OUT → D11

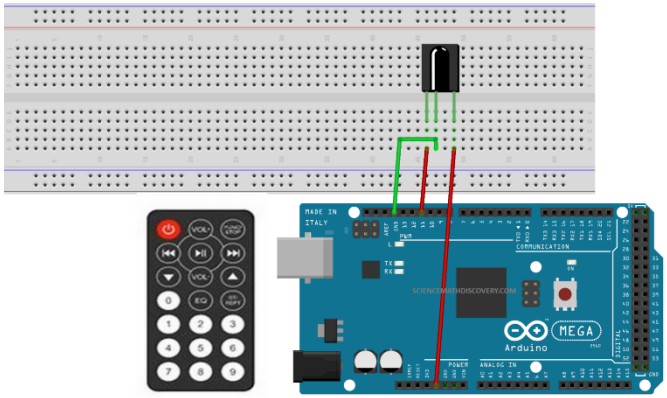



*Figure 2. 16: Wiring Connections  Decoding HEX Code*

***Step 3: Installing the IRremote Library***

- Open Arduino IDE → Sketch → Include Library → Manage Libraries
- Search for IRremote (by shirriff) → Install

***Step 4: Test Code for Decoding***

We uploaded the following simple code to capture button HEX values:

```
#include <IRremote.h>

const int RECV_PIN = 11;      // HX1838 OUT → D11 (Arduino)

IRrecv irrecv(RECV_PIN);

decode_results results;

void setup() {

  Serial.begin(9600);

  irrecv.enableIRIn(); // Start receiver

}

void loop() {

  if (irrecv.decode(&results)) {

    Serial.println(results.value, HEX); // Print HEX code

    irrecv.resume(); // Ready for next code
  }
}
```

***Step 5: Reading HEX Codes***

- Upload code → Open Serial Monitor (9600 baud)
- Press each button on the remote → Note the HEX values displayed
- Prepare a mapping table (Button → HEX Code).

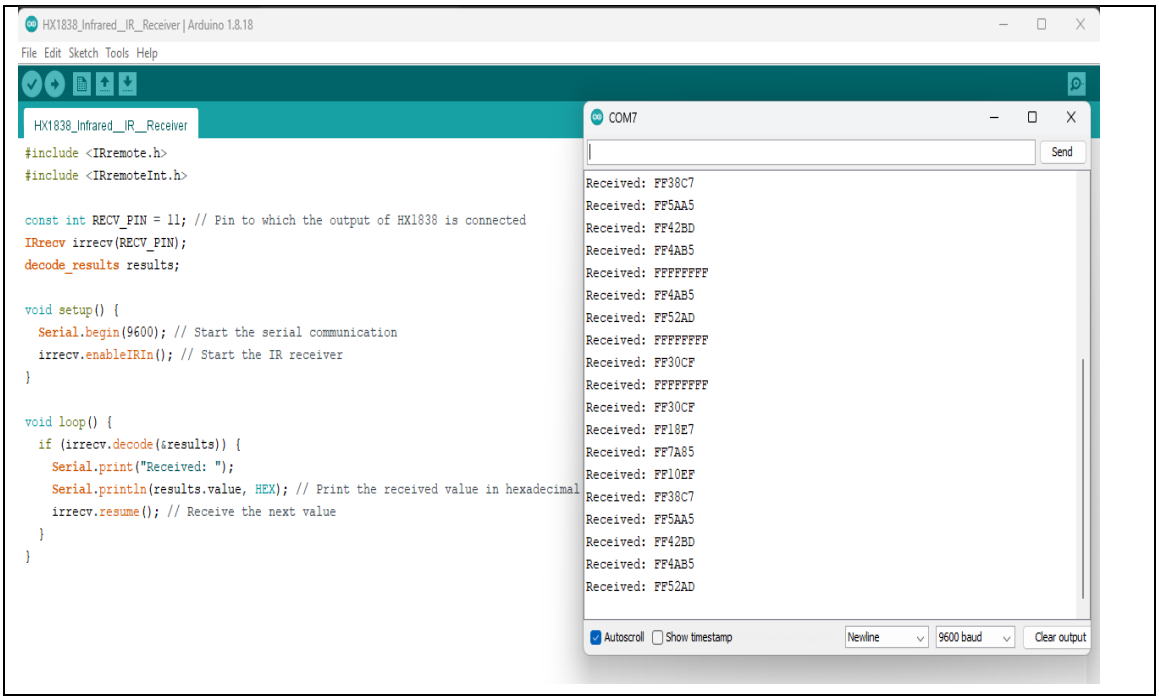



*Figure 2. 17: Reading HEX Codes*

### 2.8.3 Functions of Remote Keys

In our **AI Based Multicontrolled Smart Home Automation and Protection System**, specific remote keys are assigned to control different loads. Each button generates a **unique HEX code**, which the ESP32 decodes and maps to a function.

Example implementation in our project:

```
switch (results.value) {
  case 0xFF30CF: toggleRelay(1, light1); break;  // Remote key “1”
  case 0xFF18E7: toggleRelay(2, light2); break;  // Remote key “2”
  case 0xFF7A85: toggleRelay(3, light3); break;  // Remote key “3”
  case 0xFF10EF: toggleRelay(4, light4); break;  // Remote key “4”
  case 0xFF6897: toggleAllLights();      break; // Remote key “0” = All
}
```

*Table 2. 12: Function Key of Remote*

| Button | HEX Code (Example) | Function in Project |
|---|---|---|
| 1 | 0xFF30CF | Toggle Relay 1 (Light 1) |
| 2 | 0xFF18E7 | Toggle Relay 2 (Light 2) |
| 3 | 0xFF7A85 | Toggle Relay 3 (Light 3) |
| 4 | 0xFF10EF | Toggle Relay 4 (Light 4) |
| *0* | 0xFF6897 | Toggle **All Loads** (ON/OFF) |

## 2.9 Manual Switches

In our project, we used manual switches as a backup control option along with IoT and remote control. These switches allow us to operate home appliances directly without depending on Wi-Fi, smartphone, or IR remote. This makes the system more reliable in case of internet failure or technical issues.



*Figure 2. 18: Manual Switches*

### 2.9.1 Switching Mechanism in Project

Each switch is connected to the input pins of the ESP32 microcontroller. When we press a switch:

- The input pin goes HIGH .
- The ESP32 detects this change.
- Based on the input, the microcontroller toggles the corresponding relay, which turns the appliance ON or OFF.

So, the switch works as a **digital input device** that gives simple ON/OFF signals to the controller.

### 2.9.2 Pin Mapping with ESP32

We used 4 manual switches in this project. Each switch is mapped to a separate GPIO pin of the ESP32.

*Table 2. 13: Pin Mapping with ESP32*

| Switch | Function | GPIO Pin (ESP32) | Controlled Device |
|---|---|---|---|
| SW1 | Light 1 Control | GPIO 21 | Room Light |
| SW2 | Light 2 Control | GPIO 19 | Fan |
| SW3 | Light 3 Control | GPIO 18 | Socket/Plug Point |
| SW4 | Light 4 Control | GPIO 5 | Emergency Light/Buzzer |

## 2.10 Relay Module (4-channel)

A relay is an electromagnetic switch that allows a low-voltage circuit to control a high-voltage or high-current load. It mainly consists of a coil, a movable armature, and a set of contacts. When a small control voltage is applied to the coil, it generates a magnetic field that pulls the armature, causing the contacts to switch between **Normally Open (NO)** and **Normally Closed (NC)** positions. This switching mechanism makes relays very useful for isolating low-power control circuits from high-power loads.

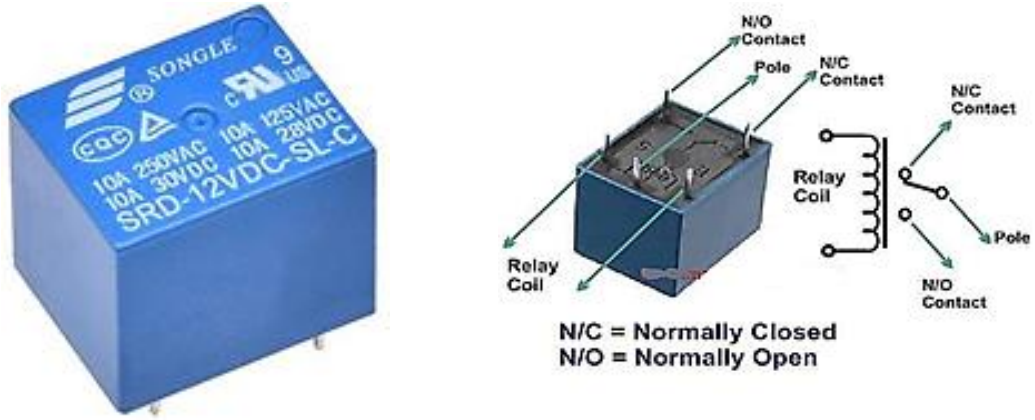



*Figure 2. 19: Relay*

Relays are commonly available in different voltage ratings such as **5V, 9V, and 12V DC**, depending on the coil voltage. In our case, we are using a **12V DC 4-channel relay module**, which means it requires a 12V DC supply to energize the relays.

A **relay module** is simply a ready-to-use circuit board that contains one or more relays along with driver circuits, optocouplers, indicator LEDs, and screw terminals for easy connection. This makes it safer and easier to use with microcontrollers like **ESP32, Arduino, and Raspberry Pi**, as the module already includes the necessary isolation and protection.

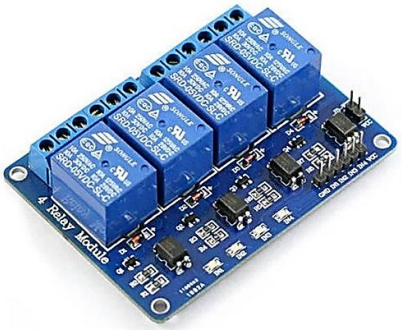



*Figure 2. 20: (4 chanel Relay Module)*

In our **IoT Based Multicontrolled Smart Home Automation and Protection System**, we are using a **12V 4-channel relay module**. Each channel of this module can switch up to **250V AC at 10A** or **30V DC at 10A**, which is more than enough to handle common home appliances like lights, fans, or small electrical devices. The relays are controlled by the ESP32, which sends low-voltage signals to activate them. This allows us to safely control high-voltage loads while keeping our control circuit protected.

### 2.10.1 Specification of Relay Module (12V)

*Table 2. 14: Specification of Relay Module (12V)*

| Parameter | Specification |
|---|---|
| **Operating Voltage (Relay)** | 12V DC |
| **Trigger Voltage (Logic)** | 5V DC (from microcontroller, e.g., ESP32/Arduino) |
| **Relay Channels** | 4 (independent relays) |
| **Maximum Switching Voltage** | AC 250V / DC 30V |
| **Maximum Switching Current** | 10A |
| **Control Input Pins** | IN1, IN2, IN3, IN4 (Active LOW) |
| **Coil Resistance** | ~400 Ω (per relay) |
| **Power Consumption** | ~900 mW (per relay when active) |
| **LED Indicators** | One red LED per channel (ON when relay is active) |
| **Output Terminals** | COM, NO, NC (per relay) |
| **Dimensions** | ~75mm × 55mm × 19mm |
| **Weight** | ~58 g |
| **PCB Color** | Blue |

### 2.10.2 Pin Functions of Relay Module (12V)

*Table 2. 15: Pin Functions of Relay Module (12V)*

| Pin Name | Description |
|---|---|
| **JD-VCC** | Powers the relay coils (typically 12V DC). Provides isolation between logic and relay circuit. |
| **VCC** | Powers the optocoupler and logic circuit (5V supply from microcontroller). |

| | |
|---|---|
| **GND** | Common ground for both relay module and control circuit. |
| **IN1** | Control signal for Relay 1 (LOW = Active, HIGH = Inactive). |
| **IN2** | Control signal for Relay 2 (LOW = Active, HIGH = Inactive). |
| **IN3** | Control signal for Relay 3 (LOW = Active, HIGH = Inactive). |
| **IN4** | Control signal for Relay 4 (LOW = Active, HIGH = Inactive). |
| **COM (x4)** | Common terminal of each relay switch. |
| **NO (x4)** | Normally Open contact – load is OFF by default, closes when relay is activated. |
| **NC (x4)** | Normally Closed contact – load is ON by default, opens when relay is activated. |

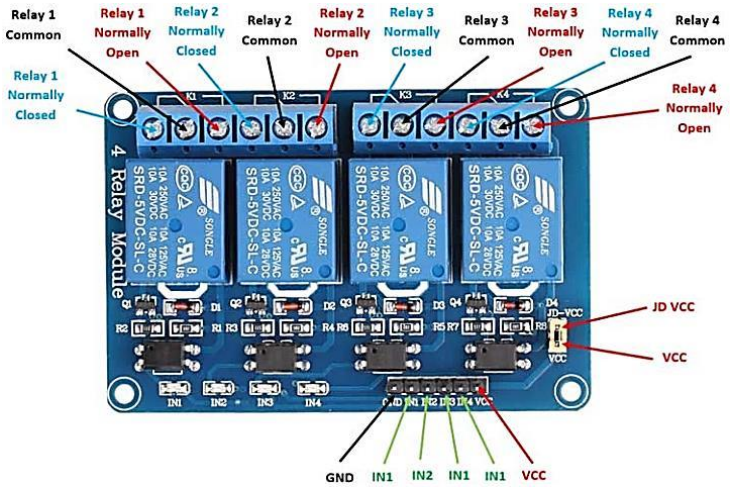



*Figure 2. 21: Pin Functions of Relay Module (12V)*

## 2.11 Buzzer Module

A buzzer is a simple audio signaling device commonly used in electronic projects to provide sound alerts. It is mainly of two types – **active buzzer** and **passive buzzer**. An **active buzzer** has an internal oscillator and produces sound when powered directly, while a **passive buzzer** requires an external signal (square wave) to generate sound.

And the buzzer module is a compact electronic component used to generate sound alerts in microcontroller-based projects. Unlike a raw piezo buzzer, the **buzzer module** comes with a built-in driving circuit (transistor and resistor), making it easy to connect directly with microcontrollers like ESP32 or Arduino.




*Figure 2. 22: Buzzer and Buzzer Module*

In our **IoT Based Multicontrolled Smart Home Automation and Protection System**, we used a **digital active buzzer module**. This module works as a safety alarm that activates whenever the MQ-2 smoke & gas sensor detects smoke or gas leakage. The buzzer provides an immediate local warning, ensuring users are alerted even if the IoT or IR system is not being monitored.

### 2.11.1 Specifications of Buzzer Module

*Table 2. 16: Specifications of Buzzer Module*

| Parameter | Specification |
|---|---|
| Operating Voltage | 3.3V – 5V DC |
| Operating Current | Low power consumption |
| Buzzer Type | Active (with internal oscillator) |
| Drive Circuit | 9012 transistor |
| Frequency Range | 2 kHz – 5 kHz (typical sound output) |
| Dimensions | ~3.3 cm × 1.3 cm PCB |
| Output | Continuous tone when powered |
| Compatibility | Directly works with 3.3V or 5V microcontrollers |

### 2.11.2 Pin Functions of Buzzer Module

*Table 2. 17: Pin Functions of Buzzer Module*

| Pin | Name | Description |
|---|---|---|
| 1 | VCC | Power supply (3.3V – 5V) |
| 2 | GND | Ground connection |
| 3 | I/O | Control pin (signal from microcontroller GPIO) |




*Figure 2. 23: Pin Functions of Buzzer Module*

### 2.11.3 Working Process of Buzzer Module

The buzzer module works by converting the control signal into an audible alarm. Since it is an **active buzzer module**, it only requires a simple HIGH/LOW signal to operate.

In our project, the **I/O pin** of the buzzer module is connected to a GPIO pin of the ESP32. When the MQ-2 sensor detects smoke or gas above a threshold, the ESP32 sends a HIGH signal to the buzzer's I/O pin. The module's built-in driver transistor then powers the buzzer, producing a continuous tone.

This makes the buzzer an important **safety feature** in our project, as it ensures instant audible alerts inside the house, even if the user is not actively checking the mobile app or IoT dashboard.

## 2.12 SMPS (Switch Mode Power Supply)

Switched Mode Power Supplies (SMPS) are modern power converters that take AC mains voltage and convert it into regulated low-voltage DC. Unlike traditional linear power supplies, which rely on bulky transformers and waste energy as heat, SMPS units are **compact, lightweight, and highly efficient (85–95%)**. Because of these advantages, SMPS is now used everywhere—laptops, mobile chargers, set-top boxes, and also in smart home applications like our project.



*Figure 2. 24: SMPS (Switch Mode Power Supply)*

### 2.12.1 Circuit Diagram of smps

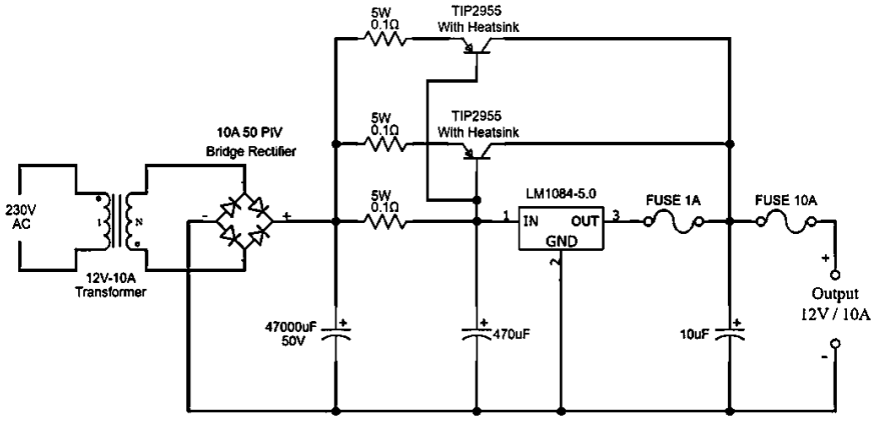



*Figure 2. 25: 12V 10A Circuit Diagram of SMPS*

### 2.12.2 Working Principle

The operation of SMPS involves several stages:

1. **Input Filtering**: The incoming AC passes through a filter that removes surges and suppresses electromagnetic interference (EMI).
2. **Rectification & Smoothing:** A bridge rectifier converts AC to pulsating DC, and capacitors smooth it.
3. **High-Frequency Switching**: MOSFETs or transistors switch the DC at frequencies between 25 kHz and 500 kHz**,** reducing transformer size.
4. **Transformer & Isolation**: The high-frequency signal passes through a transformer that steps the voltage up or down and provides electrical isolation.
5. **Output Rectification & Regulation**: The secondary voltage is rectified again, filtered by capacitors, and regulated using feedback circuits (often with an opto-isolator).

### 2.12.3 Application in Our Project

In our Project**,** the SMPS is used to convert 220V AC mains into stable 12V DC. This 12V supply powers the relay module and, through a buck converter, also provides 5V for the ESP32, sensors, and buzzer. The use of SMPS ensures that the system remains **compact, safe, and energy efficient**, while also reducing heat generation compared to a linear supply.

## 2.13 Power Module Multi Output Voltage Conversion

The DC-DC Multi Output Power Module is a compact and efficient voltage regulator board designed for projects requiring multiple supply levels. It can take an input voltage between **6V and 12V DC** and provide three stable outputs: **3.3V, 5V, and 12V**. The 12V output is directly derived from the input, while the 3.3V and 5V outputs are regulated with high precision. This makes the module highly suitable for powering microcontrollers, sensors, relays, and other electronic components in a single system. A red power LED indicator ensures easy monitoring of the module's operational status.

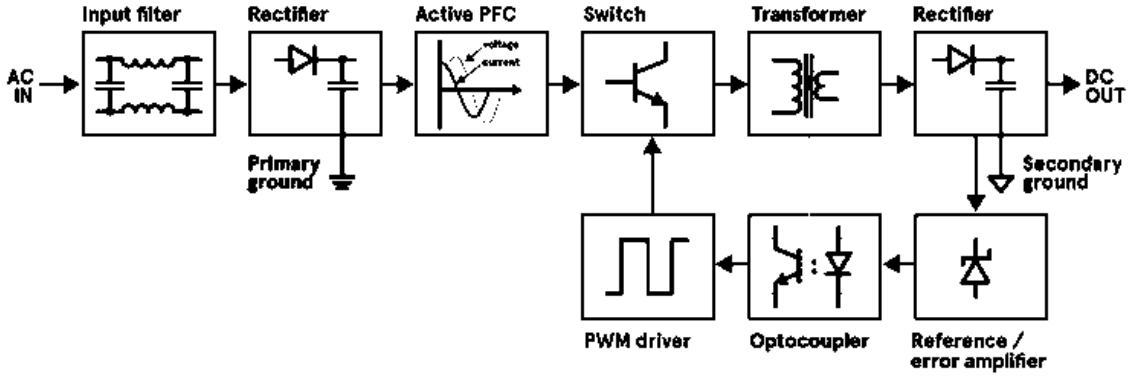



*Figure 2. 26: Power Module Multi Output Voltage Conversion*

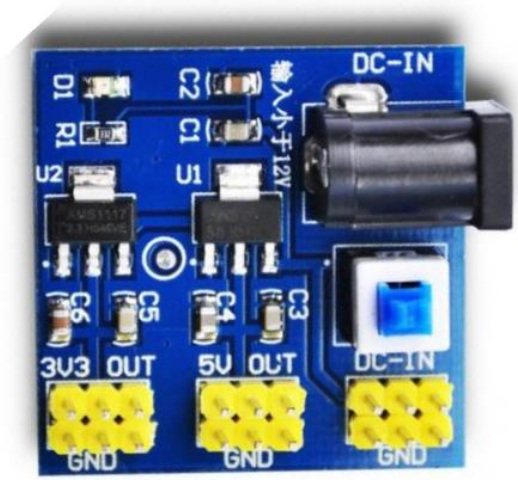



*Figure 2. 27: Multi Voltage Output Power Module*

### 2.13.1 Specifications of Power Module Multi Output Voltage

*Table 2. 18: Specifications of Multi Voltage Output Power Module*

| Parameter | Value / Description |
|---|---|
| Input Voltage | DC 6V – 12V (must be at least 1V higher than output) |
| Output Voltages | 3.3V (±0.05V)<br>5V (±0.05V)<br>12V (direct input pass-through) |
| Max Load Current (5V) | 800 mA (recommended not to exceed) |
| PCB Size | 4.5 cm × 4.5 cm |
| Indicator | Red LED (power status) |

### 2.13.2 Purpose in Our Project

In our **IoT Based Multicontrolled Smart Home and Protection System**, this power module plays a key role in distributing stable voltages to different components. The **12V output** is used for powering the relay module, the **5V output** is supplied to the ESP32 development board and MQ-2 gas sensor, and the **3.3V output** is available for modules that require lower voltage. By combining all three regulated outputs on a single board, the module simplifies the power management in our project, reduces wiring complexity, and ensures reliable operation of all devices.

## 2.14 USB Output (5V Supply for ESP32 & Sensors)

In my project ,the ESP32 requires a stable 5V power supply to operate reliably. For this purpose, I have used a USB Step Down (Buck) Converter Module, which efficiently converts a higher DC input voltage (6V–24V) into a constant 5V USB output.

This module is compact, lightweight, and provides high conversion efficiency (up to 97.5%), making it perfect for powering microcontrollers and IoT-based systems like the ESP32. It also includes built-in protection features such as overvoltage protection, short circuit protection, and fuse safety, ensuring both durability and reliability in continuous smart home operation.

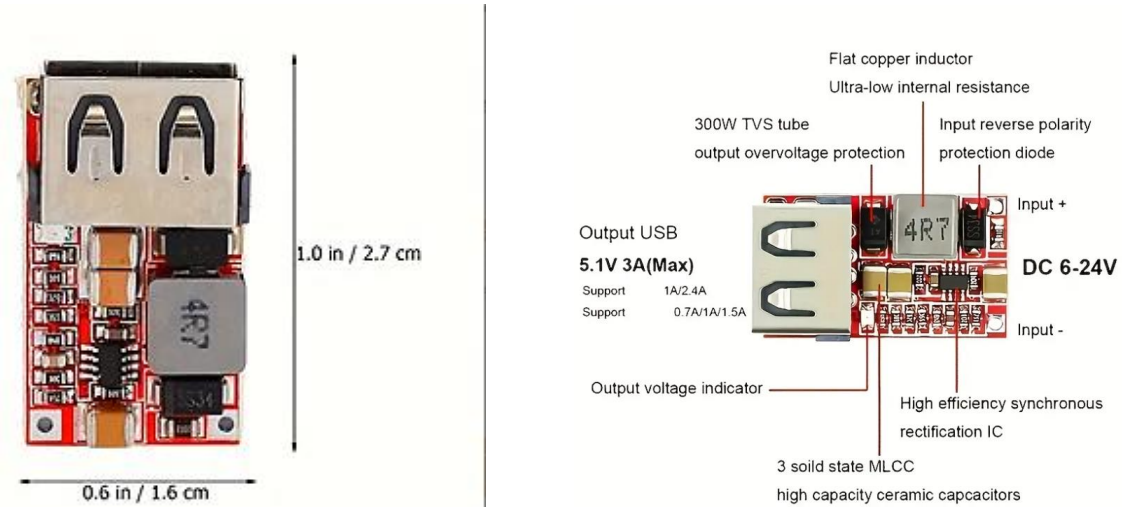



*Figure 2. 28: 5V USB Output*

### 2.14.1 Specifications of USB Output

*Table 2. 19: Specifications of USB Output*

| Parameter | Specification |
|---|---|
| **Module Type** | Non-isolated Buck Converter (with synchronous rectification) |
| **Input Voltage** | 6V – 24V DC |
| **Output Voltage** | 5.1V – 5.2V (USB Output) |
| **Max Output Current** | 3A |
| **Conversion Efficiency** | Up to 97.5% |
| **Switching Frequency** | 500 kHz |
| **Output Ripple** | ~10 mV (at 12V to 5V, 3A load) |
| **Load Regulation** | ±1% |
| **Voltage Regulation** | ±0.5% |
| **Dynamic Response** | 5% at 200 µs |
| **Protection** | Overvoltage, Short circuit, 1.5A input fuse, 300W TVS diode clamp |
| **Operating Temperature** | -40°C to +85°C (Industrial Grade) |

## 2.15 Circuit Breaker

A **circuit breaker** is an automatic electrical switch designed to protect an electrical circuit from damage caused by overload or short circuit. It works as a safety device that immediately interrupts the current flow when it detects abnormal conditions, such as excessive current or a sudden fault. Unlike a fuse, which burns out and needs replacement, a circuit breaker can be reset and used again, making it more reliable and cost-effective.

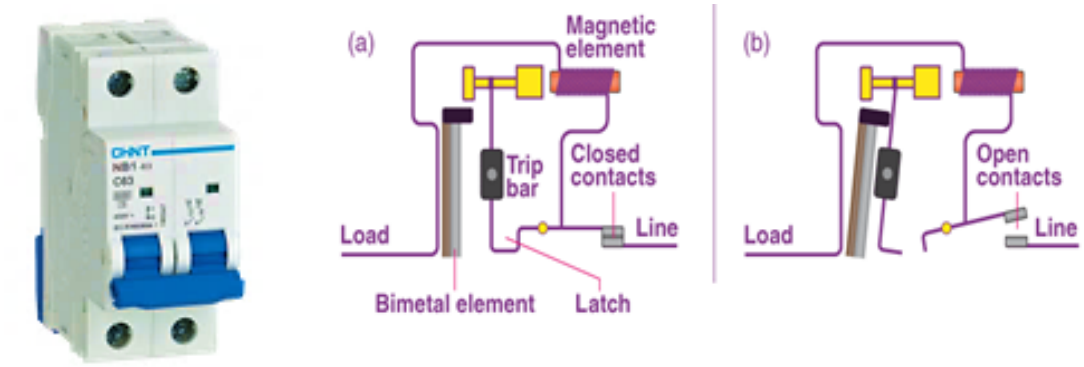



*Figure 2. 29: Miniature Circuit Breaker (MCB)*

A Miniature Circuit Breaker (MCB) works by automatically disconnecting the circuit during abnormal conditions. When an overload occurs, the bi-metallic strip heats up and bends, releasing the latch to trip the breaker. In case of a short circuit, the sudden surge of current energizes the solenoid, which pushes the plunger to trip the contacts instantly. Thus, the MCB ensures protection against both overload and short-circuit faults, and it can be reset manually after tripping.

In our project **IoT Based Multicontrolled Smart Home Automation and Protection System**, we are using a circuit breaker to provide an extra layer of protection. Since multiple loads like fans, lights, and appliances are being monitored and controlled, a sudden short circuit or overload could harm the system. The circuit breaker ensures that whenever such faults occur, the current is cut off instantly, protecting both the smart automation system and the connected devices.

## 2.16 Electrical Dual Row Terminal Block

The electrical dual row terminal block is a compact and reliable connector used to join multiple wires in an organized manner. It consists of two rows of screw or spring-clamp terminals where wires can be inserted and secured tightly, ensuring safe and durable electrical connections. Inside the block, the conductive metal strip allows proper flow of current between connected wires while the insulating body prevents short circuits. The dual row arrangement makes it easier to distribute both input and output lines, keeping the wiring neat and manageable.



*Figure 2. 30: Electrical Dual Row Terminal Block*

In our project **,** we used the dual row terminal block to distribute power supply and connect different modules such as the ESP32, relay module, sensors, and protection devices. This makes the wiring system organized, safe, and easier to maintain. It also provides flexibility during troubleshooting or replacement of components, since each connection can be accessed individually without disturbing the entire system.

## 2.17 Connecting Wire and Jumper

Wires are the basic medium for carrying current from one point to another in any electrical or electronic circuit. In our project we used insulated copper wires to connect the power supply, relays, and load appliances. These wires ensure safe current flow, provide mechanical strength, and make the whole system more reliable for long-term use compared to jumper wires that are only suitable for testing.



*Figure 2. 31: Connecting Wire*

Jumper wires are small pre-made wires that are mainly used for quick connections on breadboards and between modules. They come in three types: male-to-male, male-to-female, and female-to-female. In our project we used jumper wires to connect the ESP32 with sensors and modules during testing and development. They made the prototyping faster because no soldering was needed, and the circuit could be changed or modified anytime easily.



*Figure 2. 32: Jumper Wire*

## 2.18 Lugs (Wire Connectors/Crimping Lugs)

In electrical systems, lugs are used to terminate wires securely so they can be connected to other devices like relays, switches, or terminal blocks. In our project, we used insulated ferrule lugs (E1008 type) to make neat and reliable wire connections. These lugs are made of high-quality copper that ensures excellent conductivity, and they are insulated with PVC/Nylon which protects against high temperatures and increases durability. To improve lifespan, the surface is electro-plated with tin, which prevents corrosion. The red-colored

insulation makes them easy to identify during wiring and reduces the chance of wrong connections.

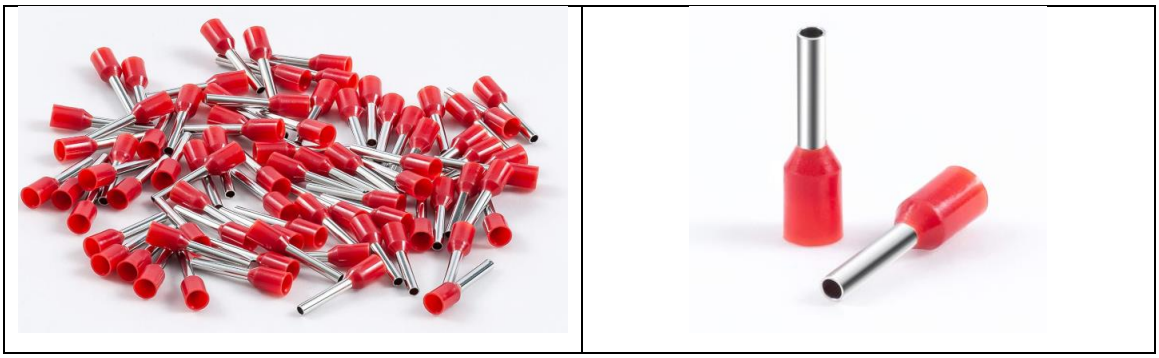


*Figure 2. 33: Lugs (Wire Connectors/Crimping Lugs)*

For our system, these ferrule lugs were mainly used for 18 AWG wires, and they operate safely even under high load conditions. By using lugs, we achieved secure, long-lasting connections that are both electrically efficient and mechanically strong, which is very important for the stability of our IoT Based Multicontrolled Smart Home and Protection System.

## 2.19 Summary

The components used are studied individually. Their purpose in the system is explained along with their ratings, pinouts and connections.

# Chapter 3

# HARDWARE DEVELOPMENT

## 3.1 Introduction

The hardware of our **AI Based Multicontrolled Smart Home and Protection System** is divided into three parts:

- **Power Section**
- **Controller Unit**
- **Relay Section**.

The power section provides stable DC supply for the ESP32, sensors, and relay module, while also helping to measure line voltage and current. The controller unit (ESP32) collects data from PT, CT, switches, and sensors, then processes it to take control decisions. Finally, the relay section switches the AC loads safely according to the controller's command. Together, these sections make the system reliable, safe, and easy to control both locally and through IoT.

## 3.2 Block Diagram

This project is divided into three block diagrams:

- **Power Section** – Supplies stable DC voltages (3.3V, 5V, 12V) for ESP32, sensors, and relays.
- **Controller Unit** – ESP32 processes sensor inputs and sends control signals.
- **Relay Module** – Operates AC loads safely based on controller commands.

### 3.2.1 Block Diagram of Power Section

In our project, the AC supply is first given to the **SMPS**, which converts 220V AC into 12V DC. This 12V output is then distributed into different parts of the system. One path goes to the **relay module**, while another path is connected to the **USB output module**, which provides a stable 5V supply for the ESP32 controller. The 12V line is also connected to a **voltage divider circuit**, which steps it down to 5V for powering the IR receiver, smoke

sensor, buzzer, and temperature sensor. In this way, the power section ensures stable voltage levels for all components used in the system.

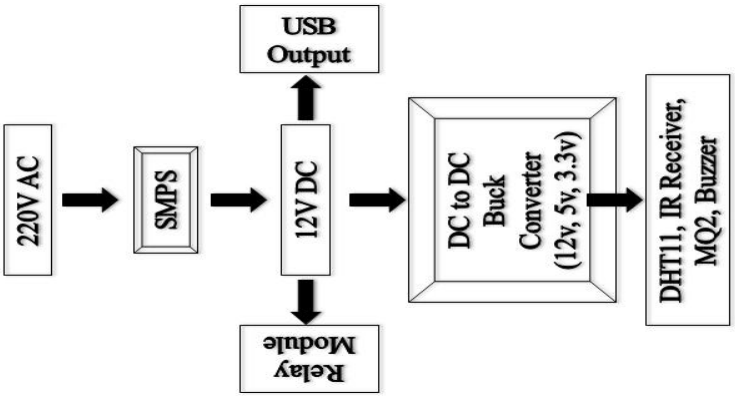



*Figure 3. 1: Block Diagram of Power Section*

### 3.2.2 Block Diagram of Controller Unit

In the control unit block diagram, all control signals pass through the ESP32 before reaching the relay module and the load. When a manual switch is pressed, the signal is sent directly to the ESP32, which then activates the relay module. Similarly, if an IR remote is used, the signal is first received by the IR receiver and then passed to the ESP32 for processing before controlling the relay. For IoT control, commands are sent from the mobile app or dashboard to the IoT Cloud, which communicates with the ESP32. The same process occurs with Google Home or AI assistants, which are connected to the IoT Cloud. In all cases, the ESP32 acts as the central controller, ensuring that the relay module responds correctly and the connected loads are managed efficiently.

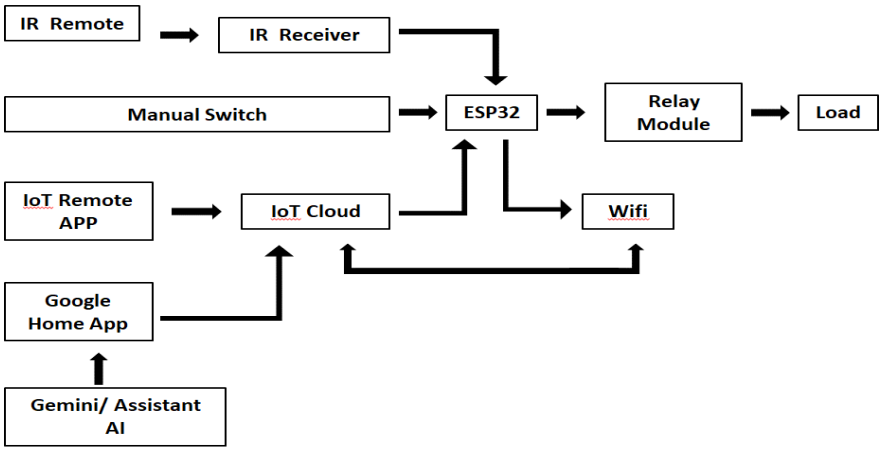



*Figure 3. 2: Block Diagram of Controller Unit*

## 3.3 Designing Software (Arduino IDE, Arduino IoT Cloud, Google Home Integration)

For designing and implementing the software part of our project, we used **Arduino IDE, Arduino IoT Cloud, and Google Home integration**.

The **Arduino IDE** is an open-source software used for writing, compiling, and uploading programs to microcontrollers such as the ESP32. It provides a simple coding environment and supports multiple programming languages like C and C++. The IDE also includes serial monitoring tools that help in debugging and testing the system.

The **Arduino IoT Cloud** is an online platform that allows real-time monitoring and remote controlling of devices. It enables data visualization, device-to-device communication, and cloud storage of sensor data. Through the IoT Cloud dashboard, users can access system data such as load status, temperature, and gas detection directly from their smartphone or computer.

For smart home integration, we used **Google Home**. It is a popular AI-powered platform that connects with IoT devices and allows users to control them through voice commands using Google Assistant. By linking the Arduino IoT Cloud with Google Home, we were able to add AI-based voice control features, making our system more interactive and user-friendly.

Thus, the combination of Arduino IDE, IoT Cloud, and Google Home created a complete environment for programming, monitoring, and intelligent control of our project.




*Figure 3. 3: Logo of Arduino IDE, Arduino IoT Cloud, Google Home*

## 3.4 Circuit Diagram

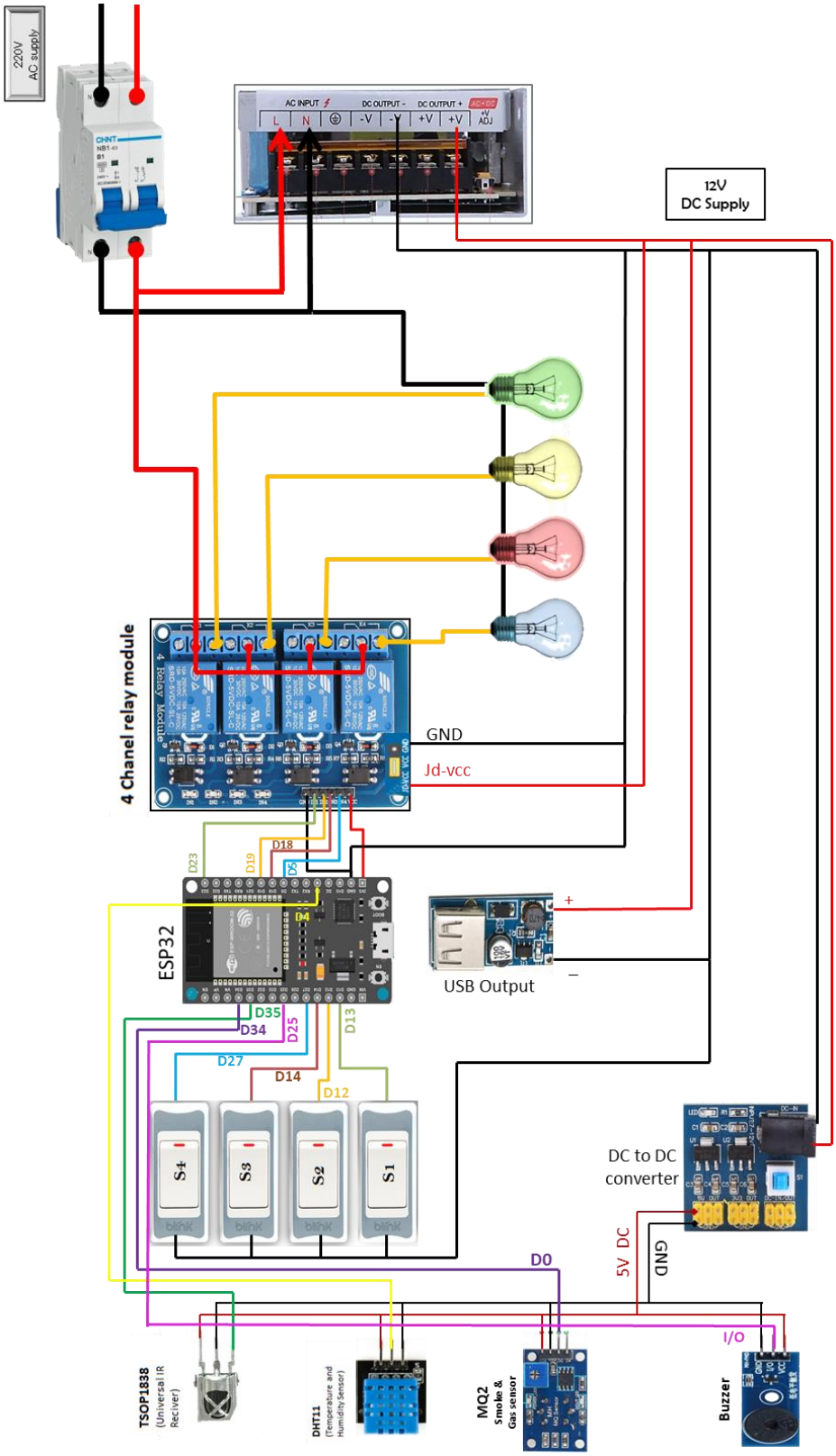



*Figure 3. 4: Circuit Diagram of the project*

## 3.5 Construction of the Main Circuit

For the construction of our project, we first collected all the required modules and components as listed in Chapter 2. The ESP32 microcontroller was selected as the main control unit because of its built-in Wi-Fi and IoT compatibility. It was mounted on a board using a female header rail for secure placement and easy handling.

The power supply section was built using a **12V SMPS**, which converts the AC mains supply into 12V DC. This 12V line was used in two ways: one part directly powered the **4-channel relay module**, and another part was connected to a **USB step-down converter**. The USB module provided a stable 5V supply to power the ESP32, ensuring smooth operation of the controller.

The **sensors and modules** were then connected. The **MQ2 gas and smoke sensor**, **IR receiver module (TSOP1838)**, **buzzer module**, and **temperature sensor** were all powered from the 5V output of the USB step-down converter and interfaced with the ESP32 through dedicated GPIO pins.

Next, the **relay module** was connected. On its input side, the module had pins for VCC, GND, and four control inputs. VCC was connected to the 12V output of the SMPS, GND was tied to the common ground, and the input pins were linked to the ESP32's GPIO pins. On the output side, the relay screw terminals were wired with the AC line and the household appliances, enabling safe switching and control.

In this way, all the components including the ESP32, relay module, IR receiver, smoke sensor, buzzer, and temperature sensor were assembled and interconnected on the board. This formed a complete hardware system where appliances could be controlled through manual relay switching, IR remote, and IoT connectivity, with protection features against gas and smoke hazards.

## 3.6 Working Principle

After assembling the system, the next step is to observe its operation and performance. The IoT Based Multicontrolled Smart Home Automation and Protection System is designed around the ESP32 microcontroller. When powered on, the ESP32 initializes and connects to the Wi-Fi network for IoT functionality. Once connected, it continuously listens for input signals from manual switches, the IR receiver, IoT Cloud, and AI voice assistants.

If a manual switch is pressed, the signal is read by the ESP32 and the corresponding relay is triggered to turn the appliance ON or OFF. Similarly, when the IR remote is used, the IR receiver decodes the infrared signal and sends it to the ESP32, which again controls the relay accordingly. In IoT mode, the ESP32 communicates with the Arduino IoT Cloud, receiving commands from the user's smartphone or computer to operate the appliances.

Additionally, the system integrates Google Assistant / Gemini AI for voice control. When the user gives a voice command such as "Turn on the light," the request is processed through Google Home or Gemini AI, transmitted to the IoT Cloud, and then forwarded to the ESP32. Based on this instruction, the ESP32 activates the appropriate relay and controls the connected device. This feature enables true hands-free and intelligent interaction with the system.

Apart from control, the system also ensures protection. The MQ-2 sensor detects smoke or gas leakage, and immediately the buzzer is activated for a local alert. Simultaneously, the ESP32 sends a warning notification through the IoT cloud to keep the user informed. By combining manual, IR, IoT, and AI voice control with safety features, the system delivers a reliable, flexible, and smart home automation solution**.**

## 3.7 Flow Chart

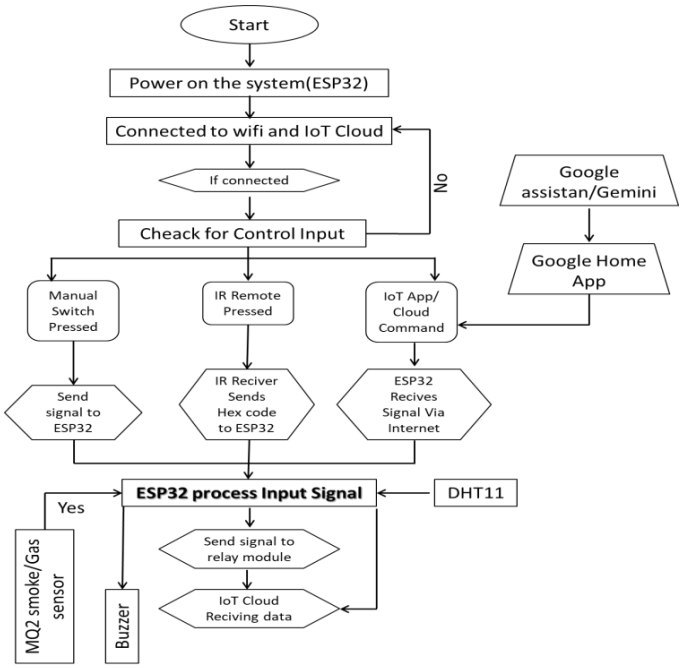



*Figure 3. 5: Flow Chart of the project*

### 3.7.1 Flowchart Step by Step Operation

1. Start → Power ON the System (ESP32 boots up)
2. Connect to Wi-Fi and IoT Cloud
    - *If connected → proceed*
    - *If not connected → keep retrying*
3. Check for Control Input
    - *Manual Switch pressed? → Send signal to ESP32*
    - *IR Remote pressed? → IR Receiver sends HEX code to ESP32*
    - *IoT App / Cloud command? → ESP32 receives signal via internet*
    - *Google Assistant / Gemini AI command? → IoT Cloud forwards to ESP32*
4. ESP32 Processes Input Command
5. Send Signal to Relay Module
    - *Relay ON → Load ON*
    - *Relay OFF → Load OFF*
6. Safety Monitoring (Parallel Process)
    - *MQ-2 Smoke/Gas Sensor detects danger?*
    - *If YES → Activate Buzzer*
7. Loop Back → Keep monitoring for next input or safety event

## 3.8 Summary

The circuit diagram of the IoT Based Multicontrol Smart Home System and Protection shows the interconnections of all components with the ESP32 microcontroller. It includes input devices such as manual switches, IR receiver, DHT11 sensor, and MQ2 smoke sensor, along with output devices like relays, buzzer, and WiFi status LED. The relays control the household loads, while the buzzer provides alerts during smoke detection. IoT Cloud and Google Home integration allow wireless and AI-based control of the system. The schematic diagram uses standard symbols to represent connections, but it does not reflect the actual physical arrangement of components. In the real setup, power supply units, circuit breaker, and terminal blocks ensure safety and stability of operation. Thus, the circuit diagram provides a clear functional view of the entire system.

# Chapter 4

# RESULT AND DISCUSSION

## 4.1 Introduction

Most home automation systems are designed only for basic control or monitoring, not for both together. Traditional systems can show simple data but cannot provide protection features or real-time smart responses. They are also limited in flexibility and usually fixed in one place, which makes them dependent on manual operation. To solve these issues, we developed an IoT Based Multicontrolled Smart Home Automation and Protection System**.** It integrates monitoring, control, and protection in one compact design. The system can be operated manually, through an IR remote, or online via IoT and Google Assistant, making it highly flexible. It also gives real-time digital data that can be accessed from a smartphone at any place, while ensuring safety through automated protection features.

## 4.2 Result

The experimental model was built according to the designed circuit diagram, and the system performed as expected. When the ESP32 is connected to the internet, the loads can be monitored and controlled directly from a smartphone through the IoT platform. The system also responds properly to manual switches and IR remote control, ensuring flexible operation. In addition, real-time monitoring of temperature, smoke, and gas levels was successfully achieved. Whenever abnormal conditions were detected, the buzzer alarm was triggered . This multicontrolled and protective approach not only provides user convenience but also enhances safety while saving electricity and time through smart load management.

## 4.3 Advantages of the Monitoring System

1. The system reduces unnecessary current waste.
2. The overall size of the system is small and compact.
3. Provides safety by monitoring the environment.
4. Wireless connectivity with the internet.

5. Ensures high wireless security.
6. The system can detect temperature, humidity, and smoke.
7. Provides quick alerts and timely warnings.
8. Very low maintenance cost.
9. All required components are easily available.
10. With AI (Google Assistant/Gemini), loads can be controlled by voice commands or scheduled to switch ON/OFF at any desired time.

## 4.4 Limitations of the System

1. The initial cost of implementation is comparatively high due to the requirement of multiple sensors, relays, and IoT modules.
2. The installation process is complex and requires proper wiring, configuration, and technical knowledge.
3. The system is fully dependent on internet connectivity; without internet, cloud and AI-based features will not work.
4. The range of IR remote control is limited and only works within line-of-sight.

## 4.5 Cost Estimation of the project

*Table 4. 1: Cost Estimation of the project*

| Sl. No | Component Name | Quantity | Unit Price (BDT) | Total (BDT) |
|---|---|---|---|---|
| 01 | ESP32 Microcontroller Board | 1 | 800 | 800 |
| 02 | ESP32 Terminal Adapter | 1 | 400 | 400 |
| 03 | DHT11 Temperature & Humidity Sensor | 1 | 150 | 150 |
| 04 | MQ-2 Smoke & Gas Sensor | 1 | 150 | 150 |
| 05 | IR Receiver (TSOP1838) | 1 | 200 | 200 |
| 06 | IR Remote Control | 1 | 120 | 120 |
| 07 | Manual Switches | 4 | 30 | 120 |
| 08 | Buzzer Module | 1 | 150 | 150 |
| 09 | Relay Module (4-Channel) | 1 | 350 | 350 |

| | | | | |
|---|---|---|---|---|
| 10 | SMPS (Switch Mode Power Supply) | 1 | 600 | 600 |
| 11 | DC to DC Buck Converter | 1 | 150 | 150 |
| 12 | USB Output Module (5V Supply) | 1 | 100 | 100 |
| 13 | Circuit Breaker | 1 | 400 | 400 |
| 14 | Electrical Dual Row Terminal Block | 2 | 80 | 160 |
| 15 | Connecting Wires (Male-Female & Normal) | 1 lot | 100 | 100 |
| 16 | Lugs (Wire Connectors) | 1 set | 70 | 70 |
| 17 | Acrylic Board (Base for Mounting) | 1 | 1100 | 1100 |
| 18 | Wire Management Channel | 1 | 100 | 100 |
| 19 | Bulb Holders, Bulbs | 4 pcs | 70 | 280 |
| | **Total Cost** | | | **5500** |

## 4.6 Applications of the System

1. Home Automation: Control lights, fans, AC, and other appliances smartly.
2. Energy Saving: Reduce electricity consumption by monitoring loads.
3. Safety: Smoke, gas detection, and circuit protection increase safety.
4. Education Sector: Manage fans, lights, and projectors in classrooms.
5. Industry & Factory: Operate motors, pumps, and machines efficiently.
6. Office Use: Control lights, ACs, and computers for better management.

## 4.7 Summary

The traditional home automation systems are costly and limited in control. They also lack advanced features like safety monitoring and intelligent decision-making. In this project, these issues are solved by developing an IoT Based Multicontrolled Smart Home and Protection System, which integrates IoT cloud, IR remote, and manual switches. The addition of **AI enables smart decision-making, predictive control, and enhanced safety**, making the system more efficient, reliable, and user-friendly.

# Chapter 5

# CONCLUSION

## 5.1 Conclusion

The project demonstrates an efficient way to integrate automation with safety. By combining IoT, IR, manual, and AI voice control, the system not only provides flexibility in appliance operation but also ensures protection through real-time smoke, gas, and temperature detection. Overall, it offers a practical and reliable approach to modern smart home management.

## 5.2 Future Scope of Modification

1. **Real-Time Smart Metering:** Monitor voltage, current, power, and calculate electricity bills instantly.
2. **Notification & Alerts:** Automatic SMS / App / Email alerts during overload, fire, or gas leakage.
3. **Heavy Load & 3-Phase Control:** Extend the system for controlling motors, pumps, and 3-phase industrial machines.
4. **Industrial Automation:** Integration with PLCs for controlling production lines, boilers, compressors, and large-scale equipment.
5. **Renewable Energy Monitoring:** Combine with solar or wind energy to track generation and usage.
6. **Smart Security System:** Add CCTV, PIR motion sensors, and door locks for safety.
7. **AI & Machine Learning:** Predict user behavior (like daily fan/light use) and automate intelligently.
8. **Data Analytics Dashboard:** Create a web/app-based dashboard for real-time performance graphs.

# REFERENCES


[1] Arduino Programming Notebook.

[2] Arduino for Beginners.

[3] ESP32 Technical Reference Manual, Espressif Systems.

[4] https://en.wikipedia.org/wiki/ESP32

[5] https://docs.espressif.com/projects/esp-idf/en/latest/esp32/

[6] https://components101.com/sensors/dht11-temperature-sensor

[7] https://components101.com/sensors/mq2-gas-sensor

[8] https://lastminuteengineers.com/ir-receiver-arduino-tutorial/

[9] https://www.vishay.com/docs/82459/tsop17xx.pdf

[10] https://en.wikipedia.org/wiki/Relay

[11] https://electronicsforu.com/resources/learn-electronics/7805-ic-voltage-regulator

[12] https://talema.com/intro-switched-mode-power-supplies-smps/

[13] https://en.wikipedia.org/wiki/Circuit_breaker

[14] https://learn.sparkfun.com/tutorials/connector-basics/power-connectors

[15] https://www.resistorguide.com/what-is-a-resistor/

[16] https://en.wikipedia.org/wiki/Capacitor

[17] https://developers.google.com/assistant

[18] https://ai.google.dev/gemini-api

[19] https://docs.arduino.cc/cloud/iot-cloud

[20] https://randomnerdtutorials.com/projects-esp32/

[21] https://en.wikipedia.org/wiki/Buzzer

[22] https://www.allaboutcircuits.com/projects/using-relays-to-control-high-voltage-devices-with-an-arduino/

[23] https://www.arduino.cc/en/software

# APPENDIX

```
#include <ArduinoIoTCloud.h>
#include <Arduino_ConnectionHandler.h>
#include <DHT.h>
#include <IRremote.h>

const char DEVICE_LOGIN_NAME[]  = "07fa21ef-3a1f-4664-a4ac-425157039316"; //Enter DEVICE
ID

const char SSID[]               = "WifiName";    //Enter WiFi SSID (name)
const char PASS[]               = "11223344";    //Enter WiFi password
const char DEVICE_KEY[]         = "5mOzcH7yVNj8gSssmycOggtgx";    //Enter Secret device
password (Secret Key)

#define DHTPIN              4 //D4  pin connected with DHT
#define IR_RECV_PIN         35 //D35 pin connected with IR Receiver IC

// define the GPIO connected with Relays and switches
#define RelayPin1 23  //D23
#define RelayPin2 19  //D19
#define RelayPin3 18  //D18
#define RelayPin4 5  //D5

#define SwitchPin1 13  //D13
#define SwitchPin2 12  //D12
#define SwitchPin3 14  //D14
#define SwitchPin4 27  //D27

#define wifiLed    2   //D2

// Uncomment whatever type you're using!
#define DHTTYPE DHT11     // DHT 11
//#define DHTTYPE DHT22   // DHT 22, AM2302, AM2321
//#define DHTTYPE DHT21   // DHT 21, AM2301

DHT dht(DHTPIN, DHTTYPE);
IRrecv irrecv(IR_RECV_PIN);
decode_results results;

int toggleState_1 = 0; //Define integer to remember the toggle state for relay 1
int toggleState_2 = 0; //Define integer to remember the toggle state for relay 2
int toggleState_3 = 0; //Define integer to remember the toggle state for relay 3
int toggleState_4 = 0; //Define integer to remember the toggle state for relay 4
```

```
// Switch State
bool SwitchState_1 = LOW;
bool SwitchState_2 = LOW;
bool SwitchState_3 = LOW;
bool SwitchState_4 = LOW;

float temperature1 = 0;
float humidity1   = 0;
int   reconnectFlag = 0;

bool allLightsOn = false;  // Toggle state for all lights on/off via 0 button

void onLight1Change();
void onLight2Change();
void onLight3Change();
void onLight4Change();

CloudSwitch light1;
CloudSwitch light2;
CloudSwitch light3;
CloudSwitch light4;
CloudTemperatureSensor temperature;

void initProperties(){

  ArduinoCloud.setBoardId(DEVICE_LOGIN_NAME);
  ArduinoCloud.setSecretDeviceKey(DEVICE_KEY);
  ArduinoCloud.addProperty(light1, READWRITE, ON_CHANGE, onLight1Change);
  ArduinoCloud.addProperty(light2, READWRITE, ON_CHANGE, onLight2Change);
  ArduinoCloud.addProperty(light3, READWRITE, ON_CHANGE, onLight3Change);
  ArduinoCloud.addProperty(light4, READWRITE, ON_CHANGE, onLight4Change);
  ArduinoCloud.addProperty(temperature, READ, 8 * SECONDS, NULL); //Update temperature
value after every 8 seconds
}

WiFiConnectionHandler ArduinoIoTPreferredConnection(SSID, PASS);


void readSensor(){

  float h = dht.readHumidity();
  float t = dht.readTemperature(); // or dht.readTemperature(true) for Fahrenheit

  if (isnan(h) || isnan(t)) {
```

```
    Serial.println("Failed to read from DHT sensor!");
    return;
  }
  else {
    humidity1 = h;
    temperature = t;
  }
}

void sendSensor()
{
  readSensor();
}

void ir_remote_control() {
  if (irrecv.decode(&results)) {

    switch(results.value){
      case 0xFF30CF:  relayOnOff(1); light1 = toggleState_1; break;
      case 0xFF18E7:  relayOnOff(2); light2 = toggleState_2; break;
      case 0xFF7A85:  relayOnOff(3); light3 = toggleState_3; break;
      case 0xFF10EF:  relayOnOff(4); light4 = toggleState_4; break;

      case 0xFF6897:  // 0 button code, toggle all lights
        if (!allLightsOn) {
          // Turn all lights ON
          digitalWrite(RelayPin1, LOW);
          digitalWrite(RelayPin2, LOW);
          digitalWrite(RelayPin3, LOW);
          digitalWrite(RelayPin4, LOW);

          toggleState_1 = 1;
          toggleState_2 = 1;
          toggleState_3 = 1;
          toggleState_4 = 1;

          light1 = 1;
          light2 = 1;
          light3 = 1;
          light4 = 1;

          allLightsOn = true;
          Serial.println("All Lights ON");
        } else {
          // Turn all lights OFF
```

```
          digitalWrite(RelayPin1, HIGH);
          digitalWrite(RelayPin2, HIGH);
          digitalWrite(RelayPin3, HIGH);
          digitalWrite(RelayPin4, HIGH);

          toggleState_1 = 0;
          toggleState_2 = 0;
          toggleState_3 = 0;
          toggleState_4 = 0;

          light1 = 0;
          light2 = 0;
          light3 = 0;
          light4 = 0;

          allLightsOn = false;
          Serial.println("All Lights OFF");
        }
        break;

      default:
        break;
    }

    irrecv.resume();
  }
}

void relayOnOff(int relay) {

  switch (relay) {
    case 1:
      if (toggleState_1 == 0) {
        digitalWrite(RelayPin1, LOW); // turn on relay 1
        toggleState_1 = 1;
        Serial.println("Device1 ON");
      }
      else {
        digitalWrite(RelayPin1, HIGH); // turn off relay 1
        toggleState_1 = 0;
        Serial.println("Device1 OFF");
      }
      delay(100);
      break;
    case 2:
```

```
      if (toggleState_2 == 0) {
        digitalWrite(RelayPin2, LOW); // turn on relay 2
        toggleState_2 = 1;
        Serial.println("Device2 ON");
      }
      else {
        digitalWrite(RelayPin2, HIGH); // turn off relay 2
        toggleState_2 = 0;
        Serial.println("Device2 OFF");
      }
      delay(100);
      break;
    case 3:
      if (toggleState_3 == 0) {
        digitalWrite(RelayPin3, LOW); // turn on relay 3
        toggleState_3 = 1;
        Serial.println("Device3 ON");
      } else {
        digitalWrite(RelayPin3, HIGH); // turn off relay 3
        toggleState_3 = 0;
        Serial.println("Device3 OFF");
      }
      delay(100);
      break;
    case 4:
      if (toggleState_4 == 0) {
        digitalWrite(RelayPin4, LOW); // turn on relay 4
        toggleState_4 = 1;
        Serial.println("Device4 ON");
      }
      else {
        digitalWrite(RelayPin4, HIGH); // turn off relay 4
        toggleState_4 = 0;
        Serial.println("Device4 OFF");
      }
      delay(100);
      break;
    default : break;
  }
}

void manual_control()
{
  if (digitalRead(SwitchPin1) == LOW && SwitchState_1 == LOW) {
    digitalWrite(RelayPin1, LOW);
```

```
  toggleState_1 = 1;
  SwitchState_1 = HIGH;
  light1 = toggleState_1;
  Serial.println("Switch-1 on");
}
if (digitalRead(SwitchPin1) == HIGH && SwitchState_1 == HIGH) {
  digitalWrite(RelayPin1, HIGH);
  toggleState_1 = 0;
  SwitchState_1 = LOW;
  light1 = toggleState_1;
  Serial.println("Switch-1 off");
}
if (digitalRead(SwitchPin2) == LOW && SwitchState_2 == LOW) {
  digitalWrite(RelayPin2, LOW);
  toggleState_2 = 1;
  SwitchState_2 = HIGH;
  light2 = toggleState_2;
  Serial.println("Switch-2 on");
}
if (digitalRead(SwitchPin2) == HIGH && SwitchState_2 == HIGH) {
  digitalWrite(RelayPin2, HIGH);
  toggleState_2 = 0;
  SwitchState_2 = LOW;
  light2 = toggleState_2;
  Serial.println("Switch-2 off");
}
if (digitalRead(SwitchPin3) == LOW && SwitchState_3 == LOW) {
  digitalWrite(RelayPin3, LOW);
  toggleState_3 = 1;
  SwitchState_3 = HIGH;
  light3 = toggleState_3;
  Serial.println("Switch-3 on");
}
if (digitalRead(SwitchPin3) == HIGH && SwitchState_3 == HIGH) {
  digitalWrite(RelayPin3, HIGH);
  toggleState_3 = 0;
  SwitchState_3 = LOW;
  light3 = toggleState_3;
  Serial.println("Switch-3 off");
}
if (digitalRead(SwitchPin4) == LOW && SwitchState_4 == LOW) {
  digitalWrite(RelayPin4, LOW);
  toggleState_4 = 1;
  SwitchState_4 = HIGH;
  light4 = toggleState_4;
```

```
    Serial.println("Switch-4 on");
  }
  if (digitalRead(SwitchPin4) == HIGH && SwitchState_4 == HIGH) {
    digitalWrite(RelayPin4, HIGH);
    toggleState_4 = 0;
    SwitchState_4 = LOW;
    light4 = toggleState_4;
    Serial.println("Switch-4 off");
  }
}

void doThisOnConnect(){
  Serial.println("Board successfully connected to Arduino IoT Cloud");
  digitalWrite(wifiLed, HIGH); //Turn off WiFi LED
}

void doThisOnSync(){
  Serial.println("Thing Properties synchronised");
}

void doThisOnDisconnect(){
  Serial.println("Board disconnected from Arduino IoT Cloud");
  digitalWrite(wifiLed, LOW); //Turn on WiFi LED
}

void setup() {
  Serial.begin(9600);
  delay(1500);

  initProperties();
  dht.begin();
  irrecv.enableIRIn();

  ArduinoCloud.begin(ArduinoIoTPreferredConnection);

  ArduinoCloud.addCallback(ArduinoIoTCloudEvent::CONNECT, doThisOnConnect);
  ArduinoCloud.addCallback(ArduinoIoTCloudEvent::SYNC, doThisOnSync);
  ArduinoCloud.addCallback(ArduinoIoTCloudEvent::DISCONNECT, doThisOnDisconnect);

  setDebugMessageLevel(2);
  ArduinoCloud.printDebugInfo();

  pinMode(RelayPin1, OUTPUT);
  pinMode(RelayPin2, OUTPUT);
  pinMode(RelayPin3, OUTPUT);
```

```
  pinMode(RelayPin4, OUTPUT);

  pinMode(wifiLed, OUTPUT);

  pinMode(SwitchPin1, INPUT_PULLUP);
  pinMode(SwitchPin2, INPUT_PULLUP);
  pinMode(SwitchPin3, INPUT_PULLUP);
  pinMode(SwitchPin4, INPUT_PULLUP);

  // All Relays OFF initially
  digitalWrite(RelayPin1, HIGH);
  digitalWrite(RelayPin2, HIGH);
  digitalWrite(RelayPin3, HIGH);
  digitalWrite(RelayPin4, HIGH);
}

void loop() {
  ArduinoCloud.update();

  manual_control();
  ir_remote_control();
  sendSensor();
}

void onLight1Change() {
  if (light1 == 1)
  {
    digitalWrite(RelayPin1, LOW);
    Serial.println("Device1 ON");
    toggleState_1 = 1;
  }
  else
  {
    digitalWrite(RelayPin1, HIGH);
    Serial.println("Device1 OFF");
    toggleState_1 = 0;
  }
}

void onLight2Change() {
  if (light2 == 1)
  {
    digitalWrite(RelayPin2, LOW);
    Serial.println("Device2 ON");
    toggleState_2 = 1;
```

```
  }
  else
  {
    digitalWrite(RelayPin2, HIGH);
    Serial.println("Device2 OFF");
    toggleState_2 = 0;
  }
}

void onLight3Change() {
  if (light3 == 1)
  {
    digitalWrite(RelayPin3, LOW);
    Serial.println("Device3 ON");
    toggleState_3 = 1;
  }
  else
  {
    digitalWrite(RelayPin3, HIGH);
    Serial.println("Device3 OFF");
    toggleState_3 = 0;
  }
}

void onLight4Change() {
  if (light4 == 1)
  {
    digitalWrite(RelayPin4, LOW);
    Serial.println("Device4 ON");
    toggleState_4 = 1;
  }
  else
  {
    digitalWrite(RelayPin4, HIGH);
    Serial.println("Device4 OFF");
    toggleState_4 = 0;
  }
}
```