Problem:

The environment is filled with pollutant and harmful gas due to the modern civilization. Inhaling pollutants for a long time causes damages in human health.

Traditional air quality monitoring systems are expensive. In this project, we are going to make an IoT Based Air Pollution Monitoring System in which we will monitor the Air Quality over a webserver using internet and will trigger an alarm when the air quality goes down beyond a certain level, means when there are sufficient amount of harmful gases are present in the air like CO2, smoke, alcohol, benzene and NH3.

It will show the air quality in PPM on the LCD and as well as on webpage so that we can monitor it very easily.

Design Components :

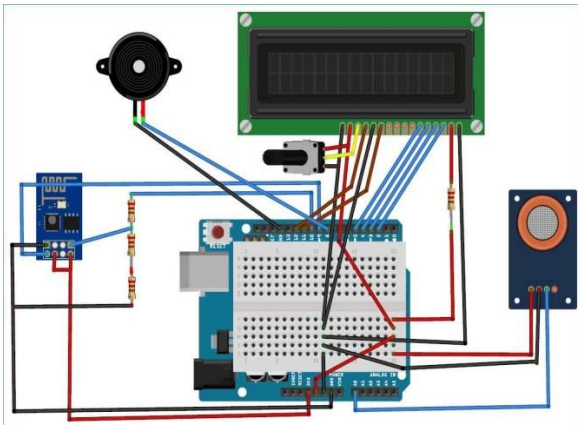
- MQ135 Gas sensor
- Arduino Uno
- Wi-Fi module ESP8266
- 16X2 LCD
- Breadboard
- 10K potentiometer
- 1K ohm resistors
- 220 ohm resistor
- Buzzer

We have used MQ135 sensor as the air quality sensor which is the best choice for monitoring Air Quality as it can detects most harmful gases and can measure their amount accurately. In this IOT project, you can monitor the pollution level from anywhere using your computer or mobile. We can install this system anywhere and can also trigger some device when pollution goes beyond some level, like we can switch on the Exhaust fan or can send alert SMS/mail to the us.

First of all we will connect the **ESP8266 with the Arduino**. ESP8266 runs on 3.3V and if you will give it 5V from the Arduino then it won't work properly and it may get damage. Connect the VCC and the CH_PD to the 3.3V pin of Arduino. The RX pin of ESP8266 works on 3.3V and it will not communicate with the Arduino when we will connect it directly to the Arduino.

So, we will have to make a voltage divider for it which will convert the 5V into 3.3V.

Circuit Diagram:



This can be done by connecting three resistors in series like we did in the circuit. Connect the TX pin of the ESP8266 to the pin 10 of the Arduino and the RX pin of the esp8266 to the pin 9 of Arduino through the resistors.

ESP8266 Wi-Fi module gives your projects **access to Wi-Fi or internet**. It is a very cheap device and make your projects very powerful. It can communicate with any microcontroller .

Then we will connect the **MQ135 sensor with the Arduino**. Connect the VCC and the ground pin of the sensor to the 5V and ground of the Arduino and the Analog pin of sensor to the A0 of the Arduino.

Connect a buzzer to the pin 8 of the Arduino which will start to beep when the condition becomes true.

- Connect pin 1 (VEE) to the ground.
- Connect pin 2 (VDD or VCC) to the 5V.

- Connect pin 3 (V0) to the middle pin of the 10K potentiometer and connect the other two ends of the potentiometer to the VCC and the GND. The potentiometer is used to control the screen contrast of the LCD. Potentiometer of values other than 10K will work too.
- Connect pin 4 (RS) to the pin 12 of the Arduino.
- Connect pin 5 (Read/Write) to the ground of Arduino. This pin is not often used so we will connect it to the ground.
- Connect pin 6 (E) to the pin 11 of the Arduino. The RS and E pin are the control pins which are used to send data and characters.
- The following four pins are data pins which are used to communicate with the Arduino.
- Connect pin 11 (D4) to pin 5 of Arduino.
- Connect pin 12 (D5) to pin 4 of Arduino.
- Connect pin 13 (D6) to pin 3 of Arduino.
- Connect pin 14 (D7) to pin 2 of Arduino.

- Connect pin 15 to the VCC through the 220 ohm resistor. The resistor will be used to set the back light brightness. Larger values will make the back light much more darker.
- Connect pin 16 to the Ground.

Working:

The MQ135 sensor can sense NH3, NOx, alcohol, Benzene, smoke, CO2 and some other gases, so it is perfect gas sensor for our **Air Quality Monitoring Project**. When we will connect it to Arduino then it will sense the gases, and we will get the Pollution level in PPM (parts per million). MQ135 gas sensor gives the output in form of voltage levels and we need to convert it into PPM. So for converting the output in PPM, here we have used a library for MQ135 sensor, it is explained in detail in “Code Explanation” section below.

Sensor was giving us value of 90 when there was no gas near it and the safe level of air quality is 350 PPM and it should not exceed 1000 PPM. When it exceeds the limit of 1000 PPM, then it starts cause Headaches, sleepiness and stagnant, stale, stuffy air and if exceeds beyond 2000 PPM then it can cause increased heart rate and many other diseases.

When the value will be less than 1000 PPM, then the LCD and webpage will display “Fresh Air”. Whenever the value will increase 1000 PPM, then the buzzer will start beeping and the LCD and webpage will display “Poor Air, Open Windows”. If it will increase 2000 then the buzzer will keep beeping and the LCD and webpage will display “Danger! Move to fresh Air”.

Code and its Explanation:

Before beginning the coding for this project, we need to first Calibrate the MQ135 Gas sensor. There are lots of calculations involved in converting the output of sensor into PPM value.

```
#ifndef MQ135_H

#define MQ135_H

#if ARDUINO >= 100

 #include "Arduino.h"

#else

 #include "WProgram.h"

#endif



#define RLOAD 10.0

/// Calibration resistance at atmospheric CO2 level

#define RZERO 76.63

/// Parameters for calculating ppm of CO2 from sensor resistance

#define PARA 116.6020682

#define PARB 2.769034857


/// Parameters to model temperature and humidity dependence

#define CORA 0.00035

#define CORB 0.02718
```

```
#define CORC 1.39538

#define CORD 0.0018


/// Atmospheric CO2 level for calibration purposes

#define ATMOCO2 397.13


class MQ135 {

 private:

  uint8_t _pin;


 public:

  MQ135(uint8_t pin);

  float getCorrectionFactor(float t, float h);

  float getResistance();

  float getCorrectedResistance(float t, float h);

  float getPPM();

  float getCorrectedPPM(float t, float h);

  float getRZero();

  float getCorrectedRZero(float t, float h);

};

#endif
```

To get PPM value directly,

```
}
```

```
void loop MQ135 gasSensor = MQ135(A0);

float air_quality = gasSensor.getPPM();
```

To calibrate the MQ135, let it run for 12 to 24 hours and *RZERO* value.

```
#include "MQ135.h"

void setup (){

Serial.begin (9600);

 () {

MQ135 gasSensor = MQ135(A0); // Attach sensor to pin A0

float rzero = gasSensor.getRZero();

Serial.println (rzero);

delay(1000);

}
```

After getting the *RZERO* value.

Now we can begin the actual code for our Air quality monitoring project.

In the code, first of all we have defined the libraries and the variables for the Gas sensor and the LCD. By using the Software Serial Library, we can make any digital pin as TX and RX pin. In this code, we have made Pin 9 as the RX pin and the pin 10 as the TX pin for the ESP8266. Then we have included the library for the LCD and have defined the pins for the same. We have also defined two more variables: one for the sensor analog pin and other for storing air quality value.

```
#include <SoftwareSerial.h>

#define DEBUG true

SoftwareSerial esp8266(9,10);

#include <LiquidCrystal.h>
```

```
LiquidCrystal lcd(12,11, 5, 4, 3, 2);

const int sensorPin= 0;

int air_quality;
```

Then we will declare the pin 8 as the output pin where we have connected the buzzer. l*cd.begin(16,2)* command will start the LCD to receive data and then we will set the cursor to first line and will print the *'circuitdigest'*. Then we will set the cursor on the second line and will print *'Sensor Warming'*.

```
pinMode(8, OUTPUT);

lcd.begin(16,2);

lcd.setCursor (0,0);

lcd.print ("circuitdigest ");

lcd.setCursor (0,1);

lcd.print ("Sensor Warming ");

delay(1000);
```

Then we will set the baud rate for the serial communication. Different ESP's have different baud rates so write it according to your ESP's baud rate. Then we will send the commands to set the ESP to communicate with the Arduino and show the IP address on the serial monitor.

```
Serial.begin(115200);

esp8266.begin(115200);

  sendData("AT+RST\r\n",2000,DEBUG);

  sendData("AT+CWMODE=2\r\n",1000,DEBUG);

  sendData("AT+CIFSR\r\n",1000,DEBUG);

  sendData("AT+CIPMUair_quality=1\r\n",1000,DEBUG);

  sendData("AT+CIPSERVER=1,80\r\n",1000,DEBUG);

pinMode(sensorPin, INPUT);

lcd.clear();
```

For printing the output on the webpage in web browser, we will have to use **HTML programming**. So, we have created a string named *webpage* and stored the output in it. We are subtracting 48 from the output because the *read()* function returns the ASCII decimal value and the first decimal number which is 0 starts at 48.

```
if(esp8266.available())

 {

  if(esp8266.find("+IPD,"))

  {

   delay(1000);

   int connectionId = esp8266.read()-48;

   String webpage = "<h1>IOT Air Pollution Monitoring System</h1>";

    webpage += "<p><h2>";

    webpage+= " Air Quality is ";

    webpage+= air_quality;

    webpage+=" PPM";

    webpage += "<p>";
```

The following code will call a function named *sendData* and will send the data & message strings to the webpage to show.

```
   sendData(cipSend,1000,DEBUG);

   sendData(webpage,1000,DEBUG);



   cipSend = "AT+CIPSEND=";

   cipSend += connectionId;

   cipSend += ",";

   cipSend +=webpage.length();

   cipSend +="\r\n";
```

The following code will print the data on the LCD. We have applied various conditions for checking air quality, and LCD will print the messages according to conditions and buzzer will also beep if the pollution goes beyond 1000 PPM.

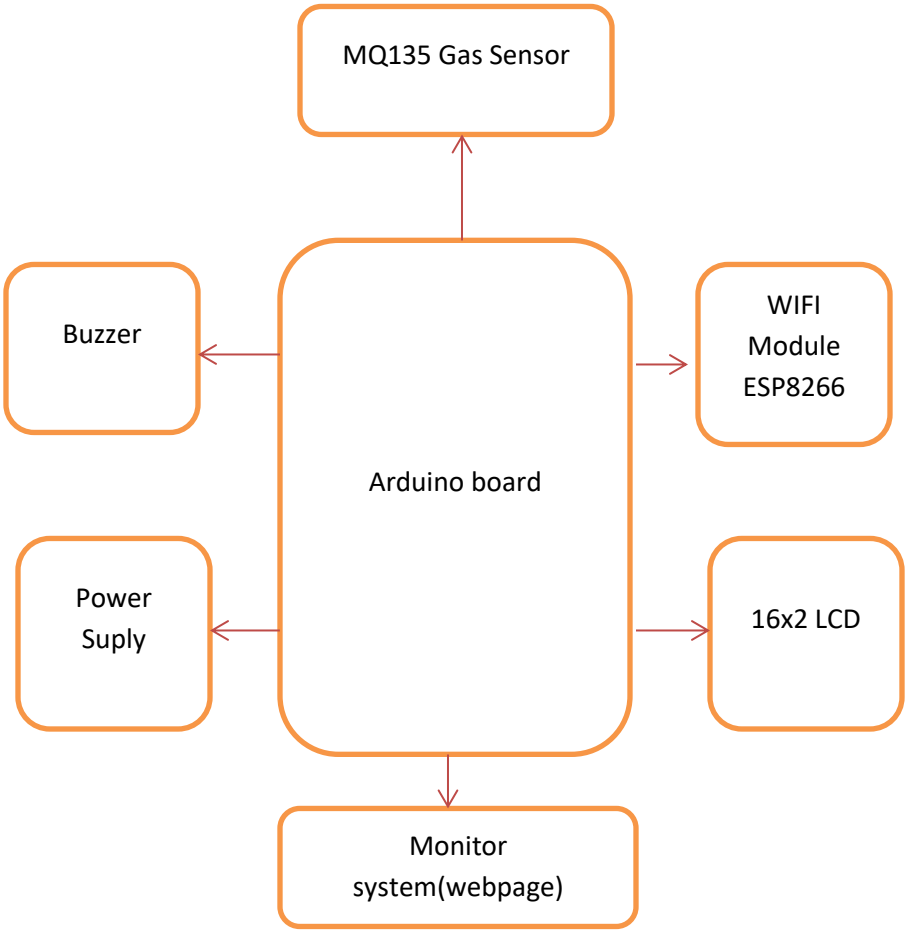
```
lcd.setCursor (0, 0);

lcd.print ("Air Quality is ");

lcd.print (air_quality);

lcd.print (" PPM ");

lcd.setCursor (0,1);

if (air_quality<=1000)

{

lcd.print("Fresh Air");

digitalWrite(8, LOW);
```

Finally the below function will send and show the data on the webpage. The data we stored in string named *'webpage'* will be saved in string named *'command'*. The ESP will then read the character one by one from the *'command'* and will print it on the webpage.

```
String sendData(String command, const int timeout, boolean debug)

{

   String response = "";

   esp8266.print(command); // send the read character to the esp8266

   long int time = millis();

   while( (time+timeout) > millis())

   {

     while(esp8266.available())

     {

       // The esp has data so display its output to the serial window

       char c = esp8266.read(); // read the next character.
```

```
      response+=c;
    }
  }
  if(debug)
  {
   Serial.print(response);
  }
  return response;
}
```

Block Diagram:



Alternative Python Code:

1. boot.py file:- The boot.py file has the code that only needs to run once on boot. This includes importing libraries, network credentials, instantiating pins, connecting to your network, and other configurations. We create our web server using sockets and the Python socket API.

The official documentation imports the socket library as follows:

```
try:
  import usocket as socket
client = MQTTClient("umqtt_client", SERVER)
except:
  import socket
```

We need to import the Pin class from the machine module to be able to interact with the GPIOs.

```
from machine import Pin
```

After importing the socket library, we need to import the network library. The network library allows us to connect the ESP32 or ESP8266 to a Wi-Fi network.

```
import network
```

The MQTT protocol is supported in a built-in library in the Micropython binaries -- this protocol is used to send data from your ESP8266, over WIFI, to a free cloud database.

We used umqtt.simple library , and knowing our SERVER ID, it is possible to create our MQTT client object :

```
from umqttsimple import MQTTClient

SERVER = "mqtt.thingspeak.com"
```

The following lines turn off vendor OS debugging messages:

```
import esp

esp.osdebug(None)
```

Then, we run a garbage collector:

```
import gc

gc.collect()
```

A garbage collector is a form of automatic memory management. This is a way to reclaim memory occupied by objects that are no longer used by the program. This is useful to save space in the flash memory.

The following variables hold network credentials:

```
ssid = '<your_ssid>'

password = '<your_network_password>'
```

Then, setting the ESP32 or ESP8266 as a Wi-Fi station:

```
station = network.WLAN(network.STA_IF)
```

After that, activating the station:

```
station.active(True)
```

Finally, the ESP32/ESP8266 connects to the router using the SSID and password defined earlier:

```
station.connect(ssid, password)
```

The following statement ensures that the code doesn't proceed while the ESP is not connected to the network.

```
while station.isconnected() == False:

  pass
```

After a successful connection, print network interface parameters like the ESP8266 IP address – using the ifconfig() method on the station object.

```
print('Connection successful')
```

```
print(station.ifconfig())
```

`from time import sleep` Create a Pin object called led that is an output, that refers to the ESP8266 GPIO2:

```
led = Pin(2, Pin.OUT)
```

2. main.py file:- The main.py file contains the code that runs the webserver to serve files and perform tasks based on the requests received by the client.

First, we have to import the libraries for modules, sensors (MQ135, DHT) and for IoT Platform Thing-speak (MQTT Client) :

```
from machine import Pin
```

```
import dht,math,time,network,micropython,esp
```

```
from machine import ADC
```

```
from umqttsimple import MQTTClient
```

```
import ubinascii
```

```
from MQ135 import MQ135
```

To initialise the sensors by creating a DHT and MQ135 instance as follows :

```
sensor_dht = dht.DHT11(Pin(5))
```

```
sensor_mq135 = MQ135(0)
```

Then create MQTT client instance and write your channel ID , write APi key from channel created on ThingSpeak. Then create MQTT "Topic"

```
SERVER = "mqtt.thingspeak.com"
```

```
client = MQTTClient("umqtt_client", SERVER)
```

```
CHANNEL_ID = "<your_channel_ID>"
```

Web Development:

```
import sensors  # Import your sensor library
import database  # Import your database library
import analysis  # Import your data analysis library
import visualization  # Import your data visualization library

# Initialize sensors (replace with actual sensor initialization code)
air_quality_sensor = sensors.AirQualitySensor()
temperature_sensor = sensors.TemperatureSensor()
humidity_sensor = sensors.HumiditySensor()

# Main data collection loop
while True:
    # Read sensor data
    air_quality_data = air_quality_sensor.read()
    temperature_data = temperature_sensor.read()
    humidity_data = humidity_sensor.read()

    # Store data in a database (e.g., SQLite, MySQL, InfluxDB)
    database.save_data(air_quality_data, temperature_data, humidity_data)

    # Analyze data for air quality index (AQI)
    aqi = analysis.calculate_aqi(air_quality_data)
```

```
    # Visualize data (e.g., on a dashboard)
    visualization.update_dashboard(air_quality_data, temperature_data, humidity_data, aqi)
```

In this example, we assume the existence of sensor libraries (e.g., "sensors," "database," "analysis," and "visualization") to handle the specific functionalities.

Here's a breakdown of what this code does:

1. Import necessary libraries: Import the libraries that handle sensor data, database operations, data analysis, and data visualization. You would need to create or use appropriate libraries for your specific sensors and requirements.

2. Initialize sensors: Initialize the air quality sensor, temperature sensor, and humidity sensor. Replace this with actual code that initializes the sensors you have.

3. Data collection loop: Continuously read data from the sensors in a loop.

4. Read sensor data: Read air quality, temperature, and humidity data from the respective sensors.

5. Store data in a database: Save the collected data to a database for further analysis and historical records. You would replace "database.save_data" with actual database-specific code.

6. Analyze data for AQI: Calculate the Air Quality Index (AQI) based on the air quality data collected. The "analysis.calculate_aqi" function is a placeholder; you would need to create your own AQI calculation function.

7. Visualize data: Update a dashboard or user interface to display the collected data and the calculated AQI. This visualization can help users understand the air quality in real-time.

Dixd4`