AIR QUALITY MONITORING USING IOT

INTRODUCTION:

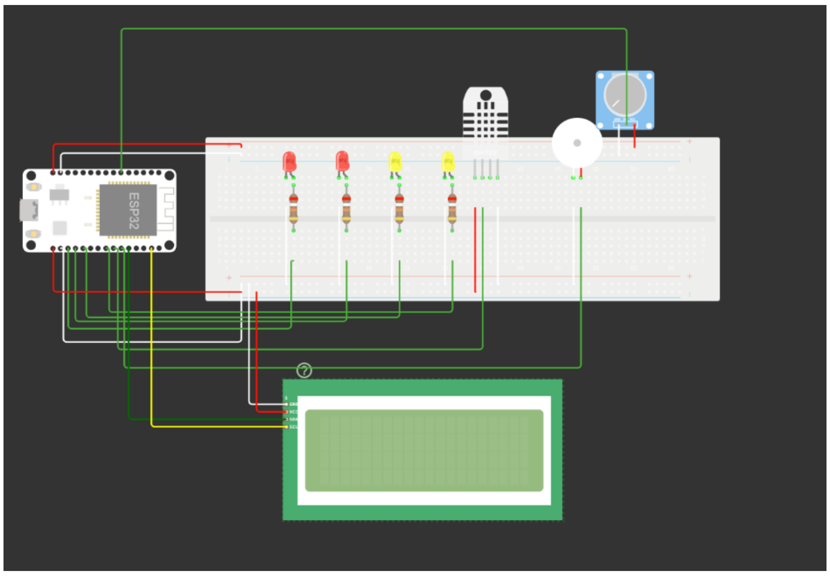
Air quality monitoring using IoT (Internet of Things) is a modern approach to assess and manage the quality of the air we breathe. It involves the use of IoT technology to collect, analyze, and share real-time data related to air quality, making it more accessible and actionable for individuals, communities, and organizations. Here's an introduction to the concept of air quality monitoring using IoT:

1. Importance of Air Quality Monitoring:

- Air quality is a critical factor that directly impacts human health, the environment, and various industries. Poor air quality can lead to respiratory problems, environmental degradation, and economic consequences.
- Monitoring air quality is essential for identifying pollution sources, assessing the effectiveness of pollution control measures, and providing timely warnings during air quality events.

Project Design:

The development of this system via XAMPP platform allows the air quality level in parts per million (ppm) data to be stored in an online database, thus allowing the public to continuously monitor the air quality level and avoid themselves to be exposed rapidly to these harmful gases. The developed hardware system consists of the MQ135 gas sensor. The gas sensor is able to sense the present of gases through the chemical reaction when the gases flows close to the sensor. The reading of air quality level appears on an I2C LCD.



SOFTWARE DEVELOPMENT:

To develop software for the DHT22 (DHT11 or DHT21) temperature and humidity sensor and the MQ-135 gas sensor, you can use a microcontroller or development board, such as an Arduino, ESP8266, or ESP32. Below, I'll provide a basic example for each sensor, showing how to read data from them. You can use these examples as a starting point for your own projects.

The Code For the above sensors:

```
#include "DHTesp.h"

#include <WiFi.h>

#include "PubSubClient.h"

#include <LiquidCrystal_I2C.h>

//#include "MQ135.h"


const char* ssid = "Wokwi-GUEST";

const char* password = "";

const char* mqttServer = "test.mosquitto.org";

const int port = 1883;


const int LED_PIN_1 = 15;

const int LED_PIN_2 = 2;

const int LED_PIN_3 = 4;

const int LED_PIN_4 = 5;
const int DHT22_PIN = 18;

const int BUZZER_PIN = 19;

const int mq135_simulator = 32;


DHTesp dhtSensor;

WiFiClient espClient;

PubSubClient client(espClient);

LiquidCrystal_I2C LCD = LiquidCrystal_I2C(0x27, 20, 4);


// decorations while connecting to wifi.

void spinner()

{

  static int8_t counter = 0;

  const char* glyphs = "\xa1\xa5\xdb";

  LCD.setCursor(15, 1);

  LCD.print(glyphs[counter++]);

  if (counter == strlen(glyphs)) {

    counter = 0;

  }

}

// connect to wifi.

void wifiConnect()

{

  WiFi.begin(ssid, password);

  while (WiFi.status() != WL_CONNECTED) {

    delay(500);

    spinner();

    Serial.print(".");

  }

  Serial.println(" Connected");

}
```

```
// activate BUZZER_PIN on sin value
// void activate_BUZZER_PIN()
// {
//   for(int x = 0; x < 180; x++)
//   {
//     float sinVal = sin(x*(3.1412/180));
//     int toneVal = 2000 + int(sinVal * 1000);
//     tone(BUZZER_PIN, toneVal);
//     delay(2);
//   }
// }
// deactivate BUZZER_PIN.
// initialize LCD
void LCD_init()
{
  LCD.init();
  LCD.backlight();
  LCD.setCursor(0, 0);
  LCD.print("Connecting to ");
  LCD.setCursor(0, 1);
  LCD.print("WiFi ");
}
// print local IP on LCD after connected.
void LCD_show_localIP()
{
  LCD.clear();
  LCD.setCursor(0, 0);
  LCD.print("Online");
  LCD.setCursor(0, 1);
  LCD.print("Local IP: ");
  LCD.print(WiFi.localIP());
  LCD.setCursor(0, 2);
  LCD.print("Getting data in ");
  LCD.print("3");
  delay(1000);
  LCD.setCursor(16, 2);
  LCD.print("2");
  delay(1000);
  LCD.setCursor(16, 2);
  LCD.print("1");
  delay(1000);
  LCD.clear();
}
// Print data on LCD
void LCD_handle(int temperature, int humidity, int mq135_value)
{
  LCD.setCursor(0, 0);
  LCD.print("Current data: ");
  LCD.setCursor(0, 1);
  LCD.print("Temperature: ");
  LCD.print(String(temperature) + "oC");
  LCD.setCursor(0, 2);
  LCD.print("Humidity: ");
  LCD.print(String(humidity) + "%");
  LCD.setCursor(0, 3);
  LCD.print("MQ135: ");
  LCD.print(String(mq135_value) + "PPM");
}

void Turn_On_All_Led()
{
  digitalWrite(LED_PIN_1, HIGH);
  digitalWrite(LED_PIN_2, HIGH);
  digitalWrite(LED_PIN_3, HIGH);
  digitalWrite(LED_PIN_4, HIGH);
}

void Turn_Off_All_Led()
{
  digitalWrite(LED_PIN_1, LOW);
  digitalWrite(LED_PIN_2, LOW);
  digitalWrite(LED_PIN_3, LOW);
  digitalWrite(LED_PIN_4, LOW);
}

void Turn_On_Warning_Led()
{
  digitalWrite(LED_PIN_1, LOW);
  digitalWrite(LED_PIN_2, LOW);
  digitalWrite(LED_PIN_3, HIGH);
  digitalWrite(LED_PIN_4, HIGH);
}

float Read_ppm_data()
{
  int value = analogRead(mq135_simulator);

  int to_5V_value = map(value, 0, 4095, 0, 1023);

  float to_ppm = to_5V_value * 1200 / 1023;

  return to_ppm;
}
```

```
void Warning(float temp, float humidity, float ppm)
{
 if (temp > 40 && temp <= 70 || ppm > 800 && ppm <= 1000) //cảnh báo cháy
 {
  Turn_On_Warning_Led();
  tone(BUZZER_PIN, 262, 250);
 }

 else if (temp > 70 || ppm > 1000 && ppm <= 1200) //phát sinh cháy
 {
  Turn_On_All_Led();
  tone(BUZZER_PIN, 523);
 }

 else
 {
  Turn_Off_All_Led();
  noTone(BUZZER_PIN);
 }

 Serial.println("Temp: " + String(temp, 2) + "°C");
 Serial.println("Humidity: " + String(humidity, 1) + "%");
 Serial.println("PPM In Air: " + String(ppm, 2) + "ppm");
 Serial.println("--------");
}

void mqttReconnect()
{
 while (!client.connected())
 {
  Serial.println("Attempting MQTT reconnection...");
  if (client.connect("21127147_21127365_21127644"))
  {
    Serial.println("Connected to your device!");
  }
  else
  {
    Serial.println("Try again in 5 seconds...");
    delay(5000);
  }
 }
}

void setup() {
 // put your setup code here, to run once:
 Serial.begin(115200);
 // pinMode(DHT22_PIN, INPUT);
 dhtSensor.setup(DHT22_PIN, DHTesp::DHT22);
 pinMode(LED_PIN_1, OUTPUT);
 pinMode(LED_PIN_2, OUTPUT);
 pinMode(LED_PIN_3, OUTPUT);
 pinMode(LED_PIN_4, OUTPUT);
 pinMode(BUZZER_PIN, OUTPUT);
 pinMode(mq135_simulator, INPUT);

 LCD_init();
 WiFi.begin(ssid, password);
 wifiConnect();
 LCD_show_localIP();

 Serial.println("IP address: ");
 Serial.println(WiFi.localIP());

 client.setServer(mqttServer, port);

}

void loop() {
 // put your main code here, to run repeatedly:
 if (!client.connected())
 {
  mqttReconnect();
 }
 client.loop();

 TempAndHumidity data = dhtSensor.getTempAndHumidity();
 float ppm = Read_ppm_data();
 LCD_handle(data.temperature, data.humidity, ppm);

 //Warning(data.temperature, data.humidity, ppm);

 float temperature = data.temperature;
 float humidity = data.humidity;

 char buffer[50];
 sprintf(buffer, "{\"temperature\": %f, \"humidity\": %f, \"ppm\": %f}", temperature, humidity, ppm);

 client.publish("21127147_21127365_21127644/Data", buffer);
 delay(5000);
}
```

HARDWARE DEVELOPMENT:

1. *Library Inclusions*:

   - The code includes several libraries for working with different sensors and display modules.

2. *Definitions*:

   - Constants and parameters are defined, such as the development board used (ESP-32), voltage resolution, analog pin, sensor type (MQ-135), and more.

3. *Sensor Initialization*:

   - The code initializes and configures the sensors used, including the MQ135 gas sensor, DHT11 temperature and humidity sensor, and the OLED display (SH1106).

4. *Temperature Measurement Task*:

   - The code sets up a task (tempTask) to read temperature values from the DHT11 sensor at regular intervals. This task runs in the background and updates the display with temperature and humidity values.

5. *Calibrating MQ135 Sensor*:

   - The MQ135 gas sensor is calibrated by taking multiple readings and calculating the average. The calibration process helps determine the "clean air" ratio, which is later used to calculate gas concentrations.

6. *Loop Function*:

   - In the main loop, the code first waits for the system to settle down. It then enables the temperature measurement task, updates and displays sensor data, and adds a delay before looping again.

   - There are functions for displaying data on the OLED screen, including showing temperature and humidity values, CO2 and NH4 concentrations.

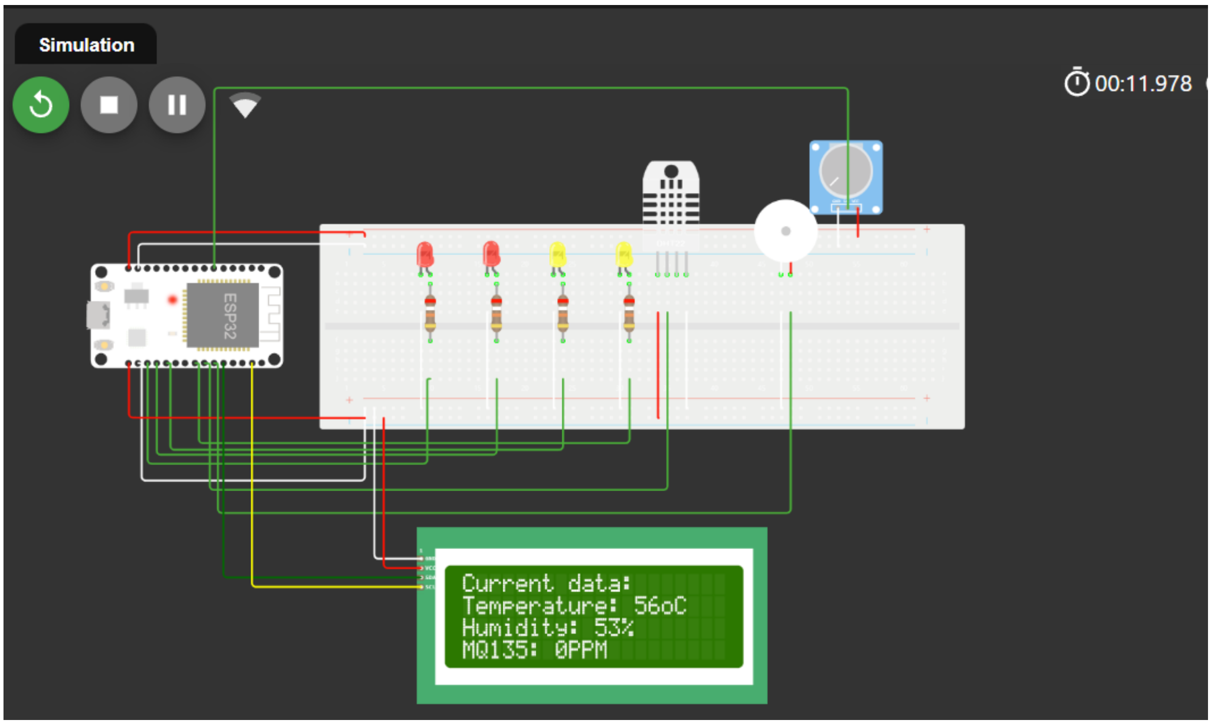
8. *Additional Notes*:

   - The code seems to have an IoT monitoring focus and is designed to periodically update and display sensor data on the OLED screen.

   - The sensors used include the DHT11 for temperature and humidity, and the MQ135 for gas (CO2 and NH4) measurements.

   - The code handles sensor calibration and updates the display with sensor readings.

   - It appears to be designed for the ESP-32 platform.



```
Current data:
Temperature: 56oC
Humidity: 53%
MQ135: 0PPM
```