R. P. Shaha University

- an institution of Kumudini Welfare Trust of Bengal (BD) Ltd.

Department: COMPUTER SCIENCE AND ENGINEERING
Semester: Spring 2024

Program: Bachelor of Computer Science and Engineering

Course Title: Peripherals and Interfacing Lab

Course Code: CSE 332

Course Teacher: Tanjina Akter

Project Name : Iot Based Home Automation

Group –1

Members	**ID**
1. Sayid Mahabur Rahman	**21300036**
2. Navid Hasan Radiat	**21300059**
3. Sabbir Ahmed	**21300070**

Submission Date: 29.06.24

Iot-Based Home Automation

Overview

IoT-based home automation refers to the use of Internet of Things (IoT) technology to control and manage home appliances and systems remotely. Home automation has become increasingly popular as it offers convenience, energy efficiency, and enhanced security. This approach leverages connected devices, cloud services, and voice assistants to create a smart home environment that enhances convenience, energy efficiency, and security.

Objective

The objective of this project is to design and implement an IoT-based home automation system that leverages the NodeMCU microcontroller, relay modules, Sinric Pro cloud service, and voice assistants like Alexa and Google Home. The system aims to provide users with seamless remote and voice-controlled management of home appliances, enhancing convenience, energy efficiency, and security. Specifically, this project seeks to achieve the following:

- **Remote Control**: Enable users to operate home appliances from anywhere using a smartphone or web interface.
- **Voice Control**: Integrate with Alexa and Google Home to allow users to control appliances through natural voice commands.
- **Energy Management**: Reduce energy consumption by scheduling appliance operation and remotely turning off unused devices.
- **Ease of Use**: Ensure a user-friendly setup and operation process, making home automation accessible to a wide range of users.

- **Scalability**: Provide a flexible framework that can be easily expanded to include additional appliances and sensors as needed.
- **Cost-Effectiveness**: Utilize affordable and readily available components to create a budget-friendly home automation solution.

Project Components List

- **NodeMCU**: An open-source IoT platform based on the ESP8266 Wi-Fi module. It serves as the central controller, connecting to the internet and communicating with other devices.

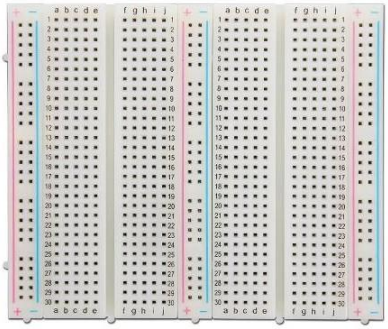
- **Relay Module**: An electronic switch controlled by the NodeMCU to turn appliances on and off.

- **Sinric Pro**: A cloud-based service that facilitates the integration of IoT devices with smart home ecosystems, offering APIs for seamless connectivity.

- **Alexa/Google Home**: Voice-controlled smart assistants that allow users to manage their home automation system through natural language commands.
-
- **Power Supply**: Both the NodeMCU and relay modules require a stable power supply, typically provided via USB or external power adapters.

- **Jumper Wires:** Jumper wires are electrical wires with connector pins at each end, used to interconnect components on a breadboard or to connect components to other devices without the need for soldering.

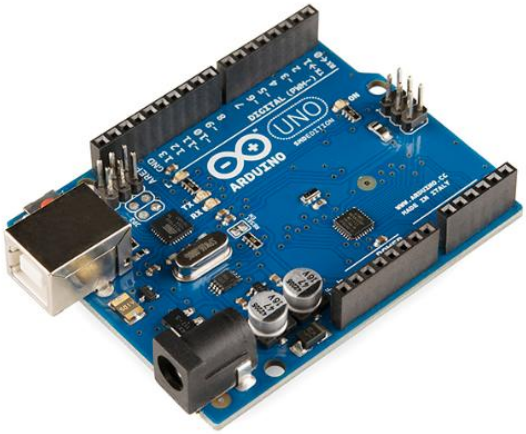
- **Breadboard:** A breadboard is a rectangular board with a grid of interconnected sockets, allowing for the easy construction and modification of temporary electronic circuits without soldering.



- **Switch:** Used for controlling the home appliances directly

- **Light Bulbs:** Used as appliances to demonstrate the project

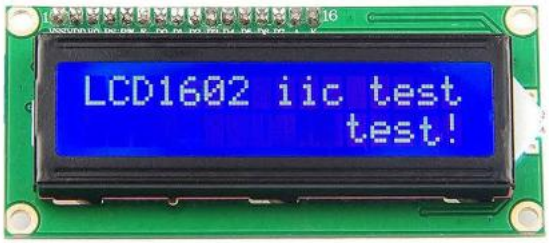
- **Arduino UNO:** A powerful and user-friendly microcontroller board that provides a flexible platform for developing a wide array of electronic projects. Its ease of use, extensive community support, and rich feature set make it an excellent choice for both beginners and experienced developers.



- **I2C display module:** A user-friendly and efficient solution

for adding text display capabilities to microcontroller projects. Its I2C interface reduces wiring complexity and makes it easy to integrate into a wide range of applications, from educational tools to sophisticated embedded systems.

- **I2C Liquid Crystal Display (LCD):** An I2C Liquid Crystal Display (LCD) module is a type of character or graphic display that uses the I2C (Inter-Integrated Circuit) communication protocol for interfacing with microcontrollers. These modules are commonly used in electronics projects to provide visual output and information display.

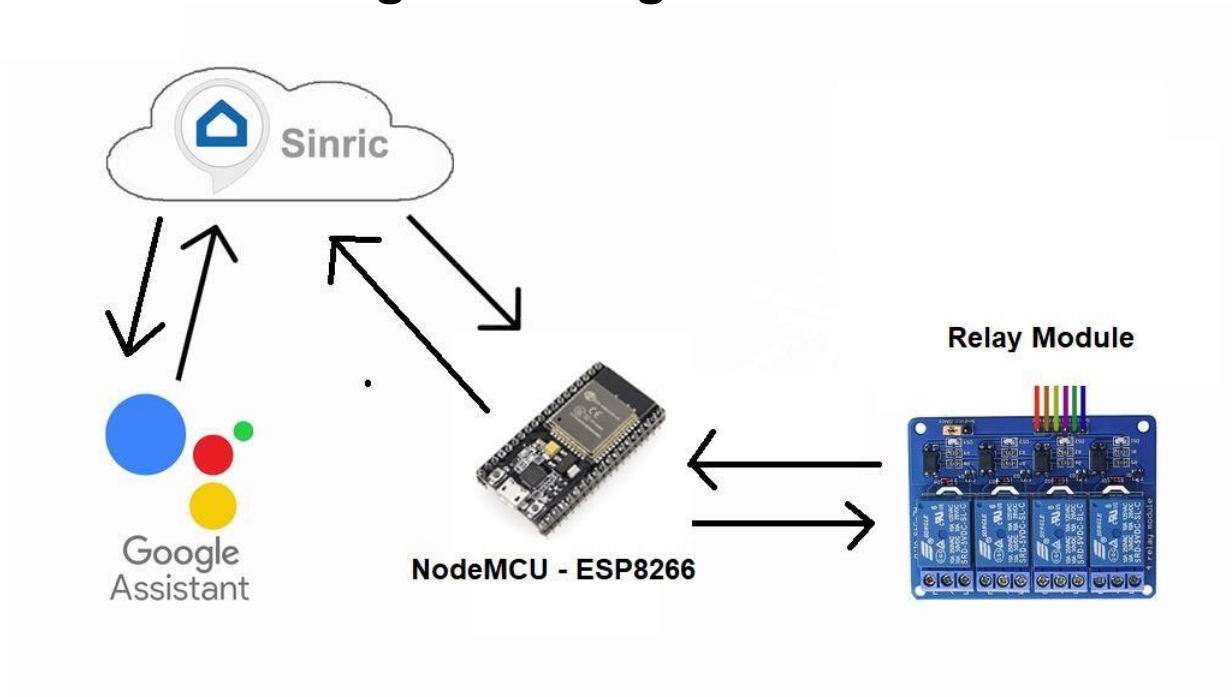


- **MQ-6 gas sensor:** The MQ-6 gas sensor is a

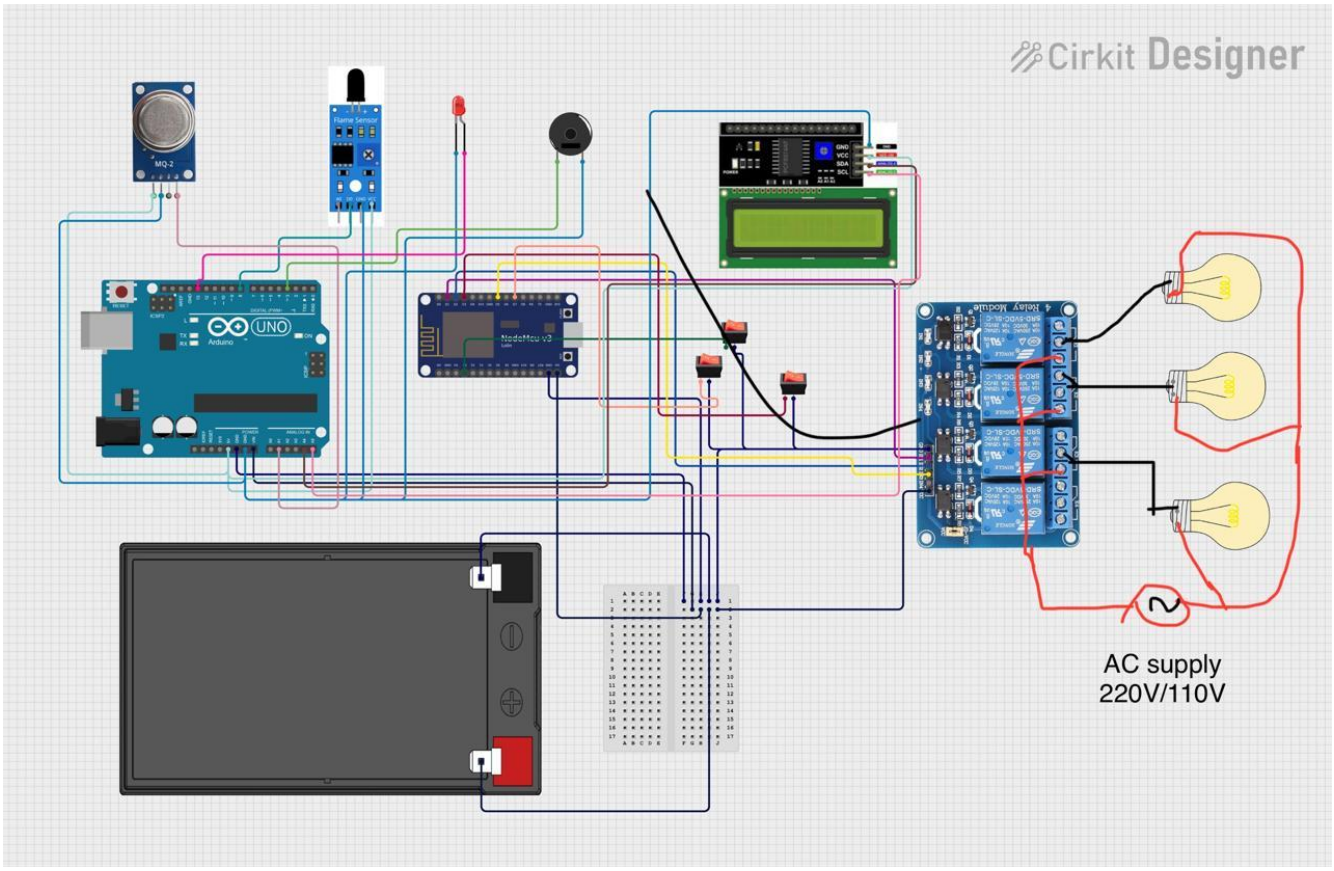
semiconductor sensor designed to detect the presence of combustible gas, such as LPG (liquefied petroleum gas), butane, propane, methane, and other gases in the air. It is commonly used in gas leakage detection systems and other safety applications

- **IR Flame Detector:** An IR Infrared 4-Wire Flame Sensor is a module designed to detect the presence of flame or other infrared sources with wavelengths between 760 nm and 1100 nm. It is commonly used in fire detection and safety systems.

- **Process Diagram for light on and off**



- **Circuit Diagram**



- # Arduino Code for NodeMCU

```
//#define ENABLE_DEBUG

#ifdef ENABLE_DEBUG
    #define DEBUG_ESP_PORT Serial
    #define NODEBUG_WEBSOCKETS
    #define NDEBUG
#endif

#include <Arduino.h>
#include <ESP8266WiFi.h>
#include "SinricPro.h"
#include "SinricProSwitch.h"

#include <map>

#define WIFI_SSID         "The Bat-Wifi"
#define WIFI_PASS         "home88++"
#define APP_KEY           "fa9d98bc-addc-4949-ba0a-4141e10c616e"   // Should look like "de0bxxxx-1x3x-4x3x-ax2x-5dabxxxxxxxx"
#define APP_SECRET        "510a2326-5df1-4df9-b6da-077f8e14fd53-6cbf54e1-867d-4c32-adce-6e7c393c8086"   // Should look like "5f36xxxx-x3x7-4x3x-xexe-e86724a9xxxx-4c4axxxx-3x3x-x5xe-x9x3-333d65xxxxxx"

//Enter the device IDs here
#define device_ID_1  "667bf3ba5d818a66fabddeb6"
#define device_ID_2 "667bf48c5d818a66fabddf3c"
#define device_ID_3  "667bf4fb888aa7f7a23cdc90"
//#define device_ID_4  "60764aa148ccc14a4674c047"

// define the GPIO connected with Relays and switches
#define RelayPin1 5  //D1
```

```
#define  RelayPin2  4  //D2
#define RelayPin3 14 //D5
#define RelayPin4 12 //D6

#define SwitchPin1 10  //SD3
#define SwitchPin2 0 //D3
#define SwitchPin3 13  //D7
//#define SwitchPin4 3  //RX

#define wifiLed  16  //D0

// comment the following line if you use a toggle switches instead of tactile buttons
//#define TACTILE_BUTTON 1

#define BAUD_RATE  9600

#define DEBOUNCE_TIME 250

typedef struct {      // struct for the std::map below
  int relayPIN;
  int flipSwitchPIN;
} deviceConfig_t;



// this is the main configuration
// please put in your deviceId, the PIN for Relay and PIN for flipSwitch
// this can be up to N devices...depending on how much pin's available on your device ;)
// right now we have 4 devicesIds going to 4 relays and 4 flip switches to switch the relay manually
std::map<String, deviceConfig_t> devices = {
    //{deviceId, {relayPIN,  flipSwitchPIN}}
    {device_ID_1, {  RelayPin1, SwitchPin1 }},
    {device_ID_2, {  RelayPin2, SwitchPin2 }},
    {device_ID_3, {  RelayPin3, SwitchPin3 }}
    //{device_ID_4, {  RelayPin4, SwitchPin4 }}
```

```
};

typedef struct {      // struct for the std::map below
  String deviceId;
  bool lastFlipSwitchState;
  unsigned long lastFlipSwitchChange;
} flipSwitchConfig_t;

std::map<int, flipSwitchConfig_t> flipSwitches;     // this map is used to map flipSwitch PINs to deviceId and handling debounce and last flipSwitch state checks
                                                    // it will be setup in "setupFlipSwitches" function, using informations from devices map

void setupRelays() {
  for (auto &device : devices) {          // for each device (relay, flipSwitch combination)
    int relayPIN = device.second.relayPIN; // get the relay pin
    pinMode(relayPIN, OUTPUT);            // set relay pin to OUTPUT
    digitalWrite(relayPIN, HIGH);
  }
}

void setupFlipSwitches() {
  for (auto &device : devices)  {                  // for each device (relay / flipSwitch combination)
    flipSwitchConfig_t flipSwitchConfig;           // create a new flipSwitch configuration

    flipSwitchConfig.deviceId = device.first;      // set the deviceId
    flipSwitchConfig.lastFlipSwitchChange = 0;     // set debounce time
    flipSwitchConfig.lastFlipSwitchState = true;   // set lastFlipSwitchState to false (LOW)--

    int flipSwitchPIN = device.second.flipSwitchPIN;  // get the flipSwitchPIN

    flipSwitches[flipSwitchPIN] = flipSwitchConfig;    // save the flipSwitch config to flipSwitches map
    pinMode(flipSwitchPIN, INPUT_PULLUP); }            // set the flipSwitch pin to INPUT
```

```
}

bool onPowerState(String deviceId, bool &state)
{
  Serial.printf("%s: %s\r\n", deviceId.c_str(), state ? "on" : "off");
  int relayPIN = devices[deviceId].relayPIN; // get the relay pin for corresponding device
  digitalWrite(relayPIN, !state);             // set the new relay state
  return true;
}

void handleFlipSwitches() {
  unsigned long actualMillis = millis();                                    // get actual millis
  for (auto &flipSwitch : flipSwitches) {                                   // for each flipSwitch in
flipSwitches map
    unsigned long lastFlipSwitchChange = flipSwitch.second.lastFlipSwitchChange;   // get the
timestamp when flipSwitch was pressed last time (used to debounce / limit events)

    if (actualMillis - lastFlipSwitchChange > DEBOUNCE_TIME) {              // if time is >
debounce time...

      int flipSwitchPIN = flipSwitch.first;                                 // get the flipSwitch pin from
configuration
      bool lastFlipSwitchState = flipSwitch.second.lastFlipSwitchState;     // get the
lastFlipSwitchState
      bool flipSwitchState = digitalRead(flipSwitchPIN);                    // read the current
flipSwitch state
      if (flipSwitchState != lastFlipSwitchState) {                         // if the flipSwitchState has
changed...
#ifdef TACTILE_BUTTON
        if (flipSwitchState) {                                              // if the tactile button is pressed
#endif
          flipSwitch.second.lastFlipSwitchChange = actualMillis;            // update
lastFlipSwitchChange time
          String deviceId = flipSwitch.second.deviceId;                     // get the deviceId from
config
```

```
        int relayPIN = devices[deviceId].relayPIN;                          // get the relayPIN from
config
        bool newRelayState = !digitalRead(relayPIN);                       // set the new relay State
        digitalWrite(relayPIN, newRelayState);                              // set the trelay to the new
state

        SinricProSwitch &mySwitch = SinricPro[deviceId];                   // get Switch device from
SinricPro
        mySwitch.sendPowerStateEvent(!newRelayState);                       // send the event
#ifdef TACTILE_BUTTON
      }
#endif
      flipSwitch.second.lastFlipSwitchState  =  flipSwitchState;                              // update
lastFlipSwitchState
    }
  }
 }
}

void setupWiFi()
{
 Serial.printf("\r\n[Wifi]: Connecting");
 WiFi.begin(WIFI_SSID, WIFI_PASS);

 while (WiFi.status() != WL_CONNECTED)
 {
  Serial.printf(".");
  delay(250);
 }
 digitalWrite(wifiLed, LOW);
 Serial.printf("connected!\r\n[WiFi]: IP-Address is %s\r\n", WiFi.localIP().toString().c_str());
}

void setupSinricPro() {
```

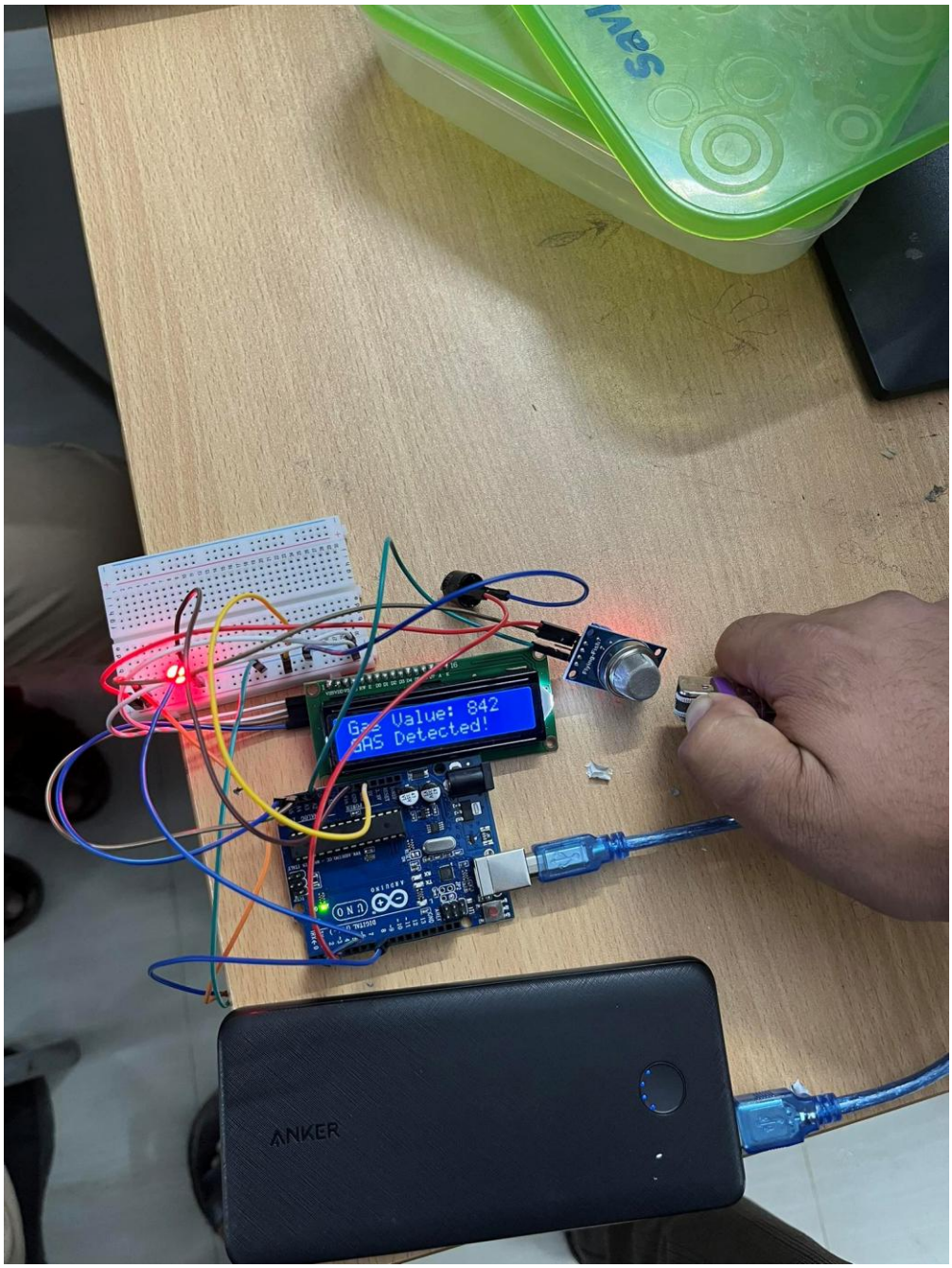
```
  for (auto &device : devices)
  {
    const char *deviceId = device.first.c_str();
    SinricProSwitch &mySwitch = SinricPro[deviceId];
    mySwitch.onPowerState(onPowerState);
  }

  SinricPro.begin(APP_KEY, APP_SECRET);
  SinricPro.restoreDeviceStates(true);
}

void setup()
{
  Serial.begin(BAUD_RATE);

  pinMode(wifiLed, OUTPUT);
  digitalWrite(wifiLed, HIGH);

  setupRelays();
  setupFlipSwitches();
  setupWiFi();
  setupSinricPro();
}

void loop()
{
  SinricPro.handle();
  handleFlipSwitches();}
```

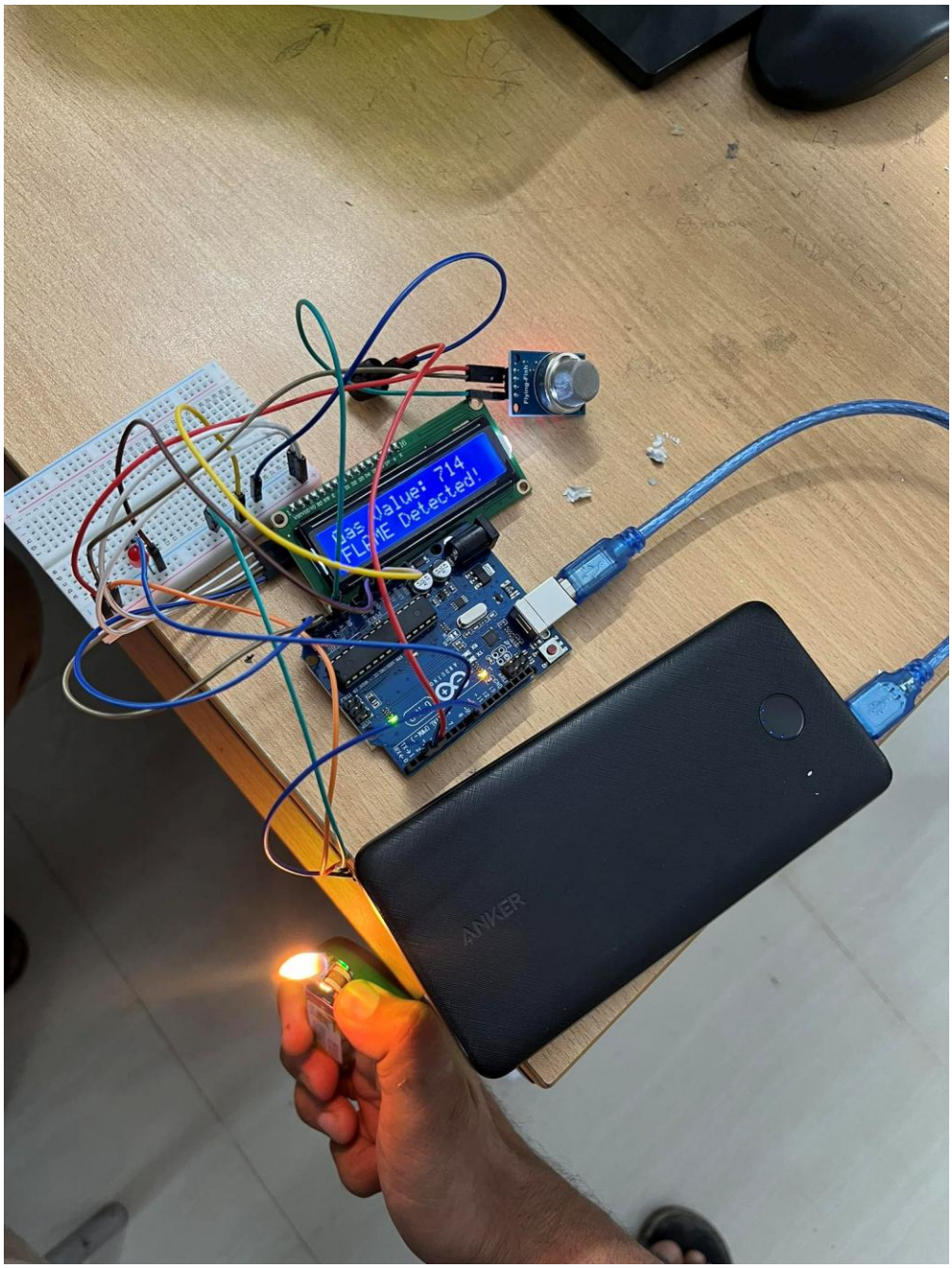
Arduino uno Code:

```
#include <LiquidCrystal_I2C.h>

LiquidCrystal_I2C lcd(0x27, 16, 2);

#define LED 2
#define Buzzer 3
#define GasSensor A1
#define FlameSensor 8
#define FlameLED 13

void setup() {
  Serial.begin(9600);
  lcd.init();
  lcd.backlight();

  pinMode(LED, OUTPUT);
  pinMode(Buzzer, OUTPUT);
  pinMode(FlameSensor, INPUT);
  pinMode(FlameLED, OUTPUT);
}

void loop() {
  int gasValue = analogRead(GasSensor);
  int flameValue = digitalRead(FlameSensor);

  lcd.setCursor(0, 0);
  lcd.print("Gas Value: ");
  lcd.print(gasValue);
  lcd.print("   ");

  if (gasValue > 800) { // Gas detected
    digitalWrite(LED, HIGH);
```

```
    digitalWrite(Buzzer, HIGH);
    lcd.setCursor(0, 1);
    lcd.print("GAS Detected!   ");
  } else {
    digitalWrite(LED, LOW);
    lcd.setCursor(0, 1);
    lcd.print("                ");
  }

  if (flameValue == LOW) { // Flame detected
    Serial.println("FLAME, FLAME, FLAME");
    digitalWrite(FlameLED, HIGH);
    digitalWrite(Buzzer, HIGH);
    lcd.setCursor(0, 1);
    lcd.print("FLAME Detected! ");
  } else {
    Serial.println("No flame");
    digitalWrite(FlameLED, LOW);
  }

  // Ensure the buzzer turns off if neither gas nor flame is detected
  if (gasValue <= 400 && flameValue != LOW) {
    digitalWrite(Buzzer, LOW);
  }

  delay(1000);
}
```

Output


Gas Value: 842
GAS Detected!
ANKER


Gas Value: 714
FLAME Detected!
ANKER

Output (Video)

Link:

https://drive.google.com/file/d/1KGJRihaGxXnxaRFbig5VDrv4YJq7MJQS/view?fbclid=IwZXh0bgNhZW0CMTAAAR31CBSVK-Ab0ebj-DBohWrA38auz9c-YQC_bsmjxz0INDoWCpl0c7ncgbo_aem_F0gN7r9y6gcvq4GVocVZ0A