MANIPAL UNIVERSITY
JAIPUR

Minor Project

IoT-Based Smoke Detection System for Fire Safety

MCA - III Sem

Subject Code: CA7131

Submitted By

Harsh Sharma

221020004

&

Ayush Vijay

221020032

Faculty Coordinator

Dr. Linesh Raja, Associate Professor

CERTIFICATE

Date-15/12/2023

This is to certify that the project titled **IoT-Based Smoke Detection System for Fire Safety** is a record of the bonafide work done by **Ayush Vijay and Harsh Sharma** (**221020032, 221020004**) submitted in partial fulfilment of the requirements for the award of the Degree of **Master of Computer Applications (M.C.A)** of Manipal University Jaipur, during the academic year 2023-24.

Dr. Linesh Raja
Associate Professor, Dept of Computer Application
Manipal University Jaipur

Dr. Devershi Pallavi Bhatt
HOD, Dept of Computer Application
Manipal University Jaipur

ACKNOWLEDGEMENT

I would like to express my heartfelt gratitude to my college faculty and project guide, Dr.Linesh Raja, for their unwavering support and guidance throughout the duration of this project.

Their expertise, encouragement, and mentorship have played a crucial role in shaping the project's direction and ensuring its successful completion. I am also thankful to my fellow classmates who have actively contributed to the project with their valuable insights and collaborative effort.

I would like to acknowledge the college administration for providing us with the necessary resources and facilities to carry out this project. Their constant support and belief in our abilities have been instrumental in our project's accomplishment. Additionally, I extend my appreciation to my family and friends for their continuous encouragement and understanding during this challenging yet rewarding journey.

ABSTRACT

In contemporary society, frequent occurrences of fire incidents pose a significant threat to communities, particularly in regions such as India, where approximately 80% of households lack adequate fire safety systems. These mishaps result in tragic loss of life and valuable resources. In response to this urgent need, this project proposes an IoT-based smoke detection system designed to mitigate the impact of fire-related emergencies. The system aims to address the prevalent lack of fire safety infrastructure in a substantial portion of households by leveraging the capabilities of Internet of Things (IoT) technology. By implementing a smart smoke detection system, we anticipate a reduction in the severity of fire incidents, leading to the preservation of both resources and human lives. This project seeks to make a valuable contribution towards enhancing fire safety measures in homes, thereby fostering a safer and more secure living environment for individuals and communities.

TABLE OF CONTENT

CHAPTER 1

INTRODUCTION

The term IoT(Internet of Things) refers to the connectivity of physical objects or things to the digital world by integrating various sensors and software into a central hub. This hub measures real-world criteria and boundaries as they continuously fluctuate, creating a database of gathered readings or values. Once the data reaches the cloud, specialized software processes it and autonomously decides on actions such as sending alerts or transferring data without user intervention. The Internet of Things involves the interconnection and interaction of objects, devices, and people through wireless networks, emerging as a novel business strategy in various sectors.

IoT devices have become integral in daily life, monitoring and controlling mechanical, electrical, and electronic systems in buildings and homes. This project explores a rapid response solution for fire incidents using an IoT-based model. Fire outbreaks pose a significant challenge to security, occurring in diverse locations such as houses, schools, factories, and offices, resulting in fortuitous deaths and property loss. Fire detector systems play a crucial role in alerting people before fire engulfs their surroundings. However, traditional fire detection systems often require extensive wiring and labour for installation, discouraging users from adopting them.

To address these challenges, the project introduces an IoT-based wireless fire detector system, leveraging the popularity of IoT systems and components, including wireless sensor networks,

for security purposes. The system incorporates NodeMCU, equipped with built-in Wi-Fi, and employs Arduino for simulation purposes. The wireless sensor network consists of

appropriately spaced nodes with microcontrollers (NodeMCU) connected to smoke detector sensors. These nodes continuously monitor the surrounding environment, detecting the presence of fire. When a fire is detected, a signal is sent to a central node, triggering a local buzzer alert. The buzzer continues until the temperature decreases to a normal value and smoke levels diminish.

This model and buzzer act as a pre-emptive measure, alerting authorities and individuals promptly to avoid potential consequences and damages. The proposed system features a simple and cost-effective architecture, with inbuilt Wi-Fi in NodeMCU facilitating data transfer to the user's mobile phone.

CHAPTER 2

Survey of Technology of various components used in the Project.

The components which we have used to design the project involve various technologies. Let us discuss these technologies one by one-

2.1 First component is NodeMCU (ESP8266)

Full form: Node Micro Controller Unit

It includes the main components of the computer: CPU, RAM, network (Wi-Fi) and even routines and SDKs. This makes it the best choice for many Internet of Things (IoT) projects.

Specifications-

The NodeMCU measures 49mm x 26mm with 0.1-inch pin pitch and 0.9-inch thread. NodeMCU is approximately 25% smaller which provides an advantage to the user in an IoT environment.

2.1.1 NodeMCU Pinout and Functions Explained

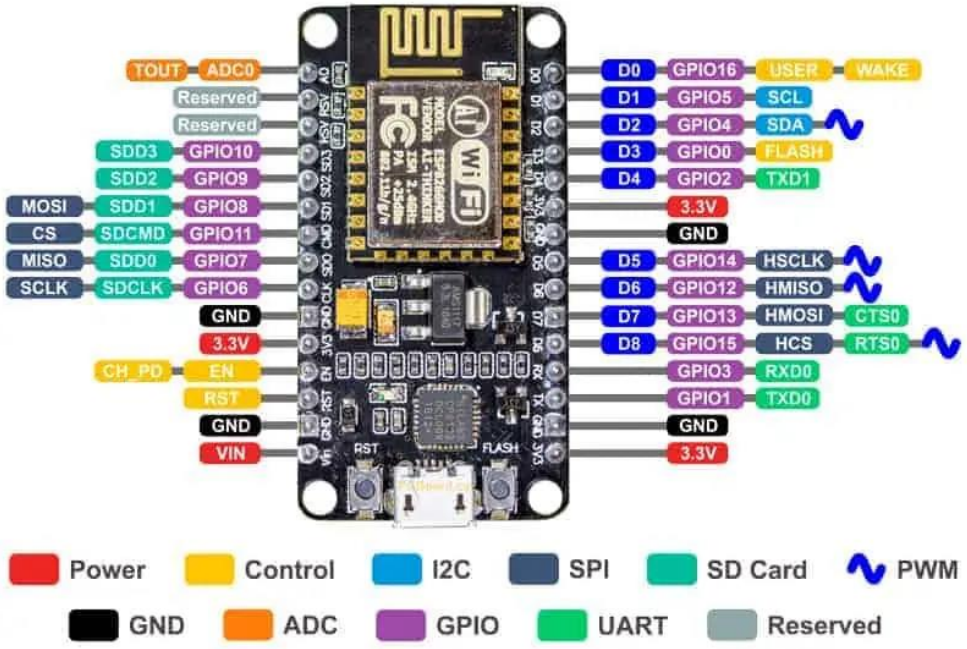

Fig 2.1: NodeMCU Pins

Power Pins There are four power pins. VIN pin and three 3.3V pins.

VIN can be used to directly power the NodeMCU/ESP8266 and its peripherals. The power supplied to VIN is regulated by the integrated regulator on the NodeMCU.

The 3.3V pins are the output of the integrated voltage regulator and can be used to power external components.

GND is the NodeMCU/ESP8266 ground pins.

I2C pins connect I2C sensors and peripherals.I2C Master and I2C Slave are supported. The functionality of the I2C interface can be implemented programmatically and the clock frequency is a maximum of 100 kHz.

GPIO Pins NodeMCU/ESP8266 has 17 This assign functions such as I2C, I2S, UART, PWM, IR remote control, LED lights and buttons via programs.

The ADC channel of the NodeMCU includes a 10-bit precision SAR ADC. Both functions can be implemented using ADCs. Test the supply voltage at the VDD3P3 pin and measure the input voltage at the TOUT pin. However, they cannot be used simultaneously.

UART Pins the NodeMCU/ESP8266 has 2 UART interfaces (UART0 and UART1) that provide asynchronous communication (RS232 and RS485) with communication speeds up to 4.5 Mbps. UART0 (TXD0, RXD0, RST0 and CTS0 pins) can be used for communication. But UART1 (TXD1 pin) only has data transfer, so it is mainly used for printing.

SPI pins NodeMCU/ESP8266 has 2 UART interfaces (UART0 and UART1) that provide asynchronous communication (RS232 and RS485) with communication speed up to 4.5 Mbps. UART0 (TXD0, RXD0, RST0 and CTS0 pins) can be used for communication. However, UART1 (TXD1 pin) only has data output so it is used for printing.

SDIO Pins the NodeMCU/ESP8266 includes an SDIO (Secure Digital Input/Output Interface) interface for direct connection to an SD card. Supports 4-bit 25 MHz SDIO v1.1 and 4-bit 50 MHz SDIO v2.0.

PWM pins This board has 4 pulse width modulation (PWM) channels. The PWM output can be used as a function and used to control motors and LEDs. The PWM frequency range is adjustable from 1000 μs to 10000 μs (100 Hz and 1 kHz).

The control pins Used to control NodeMCU/ESP8266. These pins include the chip enable pin (EN), reset pin (RST), and WAKE pin.

2.2 Second Major Component is MQ2 Gas Sensor

It is an MQ series sensor which is most widely used. It is a Metal Oxide Semiconductor sensor also known as Chemiresistors. Sensing is done based on the change in the sensing material when exposed to gases. It operates on 5V DC and consumes 800mW.

It can detect-

1. LPG
2. Smoke
3. Alcohol
4. Propane
5. Hydrogen
6. Methane
7. Carbon Monoxide

It can detect gases but cannot identify them.

2.2.1 Structure of MQ2 Gas Sensor

As we know MQ2 is a Heat sensing sensor. To protect it from Heat it has 2 stainless steel mesh known as **Anti-Explosion network**. It prevents Explosion inside the sensor. It also contains a **clamping ring** that secures the mesh to the body of the sensor.

These 2 components are marked in the image below-

Fig 2.1.1: Structure of MQ2 Sensor

2.2.2 Working of MQ2 Sensor

When the SnO2 semiconductor layer is heated to a high temperature, oxygen is adsorbed on the surface. When the air is clean, electrons in the conduction band of tin dioxide are attracted to oxygen molecules. This creates a barrier beneath the surface of the SnO2 particle, creating a process of electron depletion. As a result, the SnO2 film becomes very strong and blocks the flow of electric current.

However, in the presence of reducing gas, the density of adsorbed oxygen decreases when it reacts with the reducing gas, thus minimizing the effect. Therefore, electrons are released into the tin dioxide, allowing current to flow through the sensor.

Fig 2.2.2: MQ2 Sensor

2.3 Third major component for the project is Jumper Wires

Jumpers are wires with pins at both ends, allowing them to be used to connect two points together without needing to use a machine. Jumpers are often used with breadboards and other prototyping tools to make it easier to modify the circuit as needed. Very easy. It’s not as simple as a jumper.

Jumpers generally come in three versions: male to male, male to female, and female to female. The only difference is where the wire ends. The male end has a protruding pin to plug things into, while the female end doesn't have a pin to plug things into. Male-to-male connection

cables are the most common and the cables you will use the most. You will need a male-to-male cable when connecting two ports on the breadboard.

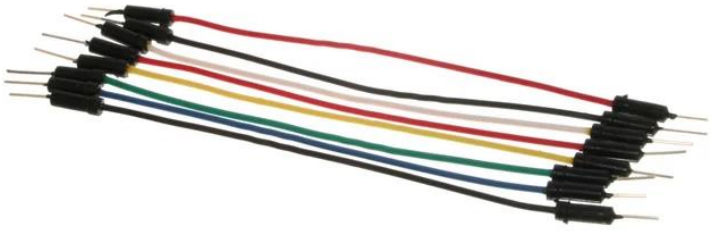

Fig 2.3: Jumper Wires

2.4 The fourth major component for the project is Buzzer.

Sound signaling devices such as bells or buzzers can be of electromechanical or piezoelectric or mechanical type. Its main function is to convert the signal from sound to sound. Generally, DC voltage is used as power supply and is used in timers, alarms, printers, alarms, computers, etc. using for. Depending on its design, the bell can produce different sounds such as music, chime, and siren.

The pin configuration of the ringer is shown below. It has two pins, positive and negative. The positive terminal is represented by a "+" symbol or a longer terminal. The positive terminal is supplied with 6V, while the negative terminal is represented by the "-" symbol or short-circuit terminal and is connected to the GND terminal.

Fig 2.4: Buzzer

2.4.1 Working Principle of Buzzer

The working principle of the buzzer is based on the theory that when electric current is applied to the piezoelectric material, a difference in pressure is created. In the piezoelectric type, a piezoelectric crystal is located between two conductors.

When an electric potential difference occurs between crystals, they repel one of the conductors and pull the other one through them. So, this constant movement will create a loud sound.

2.5 At last, we have Arduino IDE

The official software launched by Arduino.cc is used only to write, compile and load program codes for almost all Arduino modules/boards. Arduino IDE is an open-source software that can be easily downloaded and installed from the official Arduino website.

Arduino IDE is an open-source software developed by Arduino.cc and is mainly used to write, write and upload program code for almost all Arduino modules.

It is Arduino official software that makes compiling the program code very easy even for ordinary people who have no knowledge to start the learning process.

It works on all operating systems such as MAC, Windows, Linux and runs on the Java platform, which comes with functions and commands that play an important role in learning. Debugging, editing, and compiling the code.

There are many types of Arduino modules available, including Arduino Uno, Arduino Mega, Arduino Leonardo, Arduino Micro and more.

Each module has a microcontroller that is actually programmed on the board and accepts information in code form.

The main code created on the IDE platform, also known as a sketch, eventually creates a Hex file, which is then transferred and sent to the controller on the board.

The IDE environment usually consists of two main parts: editor and writer; The first one is used to write the program code, the second one is used to write the program code and send it to the Arduino module.

The environment supports C and C++ languages.

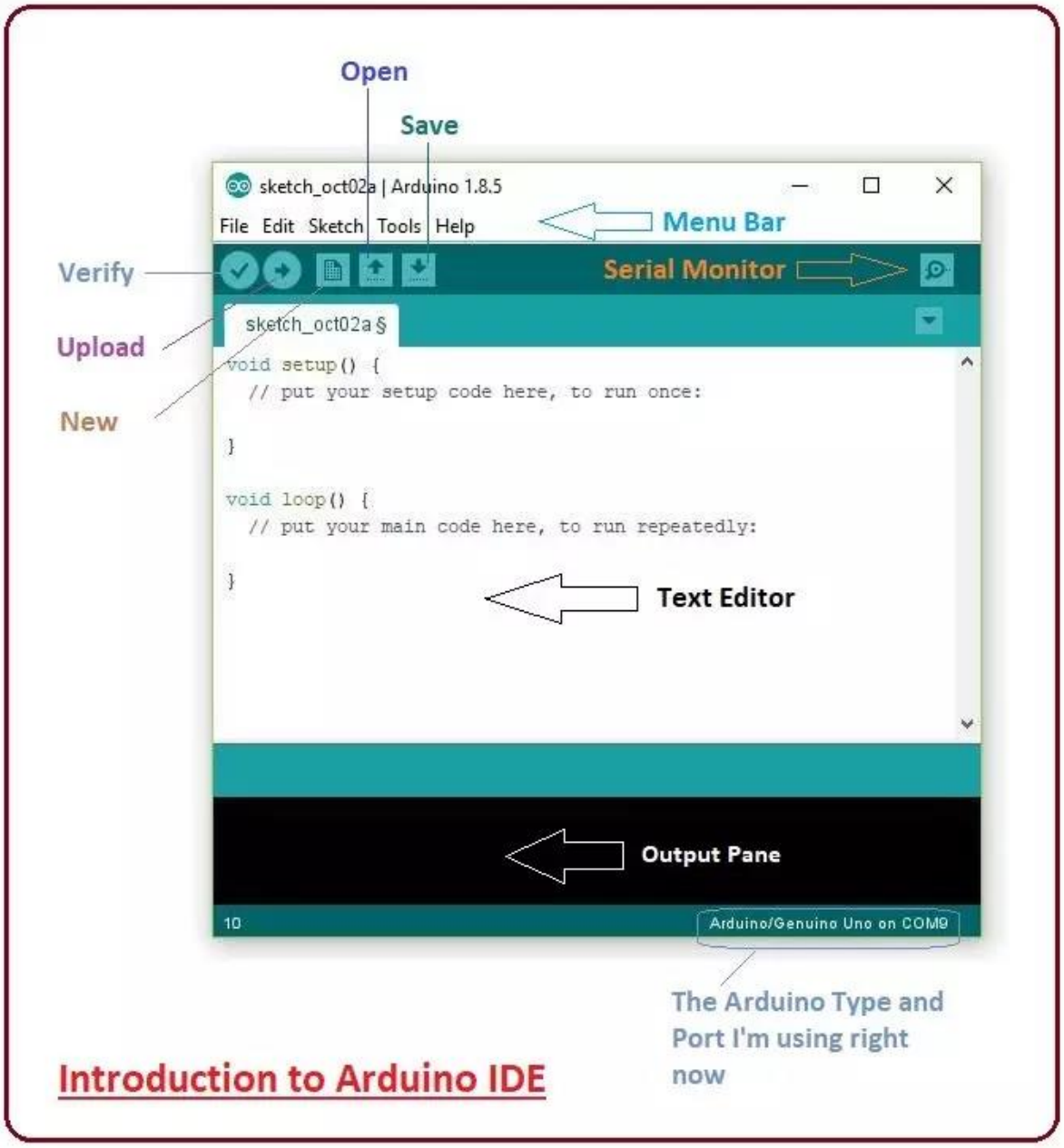

Fig 2.5: Arduino IDE

CHAPTER 3

Requirement Analysis for the Project

3.1 Problem definition:

3.1.1 Background:

In recent years, there has been a growing demand for advanced electronic devices using IoT technology. Traditional smoke detectors have limited capabilities for real-time monitoring, remote alarming, and integration with smart home devices.

3.1.2 Problem Description:

Develop an IoT-based smoke detector to improve fire safety by providing rapid detection, instant monitoring, and alarming while addressing the shortcomings of the current system.

3.2 Disadvantages of Main System:

3.2.1 Connection Limitations:

Traditional smoke detectors cannot connect to the internet or communicate with other devices, thus limiting their usefulness for remote monitoring.

3.2.2 Lack of emergency alarm:

Most existing systems rely on local alarms and cannot send instant notifications to users or the police in case of emergency.

3.2.3 Centralized Monitoring Challenges:

In large buildings or remote locations, legacy systems can be difficult to monitor centrally, resulting in slow responses.

3.2.4 Integration Complexity:

Integrating legacy systems with smart home platforms or other IoT devices can be complex and require additional hardware.

3.3. Requirement Specifications:

3.3.1 Functional requirements:

- Smoke detection: The system must use sensors such as MQ2 to detect smoke.

- Real-time monitoring: Provides continuous monitoring of smoke and smoke levels. system status.
- Alarm mechanism: When smoke is detected, an instant alarm is sent via mobile application or other communication method.
- Cloud integration: Connect to cloud platforms for data storage, analysis and remote access.
- Compatibility: Ensure compatibility with popular smart home platforms (such as HomeKit, Google Home, Alexa) for integration.
- User Interface: Create intuitive mobile applications for users to monitor and manage systems.

3.3.2 Functional Requirements:

- Reliability: The system must be reliable with minimum false alarms and downtime.
- Security: Use strong encryption for data transmission and user authentication to prevent unauthorized access.
- Scalability: Create a system that is scalable and suitable for many areas, from field to enterprise.
- Low power consumption: Increase the power consumption of the device, especially when the battery is used.
- Documentation: Provides information on installation, configuration and maintenance.

3.4. Feasibility Study:

3.4.1 Technical Feasibility:

- Hardware: NodeMCU, MQ2 sensor and other materials are easy and cheap.
- Software: Development platforms like Arduino IDE and cloud services IoT or Azure IoT like AWS are growing.

3.4.2 Operational Feasibility:

- User Training: The system should be user-friendly and training for end users should be minimal.
- Integration: The relationship with the smart home enables integration into the existing configuration.

3.4.3 Economic Feasibility:

- Cost Analysis: Evaluate hardware, cloud services, and construction costs.
- ROI: Measure the ability to stay safe and potentially save money in the event of fire.

3.4.4 Schedule Feasibility:

- Development Time: Create real-time for development, testing and deployment.
- Iterative development: Create new designs and update plans based on user feedback.

CHAPTER 4

Planning and Scheduling

What is planning?

Project Planning is the definition of the scope, objectives, tasks, resources, timelines, and deliverables for a particular project. This is that which will give information to the team on how they are going to run and follow this roadmap in all necessary steps of the process. An effective planning process builds a basis for smooth and successful execution of the project by making everyone in the team know their roles and obligations.

4.1 Scope Definition:

- Clearly define the scope of the project stating out the features and functionalities of the IoT-based smoke detection system.
- The environment being targeted should also be stated whether it is residential, commercial, or industrial type.

4.2 Setting of Objective:

- Ensure the aims of the systems are well set, for example, real-time monitoring of smoke, remote monitoring, integration with smart-home platforms, alerts to be in the shortest time and elicitation of requirements among others.

4.3 Identify the following Stakeholders:

- End-users, developers, regulatory authorities, and emergency responders.
- Identify their expectations from the system.

4.4 Risk Assessment:

- Identify the possible risks and ambiguities affecting the execution of the project.
- Developing strategies of risk minimization as well as contingency planning.

4.5 Resource Planning:

- Identification of the requisite resources required to execute the project including human, material, equipment, and technology.
- Efficient allotment of the available resources as per the requirement of the project.

4.6 Timeline Development:

- Prepare major milestones and deadlines in a project timeline/schedule.
- Consider the dependencies of the tasks and reasonable timespans to be assigned for these tasks.

4.7 Budgeting:

- Plan for the budget required for the project with regard to costs in terms of resources, materials, technology, as well as contingencies.
- Monitor and control the expenditure throughout the life of the project.

What is Scheduling?

A schedule in project management is a list of tasks, deliverables, and milestones in a project. A plan usually includes planned start and finish dates, time, and resources allocated to each task. Effective planning is the key to successful time management, especially for business services.

4.1 Work and damage analysis:

- Break down the project into activities such as hardware prototyping, software development, cloud integration, mobile application development and testing.
- Defining the tasks that need to be done for each job.

4.2 Business analysis:

- Identifies dependencies such as hardware prototyping before software development, software development before cloud integration.
- Understand the critical path to timely success.

4.3 Resource allocation:

- Allocate resources for each project, including availability of hardware, developers and cloud services.
- Provide assistance when needed to avoid delays.

4.4 Create a Timeline:

- Create a detailed work schedule with start and finish dates for each task.
- Visualize time showing work and critical path using tools such as Gantt charts.

4.5 Gantt chart or project network diagram:

- Create a Gantt chart or project network diagram to visualize work.
- Strengthen communication and understanding of members' and stakeholders' progress.

4.6 Monitoring and updating:

- Remember to monitor the progress of the study program.
- Adjust time as necessary to account for unintended delays, changes, or new risks.

4.7 Communication:

- Communicate the progress program to all partners and stakeholders.
- Keep everyone informed of progress, schedule adjustments, and potential impacts to project schedule.

CHAPTER 5

SYSTEM DESIGN

Data Flow Diagram

A Data Flow Diagram (DFD) is a graphical representation of the flow of data within a system. It illustrates how data moves through processes, stores, and external entities in a system. DFDs provide a visual way to understand the system's functionalities, data inputs and outputs, and the processes that transform the data. There are different levels of DFDs, ranging from a high-level overview to more detailed diagrams.

Key Components of a DFD:

1. **Processes (Rectangles):**
    - Represent the functions or activities that manipulate data.
    - Processes transform input data into output data.
2. **Data Stores (Parallel Lines):**
    - Represent storage locations for data within the system.
    - Data stores can be databases, files, or any repository where data is persistently stored.
3. **Data Flows (Arrows):**
    - Represent the movement of data between processes, data stores, and external entities.
    - Arrows indicate the direction of data flow.
4. **External Entities (Rectangles):**
    - Represent external entities that interact with the system but are not part of it.
    - External entities send or receive data from the system.

5.1 DFD of IoT-Based Fire Detection System using NodeMCU and MQ2 Sensor:

A simplified DFD for an IoT-based fire detection system using NodeMCU with an MQ2 sensor connected to a power supply, triggering a buzzer when smoke is detected.

Description:

1. **External Entity - User:**
    - Represents the end-user interacting with the system.
    - Sends commands or receives alerts through the mobile application.
2. **Process - IoT Device (NodeMCU):**
    - Represents the NodeMCU microcontroller.
    - Monitors the MQ2 sensor for smoke detection.
3. **External Entity - Power Supply:**
    - Represents the power supply providing electrical power to the NodeMCU.
4. **External Entity - MQ2 Sensor:**
    - Represents the MQ2 smoke sensor.
    - Detects smoke and sends signals to the NodeMCU.
5. **Process - Buzzer Activation:**
    - Represents the process of activating the buzzer when smoke is detected.
    - Produces an audible alert to notify users.
6. **Data Flow - Smoke Data:**
    - Represents the flow of data indicating the detection of smoke from the MQ2 sensor to the NodeMCU.
7. **Data Flow - Alert Signal:**
    - Represents the flow of data indicating the activation of the buzzer to produce an alert.

5.2 Level 1 DFD (Detailed Process):

Description:

1. **External Entity - User:**
    - Sends commands through the mobile app to the IoT Device.
2. **Process - IoT Device (NodeMCU):**
    - Monitors the MQ2 sensor for smoke detection.
    - Processes the smoke detection signal.
    - Activates the buzzer for an audible alert.
    - Sends an alert status to the user.
3. **External Entity - Power Supply:**
    - Provides electrical power to the NodeMCU.
4. **External Entity - MQ2 Sensor:**
    - Detects smoke and sends a signal to the NodeMCU.
5. **Process - Buzzer Activation:**
    - Activates the buzzer upon receiving the smoke detection signal.
6. **Data Flow - Smoke Data:**
    - Carries the smoke detection signal from the MQ2 sensor to the NodeMCU.
7. **Data Flow - Alert Signal:**
    - Carries the alert status from the NodeMCU to the User.

5.3 DATA FLOW DIAGRAM

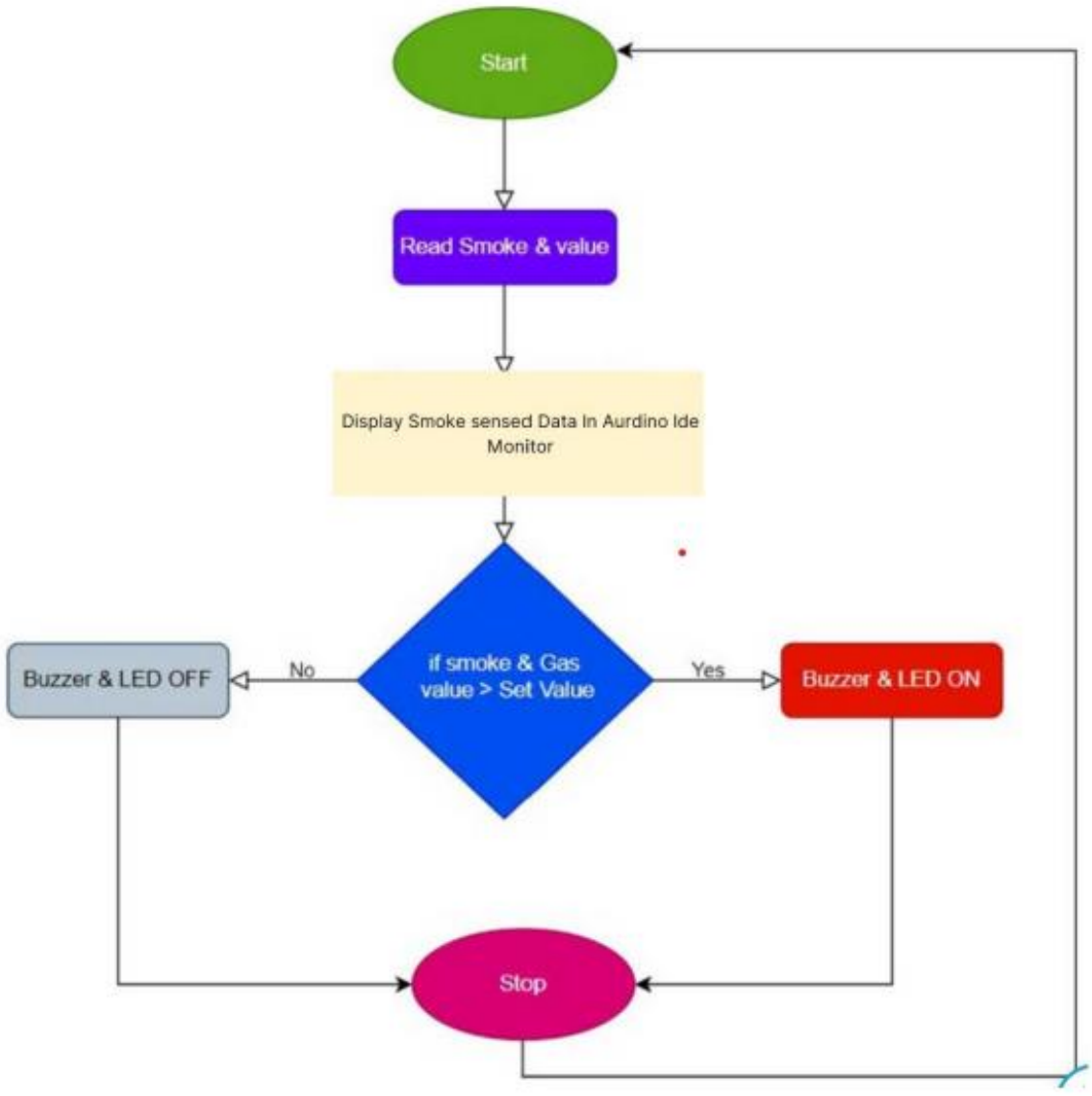

Fig 5.3: DFD Diagram

Block Diagram

A block diagram is a simplified representation of a system or process that uses blocks to represent major components or functions and arrows to depict the flow of signals or information between these components. It provides a high-level overview of the system's architecture, focusing on the interactions between different blocks without delving into the internal details of each block.

5.4 Key Components of a Block Diagram:

1. **Blocks:**
    - Represent major components or subsystems of the system.
    - Each block typically signifies a distinct function or module within the system.
2. **Arrows/Lines:**
    - Indicate the flow of signals, information, or energy between blocks.
    - Arrows show the direction of the flow, emphasizing the connections between different components.
3. **Labels:**
    - Provide descriptions or names for each block, explaining its function or purpose.
    - Labels help in understanding the role of each component within the system.

5.5 Advantages of Block Diagrams:

- **Simplicity:** Block diagrams provide a simple and clear representation of complex systems, making them easy to understand.
- **Communication:** They facilitate communication between different stakeholders by providing a visual representation that is easily interpretable.
- **Analysis:** Block diagrams are useful for analyzing the overall architecture of a system and identifying key relationships between components.

- **Design:** They aid in the design process by providing a conceptual framework before delving into the detailed design of individual components.

5.6 Block Diagram of IoT-Based Fire Detection System:

Now, let's create a block diagram for an IoT-based fire detection system using NodeMCU with an MQ2 sensor connected to a power supply, triggering a buzzer when smoke is detected.

5.7 Description:

1. **NodeMCU:**
    - Represents the NodeMCU microcontroller.
    - Responsible for interfacing with the MQ2 sensor, processing data, and activating the buzzer.
2. **MQ2 Sensor:**
    - Represents the MQ2 smoke sensor.
    - Detects smoke and sends a signal to the NodeMCU when smoke is detected.
3. **Power Supply:**
    - Represents the power source providing electrical power to the NodeMCU.
4. **Buzzer:**
    - Represents the audible alert system.
    - Activated by the NodeMCU upon detecting smoke.
5. **Arrows:**
    - Show the flow of information or signals:
        - From the MQ2 sensor to the NodeMCU (smoke detection signal).
        - From the NodeMCU to the Buzzer (activation signal).

5.8 Operation:

- The MQ2 sensor continuously monitors for smoke and sends a signal to the NodeMCU upon detection.
- The NodeMCU processes the signal and activates the buzzer to produce an audible alert.
- The system is powered by an external power supply.

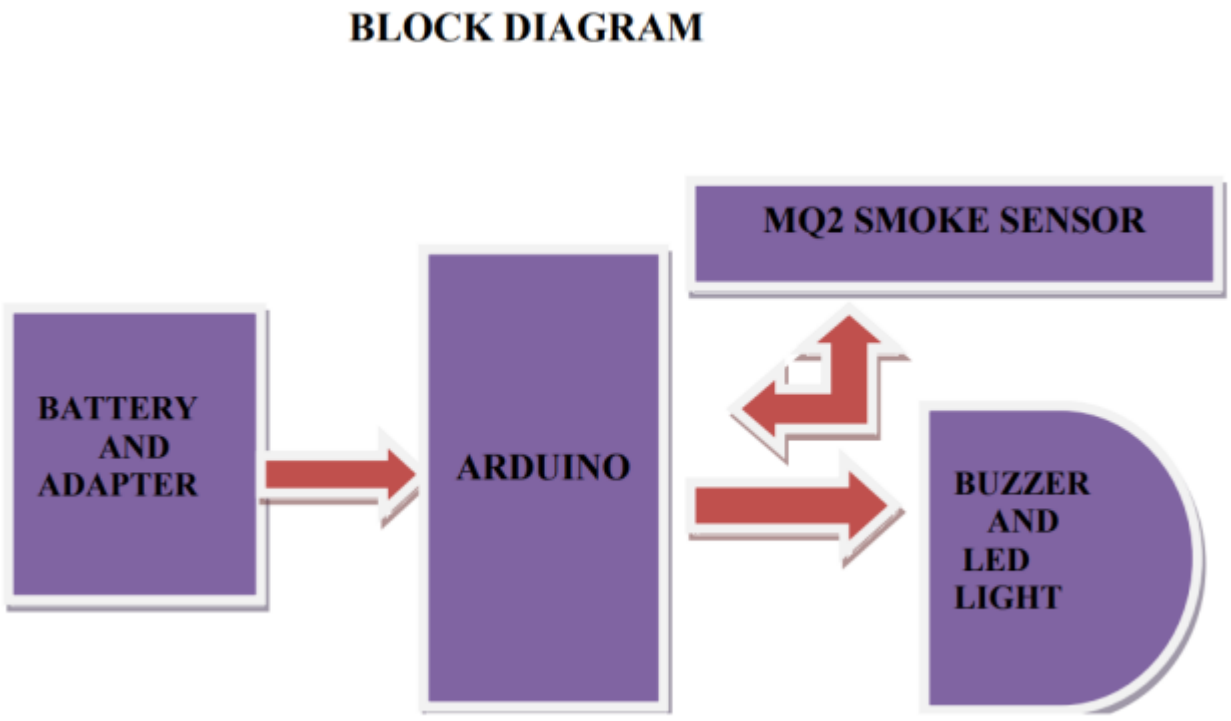

Fig 5.8: Block Diagram

CHAPTER 6

CODING SECTION

6.1 Arduino IDE CODE

```
#include <ESP8266WiFi.h>
#include "PubSubClient.h"

const char *ssid =  "WIFI_SSID";
const char *pass =  "WIFI_PASSWORD";
const char* mqttServer = "broker.hivemq.com";
int mqttPort = 1883;

WiFiClient espClient;
PubSubClient client(espClient);

int buzzer = D2;
int smokeA0 = A0;

// Your threshold value. You might need to change it.
int sensorThres = 500;

void setup() {
 pinMode(buzzer, OUTPUT);
 pinMode(smokeA0, INPUT);
 Serial.begin(115200);
  Serial.println("Starting...");
  delay(1000);

  WiFi.begin(ssid, pass);
   while (WiFi.status() != WL_CONNECTED) {
    delay(500);
```

```
    Serial.print(".");
  }

  Serial.println("Connected to the WiFi network");
   client.setServer(mqttServer, mqttPort);
   while (!client.connected()) {

    Serial.println("Connecting to MQTT...");
     if (client.connect("parking-device-01")) {
       Serial.println("connected");
     } else {
      Serial.print("failed with state ");
      Serial.print(client.state());
      delay(2000);
    }
  }
}

void loop() {
 int analogSensor = analogRead(smokeA0);

 Serial.print("Pin A0: ");
 Serial.println(analogSensor);
 // Checks if it has reached the threshold value
 if (analogSensor > sensorThres)
 {
   tone(buzzer, 1000, 200);
   String payload = "Smoke Detected!";
   client.publish("smoke_status", payload.c_str());
 }
 else
 {
   noTone(buzzer);
 }
 delay(100);
 client.loop();
}
```

6.2 ALERT SYSTEM CODE

```
<html>
<head>
<script src="https://cdnjs.cloudflare.com/ajax/libs/paho-mqtt/1.0.1/mqttws31.min.js"
type="text/javascript"></script>
<style>
 #status{
   background-color:#FBC77F;
   width:250px;
   height:150px;
   text-align: center;
   margin: auto;
   padding: 50px;
   box-shadow: rgba(6, 24, 44, 0.4) 0px 0px 0px 2px, rgba(6, 24, 44, 0.65) 0px 4px 6px -
1px, rgba(255, 255, 255, 0.08) 0px 1px 0px inset;
 }
</style>
</head>
<body>
 <h1 style="text-align: center;">Smoke Sensor</h1>
<div id="status"><h3 id="msg">NA</h3></div>

<script type="text/javascript">
// Create a client instance

client = new Paho.MQTT.Client("broker.hivemq.com", 8000 ,"smoke-client-001");

// set callback handlers
client.onConnectionLost = onConnectionLost;
client.onMessageArrived = onMessageArrived;

// connect the client
client.connect({onSuccess:onConnect});
// called when the client connects
```

```
function onConnect() {
  // Once a connection has been made, make a subscription and send a message.
 console.log("onConnect");
 client.subscribe("smoke_status");
}

// called when the client loses its connection
function onConnectionLost(responseObject) {
  if (responseObject.errorCode !== 0) {
    console.log("onConnectionLost:"+responseObject.errorMessage);
  }
}

// called when a message arrives
function onMessageArrived(message) {
  console.log("onMessageArrived:"+message.payloadString);
  console.log("onMessageArrived:"+message.destinationName);
  const now = new Date();
  const currentDateTime = now.toLocaleString();
  if(message.payloadString=="Smoke Detected!"){

    document.getElementById("msg").innerHTML=message.payloadString + "<br>On: "
+ currentDateTime;
  } else{
    document.getElementById("msg").innerHTML="NA";
  }
}
</script>

</body>

</html>
```

CHAPTER 7

ALL BASIC CODE REQUIREMENTS AND WORKING

1. All Required Libraries for NodeMCU
   - AVision_ESP8266 1.0.6 Installed
   - PubSubClient 2.8 Installed
   - Wi-Fi on BAND 2.4GHZ

2. The output will be displayed on webpage using JavaScript.

3. Using MQTT
   - MQTT (Message Queuing Telemetry Transport) is essential for enabling communication between the smoke detectors and the web-based interface in Internet of Things-based smoke detection systems.
   - MQTT is a quick and effective communications system that works great with Internet of Things applications.
   - In this case, MQTT messages would be used by smoke detectors with IoT capability to regularly deliver status updates or alarms to a central server.
   - On the web interface, JavaScript may be used to subscribe to the MQTT topic associated with smoke detection. The smoke detector sends a message to the MQTT topic when detecting smoke or any other relevant event.
   - These MQTT messages are received by the webpage's JavaScript code, which then dynamically adjusts the webpage's content to reflect the smoke detectors' current state.
   - Users may access real-time updates on the webpage with the integration of JavaScript and MQTT, giving them instant access to smoke detector status information.
   - The smoke detection system's monitoring and alerting capabilities are improved by the smooth connection between IoT devices and the web interface, enabling users to respond promptly in the event of a possible fire threat.

4. Using JavaScript for output
   - A basic webpage for MQTT smoke sensor status monitoring is created using HTML and JavaScript code. The status is shown in a paragraph that has been styled inside of a specified division.
   - The homepage dynamically refreshes to reflect the event when the smoke detector detects smoke and delivers a message to the "smoke status" MQTT topic.

5. **The code's description is as follows:**
   - The homepage is composed of a styled div with the id "status," and a header with the headline "Smoke Sensor."
   - The status message is displayed in this div, which also has a h3 element with the id "msg" for the purpose of serving as the status display area.
   - Using the client ID "smoke-client-001," the JavaScript code creates a MQTT connection with the broker at "broker.hivemq.com" on port 8000. After a successful connection, it subscribes to the "smoke status" topic.
   - There are three specified callback functions:
   - onConnect: Started as soon as the MQTT client establishes a connection. It subscribes to the "smoke status" topic and records the connection status.
   - When the client loses connection, the onConnectionLost function is called. If there is an error message, it logs it.
   - The onMessageArrived function is called. It changes the site when it detects if the payload is "Smoke Detected!"
   - The message is shown along with the timestamp, and the background color of the status div is changed to indicate smoke detection.
   - The code also incorporates a transition effect to enhance the background color change user encounter.
   - The smoke sensor status may be seen in real-time on the webpage by embedding this code into an HTML file and opening it in a web browser. When smoke is discovered, the styled div with the id "status" will dynamically change its background color, giving a clear visual cue.

6. Extensions Used for Output in Visual Studio Code on webpage.
   - Install the **Live Server Extension**: To install the Live Server extension, follow these steps if you haven't already:
   - Launch the Visual Studio Code.

- Press Ctrl+Shift+X to open the Extensions view or click the Extensions symbol in the Activity Bar on the side of the window.
- Utilizing the Extensions view search box, look up "Live Server".
- Install Ritwick Dey's "Live Server" plugin.
- Get Your HTML File Opened:
- Launch Visual Studio Code and open your HTML file.
- Select "Open with Live Server" from the menu.
- Anywhere in your HTML file, do a right-click.
- Click the context menu and choose "Open with Live Server."
- Click Here to Visit Your Live Server:
- When you choose "Open with Live Server," your usual web browser will create a new tab with the live version of
- Usually, the URL looks like this: http://127.0.0.1:5500/your-file-name.html.
- Open your web browser and navigate to this URL to view the live output of your IoT smoke detecting application.
- You may now see the changed output in real-time by changing your HTML or JavaScript files, which will instantly cause a page to reload. This is particularly helpful for dynamically updating the homepage with the smoke sensor status as MQTT messages are received.
- During development, Live Server offers a convenient method to examine and interact with your web application in real-time, but it has no effect on your actual code files.

CHAPTER 8

LIVE SERVER EXTENSION SCREENSHOT

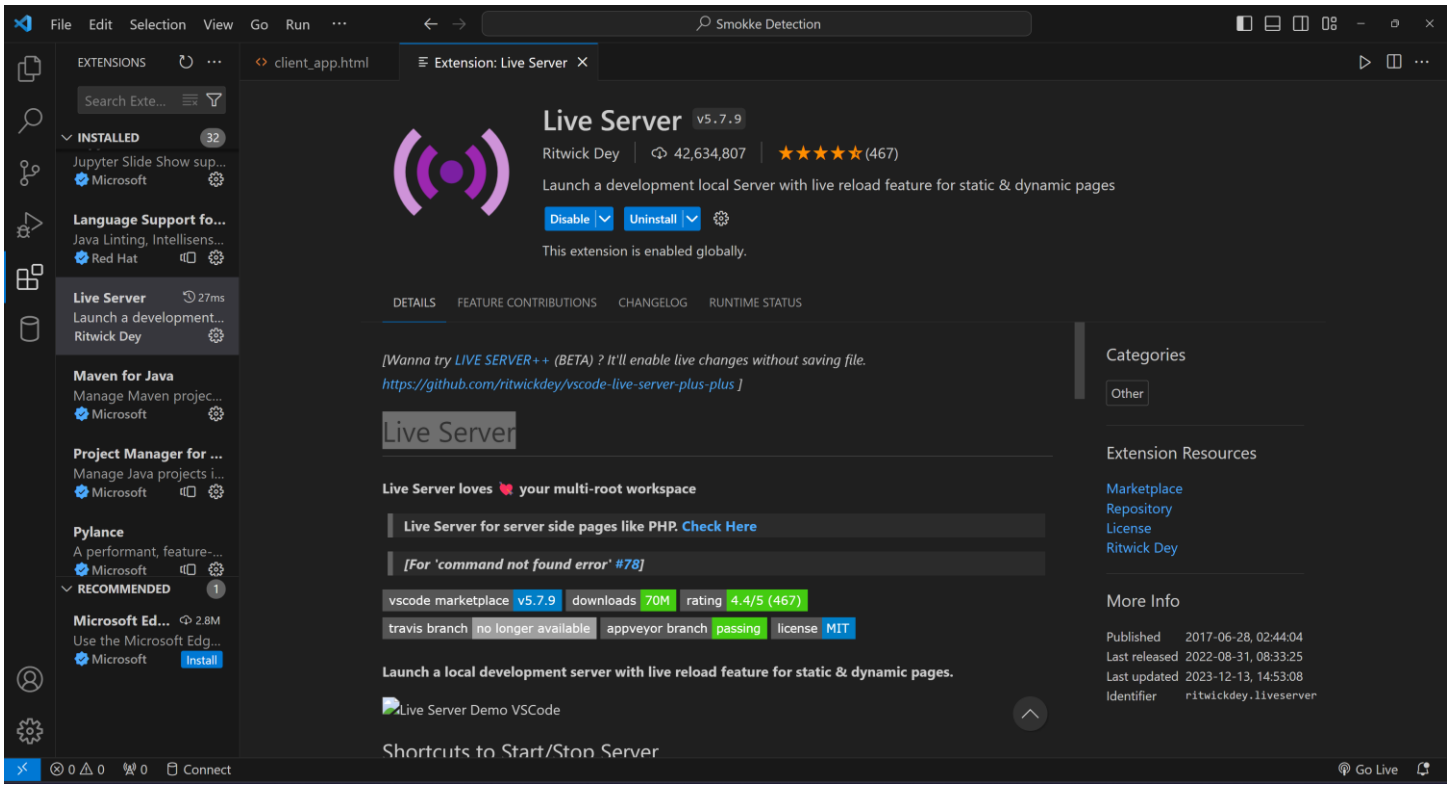

Fig 8.1: Live Server Extension

OUTPUT SCREENSHOT

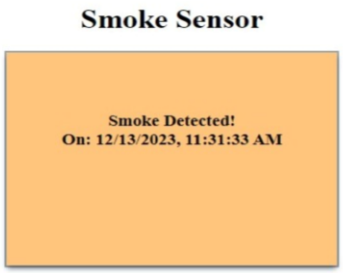

Fig 8.2: Output

HARDWARE SETUP

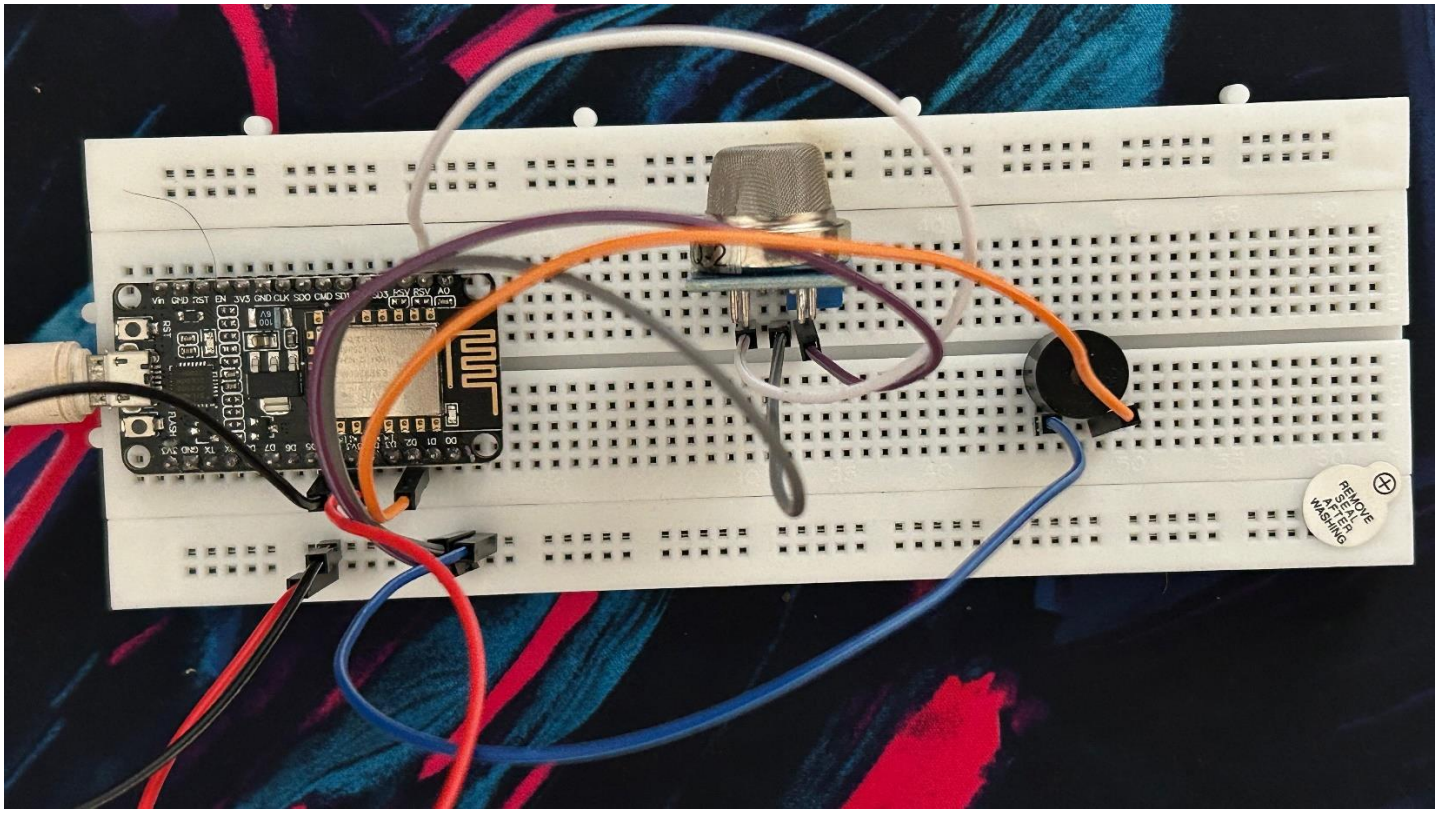

Fig 8.3: Hardware Setup

CHAPTER 9
Testing , Limitation & Future Scope

1. Functional Testing:

- Verify that the NodeMCU correctly interfaces with the MQ2 sensor.
- Ensure the NodeMCU processes the smoke detection signal accurately.
- Confirm that the buzzer is activated upon detecting smoke.

2. Smoke Detection Accuracy:

- Test the system under various conditions to ensure accurate smoke detection by the MQ2 sensor.
- Evaluate the sensor's sensitivity and specificity.

3. Communication Testing:

- Ensure seamless communication between the NodeMCU, MQ2 sensor, and the buzzer.
- Test the system's ability to send alerts promptly.

4. Power Supply Reliability:

- Verify that the power supply provides a stable and reliable source of power for the NodeMCU.
- Test the system's response to power interruptions.

5. Mobile App Integration:

- If applicable, test the integration with the mobile app, ensuring real-time monitoring and alerting functionalities.

6. Environmental Testing:

- Test the system's performance under different environmental conditions, such as temperature and humidity variations.

9.1 Limitations of the IoT-Based Fire Detection System:

1. Limited Detection Range:

- The detection range of the MQ2 sensor may be limited, affecting the system's coverage.

2. False Positives and Negatives:

- The MQ2 sensor may produce false positives or negatives, leading to inaccurate smoke detection.

3. Dependency on Wi-Fi Connectivity:

- The system's functionality may be impacted by the availability and reliability of Wi-Fi connectivity for the NodeMCU.

4. Power Dependency:

- The system is dependent on a stable power supply, which may limit its use in areas with unreliable power sources.

5. Single Point of Detection:

- The system may have limitations in detecting fires at multiple locations simultaneously.

9.2 Future Scope for the IoT-Based Fire Detection System:

1. Multi-Sensor Integration:

- Integrate multiple sensors to enhance detection accuracy and cover a broader area.

2. Machine Learning Algorithms:

- Implement machine learning algorithms to improve the system's ability to distinguish between false alarms and real threats.

3. Redundancy and Fault Tolerance:

- Implement redundancy mechanisms to ensure system reliability, even in the case of sensor or component failures.

4. Enhanced Communication Protocols:

- Explore advanced communication protocols for more robust and secure data transmission.

5. Integration with Emergency Services:

- Develop features to automatically alert emergency services, providing a faster response to potential fire incidents.

6. Remote Monitoring and Control:

- Enable remote monitoring and control of the system through cloud platforms for user convenience.

7. Energy-Efficient Solutions:

- Explore energy-efficient solutions, such as low-power modes or alternative power sources, to extend the system's operational life.

8. Mobile App Enhancements:

- Enhance the mobile app with additional features, such as historical data analysis and user customization.

9. Global Standards Compliance:

- Ensure compliance with global standards and regulations related to fire detection systems for broader adoption.

10. Integration with Smart Home Ecosystems:

- Integrate the system with popular smart home ecosystems for seamless interoperability with other smart devices.

CHAPTER 10

REFERENCES

1. Smoke Sensor Types: National Fire Protection Association (NFPA): https://www.nfpa.org/education-and-research/home-fire-safety/smoke-alarms.
2. Sensor Accuracy and Sensitivity: Comparison of Photoelectric and Ionization Smoke Detectors: https://blog.koorsen.com/what-is-the-difference-between-ionization-and-photoelectric-smoke-alarms.
3. Communication Protocols: A Survey of Internet of Things Connectivity Technologies: https://link.springer.com/article/10.1007/s10796-014-9492-7.
4. Case Studies and Real-World Deployments: IoT-based Smart Homes for Fire Safety: https://www.arcweb.com/blog/iot-and-analytics-can-add-smartness-fire-detection.
5. Research Papers and Technical Articles: Recent Advances in Smoke Detection Technologies: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC9100504/.
6. Online platforms like IEEE Xplore and ACM Digital Library offer access to numerous research papers and technical articles on IoT, fire safety, and sensor technology.