Name : SAIDEEP RAHUL KULKARNI

College : PRAVARA RURAL ENGINEERING COLLEGE, LONI

Project : IOT-Based Gas Detector Using Arduino

Components :

Hardware Components

- **Arduino Uno** – Microcontroller for processing data.
- **MQ-2 Gas Sensor** – Detects gas concentrations.
- **Ethernet Shield** – Enables internet connectivity.
- **Buzzer** – Provides an audible warning.
- **Power Supply (5V)** – Powers the circuit.

Software Requirements

- **Arduino IDE** – Programming the Arduino.
- **ThingSpeak** – Cloud platform for data visualization.
- **SMTP (Email Service)** – Sends email alerts

Description :

Gas leakage is a serious safety hazard in residential and industrial areas. This project aims to develop an IoT-based gas detection and alert system using an Arduino, a gas sensor, and an Ethernet shield. The system continuously monitors gas levels and sends alerts via email when dangerous levels are detected. Additionally, data is uploaded to **ThingSpeak** for real-time monitoring.

Working Principle :

1. Gas Detection Mechanism

1.1 How the MQ-5 Sensor Works

The MQ-5 sensor is a metal oxide semiconductor gas sensor that operates on the principle of chemisorption.

Inside the sensor, there is a heating element and a sensing material (SnO_2 - Tin Dioxide).

When gas molecules like LPG, methane, or natural gas come into contact with the sensor, they get adsorbed on the surface, changing the resistance of the material.

The resistance change produces a corresponding voltage output, which is read by the Arduino.

1.2 Gas Level Measurement

The sensor outputs an analog voltage proportional to the gas concentration.

The Arduino reads this voltage through Analog Pin A0 and converts it into a numerical value.

A predefined threshold value (250 ppm in the code) determines whether the gas level is dangerous.

2. Buzzer Alert Mechanism

2.1 Buzzer Activation Condition

If the gas concentration exceeds 250 ppm, the system triggers an alarm.

The buzzer is connected to Digital Pin 3 of the Arduino.

The digital output is set to HIGH, activating the buzzer.

2.2 Buzzer Reset Condition

If the gas concentration falls below 250 ppm, the buzzer is turned OFF by setting the output to LOW.

---

3. Internet Communication and Cloud Integration

3.1 Ethernet Shield for Internet Connectivity

The system uses an Ethernet Shield (W5100) to connect to the internet via DHCP (Dynamic Host Configuration Protocol).

The MAC address of the shield ensures the device gets a unique IP address.

3.2 Uploading Gas Data to ThingSpeak

Every 15 seconds, the system uploads gas sensor readings to ThingSpeak for remote monitoring.

The data is sent in HTTP POST format with the API key for authentication.

This allows users to view gas levels in real-time from anywhere in the world.

3.3 Reconnection Handling

If the Ethernet connection fails, the system automatically attempts to reconnect.

If the connection fails more than 3 times, the Arduino restarts the Ethernet interface.

---

4. Email Notification System

4.1 When an Email is Sent

If the gas level exceeds 250 ppm, an email alert is triggered. The email contains:

Subject: "Gas Alert!"

Message: "Gas levels exceeded safe limits! Take action immediately."

4.2 SMTP-Based Email Sending

The email is sent using Simple Mail Transfer Protocol (SMTP).

The system connects to an SMTP server (e.g., smtp.gmail.com, smtp.example.com).

Uses base64-encoded login credentials for authentication.

4.3 Email Failsafe Mechanism

If the email fails to send, the system retries the connection.

After multiple failures, it logs the failure in Serial Monitor.

---

5. System Stability Using Watchdog Timer

5.1 Purpose of Watchdog Timer

Embedded systems can sometimes freeze or crash due to:

Internet failures

High sensor noise

Power fluctuations

To handle this, the system uses the AVR watchdog timer (WDT), which:

Automatically resets the Arduino if it becomes unresponsive.

Prevents the system from remaining in a faulty state for too long.

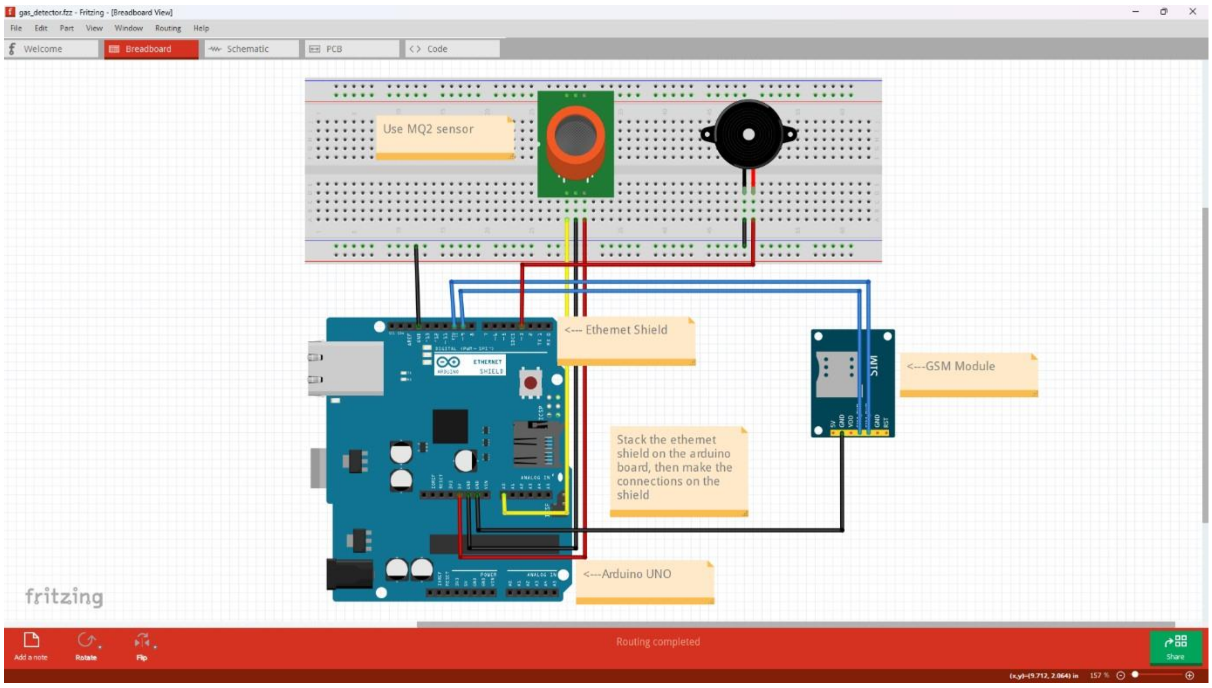
5.2 Watchdog Timer Implementation

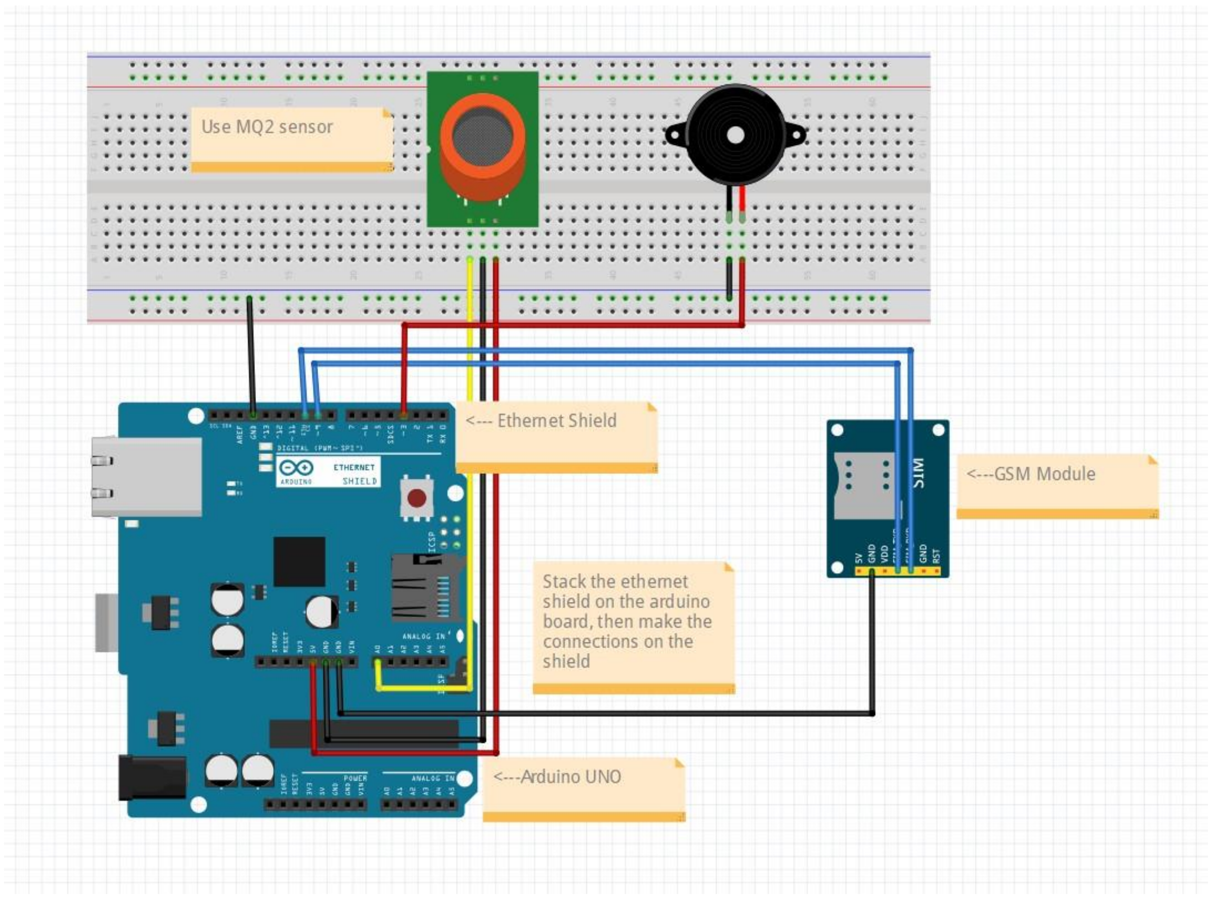
The watchdog timer is set for 8 seconds.

In every loop, wdt_reset(); is called to prevent an unnecessary reset.

If the Arduino crashes and does not reset the watchdog, it automatically restarts after 8 seconds.

Schematic Diagram :


gas_detector.fzz - Fritzing - [Breadboard View]
File Edit Part View Window Routing Help
Welcome
Breadboard
Schematic
PCB
Code
Use MQ2 sensor
<--- Ethernet Shield
<---GSM Module
Stack the ethernet shield on the arduino board, then make the connections on the shield
<---Arduino UNO
fritzing
Add a note
Rotate
Flip
Routing completed
Share


Use MQ2 sensor
<--- Ethernet Shield
<---GSM Module
Stack the ethernet shield on the arduino board, then make the connections on the shield
<---Arduino UNO
ETHERNET SHIELD
SIM

Code Explaination :

1. Libraries and Definitions

```
#include <SPI.h>

#include <Ethernet.h>

#include <EthernetClient.h>

#include <avr/wdt.h> // Watchdog timer for stability
```

- SPI.h: Enables communication with the Ethernet shield.
- Ethernet.h: Allows the Arduino to connect to a network via Ethernet.
- EthernetClient.h: Provides functions to connect to remote servers (ThingSpeak, SMTP).
- avr/wdt.h: Enables the watchdog timer, which resets the Arduino if it crashes.

```
#define ETHERNET_SHIELD_MAC {0xC0, 0x26, 0x1B, 0x5A, 0x70, 0x26}
```

- Defines the MAC address for the Ethernet shield. This must be unique in a network.

2. Sensor and Network Setup

```
const int gasPin = A0; const
int buzz = 3;
```

- gasPin is the analog input pin connected to the gas sensor.
- buzz is the digital output pin controlling the buzzer.

```
byte ethernetMACAddress[] = ETHERNET_SHIELD_MAC;
EthernetClient client;
```

- ethernetMACAddress[]: Stores the MAC address.
- EthernetClient client; is used for network communication.

---

3. ThingSpeak Setup

```
char thingSpeakAddress[] = "api.thingspeak.com"; String
writeAPIKey = "KE099C98QMLUZR0G";
const int updateThingSpeakInterval = 15000; // 15 seconds
(API limit)
```

- thingSpeakAddress[]: Stores the URL of ThingSpeak’s API.

•

•

writeAPIKey: Required for writing data to ThingSpeak.

updateThingSpeakInterval: Sets a 15-second interval for updating ThingSpeak.

---

4. Global Variables long

lastConnectionTime = 0;

boolean lastConnected =

false; int failedCounter = 0;

- lastConnectionTime: Stores the last time ThingSpeak was updated.
- lastConnected: Keeps track of the last connection state.
- failedCounter: Counts failed connection attempts to reconnect when necessary.

---

5. setup() Function void

setup() {   pinMode(buzz,

OUTPUT);

Serial.begin(9600);

startEthernet();

wdt_enable(WDTO_8S); // Enable watchdog timer (resets Arduino if frozen)

```
    .

    .
}
```

Sets the buzzer pin as an output.

Starts serial communication for debugging.

- Initializes the Ethernet connection using startEthernet().
- Enables the watchdog timer to reset the Arduino if it crashes for more than 8 seconds.

---

6. loop() Function void loop()

```
   {
   wdt_reset(); // Reset watchdog timer to prevent unwanted
resets    updateCloud();
   int gasValue = analogRead(gasPin);
   Serial.println("Gas Sensor Value: " + String(gasValue));

   if (gasValue > 250) {
digitalWrite(buzz, HIGH);
sendEmail();
   } else {
      digitalWrite(buzz, LOW);
   }
   delay(1000);
```

.

.

}

wdt_reset() prevents unnecessary resets.

updateCloud() uploads the sensor value to ThingSpeak.

- Reads the gas sensor value using analogRead().
- Prints the gas level to the Serial Monitor.
- If gas concentration is above 250, it:
  - Turns ON the buzzer.
  - Sends an email alert using sendEmail().
- If gas concentration is safe, it turns OFF the buzzer.
- Pauses execution for 1 second (delay(1000);).

---

7. Uploading Sensor Data to ThingSpeak void

```
updateCloud() {
   String gasData = "field1=" + String(analogRead(A0));     if
(!client.connected() && (millis() - lastConnectionTime >
updateThingSpeakInterval)) {
updateThingSpeak(gasData);
   }
   lastConnected = client.connected();
}
```

-
-
- Formats the gas sensor value into a ThingSpeakcompatible string.

  Checks if enough time has passed since the last update.

  Calls updateThingSpeak() to send the data.

```
void updateThingSpeak(String tsData) {     if
(client.connect(thingSpeakAddress, 80)) {
client.print("POST /update HTTP/1.1\n");
client.print("Host: api.thingspeak.com\n");
client.print("Connection: close\n");
       client.print("X-THINGSPEAKAPIKEY: " + writeAPIKey +
"\n");
       client.print("Content-Type: application/x-www-
formurlencoded\n");
       client.print("Content-Length: ");
client.println(tsData.length());
client.println();        client.println(tsData);
lastConnectionTime = millis();
       Serial.println("Data sent to ThingSpeak");
failedCounter = 0;
```

```
    .

    .
    } else {
failedCounter++;
        Serial.println("ThingSpeak connection failed (" +
String(failedCounter) + ")");
```

```
        if (failedCounter > 3) startEthernet();
    }
    client.stop();
}
```

- Connects to ThingSpeak.
- Sends the gas data using an HTTP POST request.
- If successful, updates lastConnectionTime and resets failedCounter.
- If the connection fails multiple times, it restarts Ethernet.

---

8. Connecting to the Network void startEthernet() {

```
client.stop();
    Serial.println("Connecting to network...");     if
(Ethernet.begin(ethernetMACAddress) == 0) {
        Serial.println("DHCP Failed, retrying...");
    } else {
        Serial.println("Connected to network");
    }
    delay(1000);
}
```

- Stops any previous client connections. • Tries to obtain an IP address via DHCP.
- Retries if DHCP fails.

---

9. Sending Email Alerts void

```
sendEmail() {
   Serial.println("Sending Email Alert...");
   if (client.connect("smtp.example.com", 587)) { // Replace
with SMTP provider
      client.println("EHLO smtp.example.com");
client.println("AUTH LOGIN");
      client.println("BASE64_ENCODED_USERNAME"); //
Replace with actual base64 encoded username
client.println("BASE64_ENCODED_PASSWORD"); // Replace
with actual base64 encoded password
client.println("MAIL FROM: <you@example.com>");
client.println("RCPT TO: <recipient@example.com>");
client.println("DATA");
      client.println("Subject: Gas Alert!\n");
      client.println("Gas levels exceeded safe limits! Take
action immediately.\n.");        client.println("QUIT");
      Serial.println("Email Sent!");
   } else {
        Serial.println("Email sending failed");
    }
    client.stop();
}
```

- Connects to an SMTP server (replace with a real SMTP provider like Gmail, Mailgun, or SMTP2GO).
- Authenticates using base64-encoded credentials (avoid storing these in code!).
- Sends an email with a gas alert message.
- Closes the connection.

NOTE :

ARDUINO AND FRITZING FILES ARE UPLOADED ON GITHUB REPOSITORY