INDOOR AIR QUALITY SYSTEM WITH AQI MEASUREMENTS

INTRODUCTION:

This project monitors environmental conditions and air quality by combining physical parameters (temperature, pressure, humidity, altitude) with pollutant detection (CO_2, NH_3, alcohol, benzene, LPG). The consolidated data enables reliable Air Quality Index (AQI) estimation.

SENSORS USED AND THEIR PARAMETERS:

BME688	Temperature, Pressure, Humidity, Altitude
MQ-2	LPG
MQ-135	CO2, NH3, Alcohol, Benzene

PRINCIPLE:

This project works by using sensors to detect environmental conditions and gases. The BME688 measures temperature, humidity, pressure, and air quality, while the MQ-135 and MQ-2 detect different pollutant and combustible gases. The ESP32 reads these sensor signals, processes them into values like ppm and AQI, and displays the results in real time to show the air quality and ensure safety.

WORKING:

The sensors BME688, MQ-135, and MQ-2 detect environmental conditions and different gases. These signals are read by the ESP32, which converts them into useful values like ppm and AQI. The processed results are then displayed in real time, giving a clear picture of air quality and ensuring safety.

BME688:

- **Measures temperature, humidity, pressure, and gas in one sensor.**
- **Provides Indoor Air Quality (IAQ) index using Bosch's algorithm.**
- **Small, low-power, and easy to connect with ESP32.**
- **Useful for air quality monitoring, weather stations, and smart IoT devices.**

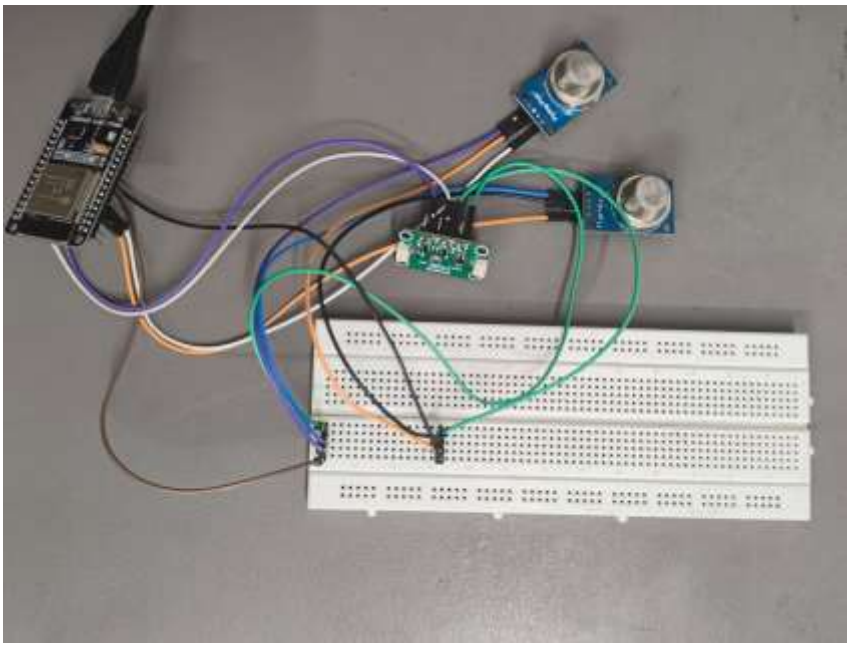
MQ2:

- **Detects LPG, methane, propane, hydrogen, alcohol, and smoke.**
- **Gives an analog output that changes with gas concentration.**
- **Low cost and easy to use with microcontrollers like ESP32.**
- **Commonly used for gas leakage detection and safety systems.**

MQ135:

- **Detects gases like CO_2, NH_3, alcohol, benzene, and smoke.**
- **Provides an analog output that changes with gas concentration.**
- **Useful in pollution detection, safety systems, and IoT applications.**

PROTOTYPE:



CODING:

```
#include <Wire.h>

#include <SPI.h>

#include <Adafruit_Sensor.h>

#include <bsec.h>

#include <MQUnifiedsensor.h>


/************************ Hardware Configuration (MQ Sensors) ************************/

#define Board              "ESP32"

#define Voltage_Resolution  3.3

#define ADC_Bit_Resolution  12


// MQ-135 on GPIO34

#define PinMQ135          34

#define TypeMQ135          "MQ-135"


// MQ-2 on GPIO35 (choose another ADC pin)

#define PinMQ2            35

#define TypeMQ2            "MQ-2"
```

```
#define RatioMQ2CleanAir    9.83


MQUnifiedsensor MQ135(Board, Voltage_Resolution,
ADC_Bit_Resolution, PinMQ135, TypeMQ135);
MQUnifiedsensor MQ2(Board, Voltage_Resolution,
ADC_Bit_Resolution, PinMQ2, TypeMQ2);


/************************ AQI Helpers (MQ Sensors)
************************/
String getAQICategory_CO2(float ppm) {
  if (ppm <= 400) return "Excellent";
  else if (ppm <= 1000) return "Good";
  else if (ppm <= 2000) return "Moderate";
  else if (ppm <= 5000) return "Poor";
  else return "Hazardous";
}


int getSimulatedAQIScore(float ppm) {
  if (ppm < 200) return map(ppm, 0, 200, 0, 50);
  else if (ppm < 400) return map(ppm, 200, 400, 51, 100);
  else if (ppm < 800) return map(ppm, 400, 800, 101, 200);
```

```
  else if (ppm < 1600) return map(ppm, 800, 1600, 201, 300);
  else return map(ppm, 1600, 5000, 301, 500);
}

String getSimulatedAQICategory(int score) {
  if (score <= 50) return "Good";
  else if (score <= 100) return "Satisfactory";
  else if (score <= 200) return "Moderate";
  else if (score <= 300) return "Poor";
  else if (score <= 400) return "Very Poor";
  else return "Severe";
}

/************************ BME688 Global Variables and Setup **********************/
#define SEALEVELPRESSURE_HPA (1013.25)

float pressure;
float Pressure;
float altitude;
```

```
// I2C address of the BME688
const uint8_t BME688_I2C_ADDRESS = BME68X_I2C_ADDR_HIGH;

Bsec iaqSensor;

// AQI stabilization flag
bool aqiSettled = false;

float calculateAltitude(float pressure, float seaLevelPressure) {
   return 44330 * (1.0 - pow((pressure / seaLevelPressure), 0.1903));
}

void configureBSEC() {
  // Subscribe to IAQ and environmental outputs
  bsec_virtual_sensor_t sensorList[] = {
    BSEC_OUTPUT_IAQ,
    BSEC_OUTPUT_SENSOR_HEAT_COMPENSATED_TEMPERATURE,
    BSEC_OUTPUT_SENSOR_HEAT_COMPENSATED_HUMIDITY,
    BSEC_OUTPUT_RAW_PRESSURE,
```

```
  };

  // Configure for 2-second sampling interval
  iaqSensor.updateSubscription(sensorList, 4, BSEC_SAMPLE_RATE_LP);

  if (iaqSensor.bsecStatus < BSEC_OK) {
    Serial.println("Error while subscribing to BSEC outputs.");
  }
}

// Renamed BME functions to be called by the master loop
void bme_setup_func() {
  Serial.begin(115200);
  Wire.begin();

  // Initialize BSEC library
  iaqSensor.begin(0x77, Wire);

  if (!iaqSensor.bsecStatus) {
    Serial.println("BSEC library initialized successfully!");
```

```
  } else {
    Serial.print("BSEC initialization failed! Error code: ");
    Serial.println(iaqSensor.bsecStatus);
    while (true); // Stop execution if initialization fails
  }

  configureBSEC();
}

void bme_loop_func() {
  // Check if the BSEC library is running
  if (iaqSensor.run()) {
    // Stabilize AQI
    if (!aqiSettled) {
      Serial.print("Waiting for AQI to stabilize... Current AQI: ");
      Serial.println(iaqSensor.iaq);

      if (iaqSensor.iaq != 50) { // Ensure AQI accuracy is sufficient
        aqiSettled = true;
        Serial.println("AQI has stabilized. Reading data every 2 seconds.");
```

```
    }
   }
   else {
    pressure = iaqSensor.pressure/100;
    altitude = calculateAltitude(pressure, SEALEVELPRESSURE_HPA);
    // Once AQI is stable, read all parameters (Printing removed here to avoid double prints, moved to master loop)
  }
  }
  else {
   // Handle BSEC errors if necessary
  }
}



// Renamed MQ functions to be called by the master loop
void mq_setup_func() {
  // Serial.begin(115200); // Already called in bme_setup_func
  // Wire.begin(); // Already called in bme_setup_func
```

```
// MQ-135 init
MQ135.setRegressionMethod(1); // Exponential
MQ135.init();
MQ135.setRL(10); // Load resistance in kOhms
float calcR0_135 = 0;
for (int i = 0; i < 50; i++) {
  MQ135.update();
  calcR0_135 += MQ135.calibrate(0.4); // 400 ppm CO2 baseline
  delay(100);
}
MQ135.setR0(calcR0_135 / 50.0);
Serial.print("MQ-135 Calibrated R0: "); Serial.println(MQ135.getR0());

// MQ-2 init
MQ2.setRegressionMethod(1); // Exponential
MQ2.init();
MQ2.setA(574.25); MQ2.setB(-2.222); // LPG curve
float calcR0_2 = 0;
for (int i = 0; i < 10; i++) {
  MQ2.update();
```

```
    calcR0_2 += MQ2.calibrate(RatioMQ2CleanAir);
    delay(1000);
  }
  MQ2.setR0(calcR0_2 / 10.0);
  Serial.print("MQ-2 Calibrated R0: "); Serial.println(MQ2.getR0());

  if (isinf(calcR0_2) || calcR0_2 == 0) {
    Serial.println("MQ-2 Calibration error. Check wiring.");
    while (true);
  }
}

void mq_loop_func() {
  // MQ-135 readings
  MQ135.update();
  MQ135.setA(110.47); MQ135.setB(-2.862); float co2 =
MQ135.readSensor();
  MQ135.setA(102.2);  MQ135.setB(-2.473); float nh3 =
MQ135.readSensor();
  MQ135.setA(77.255); MQ135.setB(-3.18);  float alcohol =
MQ135.readSensor();
```

```
  MQ135.setA(44.947); MQ135.setB(-3.445); float benzene =
MQ135.readSensor();
  String aqiCO2 = getAQICategory_CO2(co2);

  // MQ-2 readings
  MQ2.update();
  float lpg_ppm = MQ2.readSensor(false);
  int aqiScore = getSimulatedAQIScore(lpg_ppm);
  String aqiCategory = getSimulatedAQICategory(aqiScore);

  // Print results
  Serial.println("===== Combined Sensor Readings =====");

  // Print BME data within the MQ print block to consolidate output
timing
  if (aqiSettled) {
    Serial.println("-- BME688 Data --");
    Serial.print("Temperature: "); Serial.print(iaqSensor.temperature);
Serial.println(" °C");
    Serial.print("Pressure: "); Serial.print(iaqSensor.pressure / 1000.0);
Serial.println(" kPa");
```

```
    Serial.print("Humidity: "); Serial.print(iaqSensor.humidity);
Serial.println(" %");
    Serial.print("AQI (BME): "); Serial.println(iaqSensor.iaq);
    Serial.print("Altitude: "); Serial.print(altitude); Serial.println(" m");
  } else {
    Serial.println("BME688 AQI is stabilizing...");
  }


  Serial.println("-- MQ Sensor Data --");
  Serial.print("MQ-135 -> CO2eq: "); Serial.print(co2); Serial.print(" ppm
| AQI: "); Serial.println(aqiCO2);
  Serial.print("MQ-135 -> NH3: "); Serial.print(nh3); Serial.print(" ppm |
Alcohol: "); Serial.print(alcohol);
  Serial.print(" ppm | Benzene: "); Serial.println(benzene);


  Serial.print("MQ-2 -> LPG: "); Serial.print(lpg_ppm); Serial.print(" ppm
| AQI Score: ");
  Serial.print(aqiScore); Serial.print(" | AQI Category: ");
Serial.println(aqiCategory);
```

```
  Serial.println("----------------------------------");
  // delay(3000); // Original delay kept
}



/************************ Master Setup and Loop ************************/
void setup() {
  // Call all individual setup functions
  bme_setup_func(); // Initializes Serial and Wire
  mq_setup_func();
}

void loop() {
  // Call all individual loop functions
  bme_loop_func();
  mq_loop_func();
  // The delay(3000) from the original MQ loop is executed inside mq_loop_func
}
```

TESTING AND RESULT: