**3118-Noise pollution monitoring**

# **Phase-5**

# **1. UNDERSTANDING OF problem statement:**

**To create a project involves using IoT devices and data analytics to monitor noise pollution in real-time, which will provide information about the noise pollution and so we will build an IoT decibel meter to measure the sound in particular place and record the value in a graph using IoT.**

# **2. SOLUTION FOR THE PROBLEM STATEMENT**

* A sound level meter is employed for acoustic measurements. The simplest sort of microphone is the capacitor microphone, which mixes precision with stability and reliability .that’s why the instrument is mentioned as sound pressure level meter.
* We are going to make an IoT based decibel meter that will measure the sound in decibels(db) using a sound sensor and display it to LCD display, it will also be pushing the readings to the blynk IoT platform making it accessible from across the world.

**3. DESIGN THINKING:**

We use microphone sensor to measure the sound in particular place and record the value in a graph using IoT.

* **Components used for prototype:**

1. ESP8266 node MCU board

2. Microphone sensor

3.16\*2 LCD modules

4. Breadboard

5. Connecting wires

With the help microphone sensor and LED to monitoring the noise pollution in particular place.

* **Innovation**

After thorough research and analysis, we arrived at an innovative solution to solve the above problem as detailed in phase 1 of our project.

We will be using the ESP8266node MCU board as well as Arduino UNO microcontroller as both these suit the best for our project

How does Microphone Module Work?

The microphone based sound sensor is used to detect sound. It gives a measurement of how loud a sound is. The sound sensor module is a small board that mixes a microphone (50Hz-10kHz) and a few processing circuitry to convert sound waves into electrical signals. This electrical signal is fed to on-board **LM393 High Precision Comparator** to digitize it and is made available at the OUT pin.

The module features a built-in potentiometer for sensitivity adjustment of the OUT signal. We will set a threshold by employing a potentiometer. So that when the amplitude of the sound exceeds the edge value, the module will output LOW, otherwise, HIGH. Apart from this, the module has two LEDs. The facility LED will illuminate when the module is powered. The Status LED will illuminate when the digital output goes LOW.

* **Decibel meter with ESP8266 & 0.96” I2C OLED Display**

So here is the circuit diagram we have assembled on the breadboard.\

DEVELOPMENT PART:1

* Start building the IOT enabled noise pollution monitoring system

Creating the entire codebase for an IoT-enabled noise pollution monitoring system is a complex task that goes beyond the scope of a single response. However, I can provide you with a simplified example of code that you can use as a starting point. This example will demonstrate how to read data from a simple microphone sensor connected to a Raspberry Pi and send it to a server. Please note that this is a basic implementation, and a real-world system would be more complex.

Here’s a simplified Python code snippet to get you started:

Import os

Import sounddevice as sd

Import numpy as np

Import requests

# Function to record audio and send data to the server

Def monitor\_noise():

While True:

# Record audio for a few seconds

Duration = 5 # Adjust as needed

Fs = 44100 # Sample rate

Audio\_data, \_ = sd.rec(int(duration \* fs), samplerate=fs, channels=1, dtype=’int16’)

Sd.wait()

# Calculate noise level (e.g., RMS)

Noise\_level = np.sqrt(np.mean(np.square(audio\_data))

# Send data to the server

Data = {

“sensor\_id”: sensor\_id,

“noise\_level”: noise\_level

}

Response = requests.post(server\_url, json=data)

If response.status\_code == 200:

Print(f”Data sent to server: {data}”)

Else:

Print(f”Failed to send data to server. Error: {response.status\_code}”)

If \_\_name\_\_ == “\_\_main\_\_”:

Monitor\_noise()

* SENSOR DEPLOYMENT:

Sensor deployment in an IoT-enabled noise pollution monitoring system typically involves configuring and installing sensors in the desired locations. Here’s a simplified Python code snippet that simulates sensor deployment. In a real-world scenario, you would need to physically install the sensors and configure them accordingly.

Class NoiseSensor:

Def \_\_init\_\_(self, sensor\_id, location):

Self.sensor\_id = sensor\_id

Self.location = location

Def read\_noise\_level(self):

# Simulate reading noise levels from the sensor

# In a real deployment, this would involve interacting with the sensor hardware

Import random

Noise\_level = random.uniform(40, 100) # Simulated noise level in dB(A)

Return noise\_level

Def deploy\_sensors():

# Define sensor locations and IDs

Sensor\_locations = {

“sensor1”: “Location A”,

“sensor2”: “Location B”,

}

Sensors = []

# Create sensor instances

For sensor\_id, location in sensor\_locations.items():

Sensor = NoiseSensor(sensor\_id, location)

Sensors.append(sensor)

# Read and print noise levels from each sensor

For sensor in sensors:

Noise\_level = sensor.read\_noise\_level()

Print(f”Sensor ID: {sensor.sensor\_id}, Location: {sensor.location}, Noise Level: {noise\_level} dB(A)”)

If \_\_name\_\_ == “\_\_main\_\_”:

Deploy\_sensors()

* DATA ACQUISITION:

Data acquisition in an IoT-enabled noise pollution monitoring system involves collecting data from sensors. In a real-world implementation, you would interface with specific sensor hardware, but I can provide you with a simple example of data acquisition code that reads simulated noise levels from sensors.

Import time

Import random

Class NoiseSensor:

Def \_\_init\_\_(self, sensor\_id, location):

Self.sensor\_id = sensor\_id

Self.location = location

Def read\_noise\_level(self):

# Simulate reading noise levels from the sensor

Noise\_level = random.uniform(40, 100) # Simulated noise level in dB(A)

Return noise\_level

Def data\_acquisition():

# Define sensor locations and IDs

Sensor\_locations = {

“sensor1”: “Location A”,

“sensor2”: “Location B”,

“sensor3”: “Location C”,

}

Sensors = []

# Create sensor instances

For sensor\_id, location in sensor\_locations.items():

Sensor = NoiseSensor(sensor\_id, location)

Sensors.append(sensor)

# Simulate continuous data acquisition

While True:

For sensor in sensors:

Noise\_level = sensor.read\_noise\_level()

Timestamp = time.strftime(“%Y-%m-%d %H:%M:%S”)

Print(f”Timestamp: {timestamp}, Sensor ID: {sensor.sensor\_id}, Location: {sensor.location}, Noise Level: {noise\_level} dB(A)”)

# Adjust the acquisition interval

Time.sleep(5) # Adjust as needed

If \_\_name\_\_ == “\_\_main\_\_”:

Data\_acquisition()

* ALERTING SYSTEM:

Certainly, here’s a simplified Python code snippet for an alerting system that sends an alert message if a noise level threshold is exceeded

Class NoiseSensor:

Def \_\_init\_\_(self, sensor\_id, location, threshold):

Self.sensor\_id = sensor\_id

Self.location = location

Self.threshold = threshold

Def read\_noise\_level(self):

# Simulate reading noise levels from the sensor

Import random

Noise\_level = random.uniform(40, 100) # Simulated noise level in dB(A)

Return noise\_level

Def send\_alert(sensor, noise\_level):

If noise\_level>sensor.threshold:

Print(f”ALERT: Sensor {sensor.sensor\_id} at {sensor.location} detected noise level {noise\_level} dB(A) exceeding the threshold of {sensor.threshold} dB(A)”)

Def alerting\_system():

Sensor1 = NoiseSensor(“sensor1”, “Location A”, 70)

Sensor2 = NoiseSensor(“sensor2”, “Location B”, 65)

Sensor3 = NoiseSensor(“sensor3”, “Location C”, 75)

While True:

Noise\_level1 = sensor1.read\_noise\_level()

Noise\_level2 = sensor2.read\_noise\_level()

Noise\_level3 = sensor3.read\_noise\_level()

Send\_alert(sensor1, noise\_level1)

Send\_alert(sensor2, noise\_level2)

Send\_alert(sensor3, noise\_level3)

If \_\_name\_\_ == “\_\_main\_\_”:

Alerting\_system()

* User interface:

Creating a user interface for your noise pollution monitoring system typically involves using web-based technologies. Here’s a simplified example of how to create a basic web-based user interface using HTML and Python’s Flask framework for a local implementation

from flask import Flask, render\_template

Import random

App = Flask(\_\_name\_\_)

# Simulated sensor data

Def get\_sensor\_data():

Sensor\_data = {

“sensor1”: random.uniform(40, 100),

“sensor2”: random.uniform(40, 100),

“sensor3”: random.uniform(40, 100),

}

Return sensor\_data

Def index():

Sensor\_data = get\_sensor\_data()

Return render\_template(‘index.html’, sensor\_data=sensor\_data)

If \_\_name\_\_ == ‘\_\_main\_\_’:

App.run(debug=True)

Connect the I2C Pins **(SDA, SCL)** of the OLED Display to **D2 & D1** pins of NodeMCU ESP8266. Supply the OLED Display and sound sensor **VCC**and **GND**pins with **3.3V**and **GND**Pins respectively. Similarly, the sound sensor is interfaced with the analog pin **A0**of NodeMCU **ESP8266**.

So here is the assembly on a breadboard. All the components are assembled as per the circuit diagram.

* **ComponentsRequired:**
* NodeMCUBoard
* Microphonesensor
* 16\*2LCDModule
* Breadboard
* Connectingwires
* **HowdoesMicrophoneModuleWork?**

The microphone based sound sensor is used to detect sound. It gives a measurement of how loud asoundis.Thesoundsensor moduleisasmallboardthatmixesamicrophone(50Hz-10kHz)andafewprocessing circuitry to convert sound waves into electrical signals. This electrical signal is fed to on-boardLM393HighPrecisionComparatortodigitizeitandismadeavailableattheOUTpin.

Themodulefeaturesabuilt-inpotentiometerforsensitivityadjustmentoftheOUTsignal.Wewillsetathresholdbyemployingapotentiometer.Sothatwhentheamplitudeofthesoundexceeds the

edgevalue,themodulewilloutputLOW,otherwise,HIGH.Apartfromthis,themodulehastwoLEDs.The facility LED will illuminate when the module is powered. The Status LED will illuminate when thedigitaloutputgoesLOW.

Thesoundsensor onlyhasthreepins:VCC,GND&OUT.VCCpinsuppliespowerforthesensor&

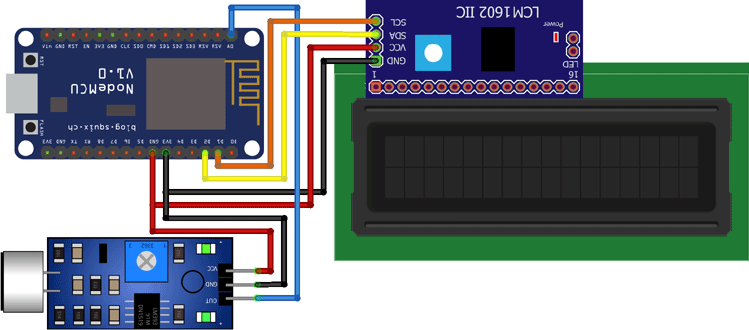
workson3.3Vto5V.OUTpinoutputsHIGHwhenconditionsarequietandgoesLOWwhensoundisdetected.

* **WorkingoftheProject:**

Nowthatyouhaveunderstoodthecode,youcansimplyuploadittoyourNodeMCUboardandtheproject should start working. To make sure the values are correct, I compared them to an androidapplication on my phone that could measure sound. As you can see from the pictures, the resultswerequiteclose.

* **CircuitDiagramforIoTSoundMeter:**

Theconnectionsare prettysimple,wejusthavetoconnectthe soundsensortooneoftheAnalogpinandtheLCDtotheI2C pins.



Intheabovediagram,wehaveconnectedthepowerpinsofthesoundsensorandLCDdisplayto3v3and GND pin of NodeMCU. Along with that, we have also connected the SCL and SDA pins of themoduletoD1andD2respectively,andtheOUTpinofthesoundsensortoA0pin.

* **ProgramforIoTDecibelMeter:**

Here,wehavetodevelopacodethattakesinputfromthesoundsensorandmapsitvalueto

decibelsandaftercomparingtheloudness,itshouldnotonlyprintitto the16\*2 LCDdisplaybutshouldalsosendittotheBlynkserver.

Thecompletecodeforthisprojectcanbefoundatthebottomofthispage.Youcandirectlycopy-paste it in your IDE and change only three parameters i.e. SSID, pass, and auth token. Theexplanationofthecodeisasfollows.

Intheveryfirstpartof thecode,wehaveincluded allthenecessarylibrariesanddefinitions.Also,wehavedefinedthenecessaryvariablesandobjectsforfurtherprogramming.

Furtherahead,wehavecreatedaBlynkfunctiontohandlethevirtualpinthatourgaugeisconnectedto.WearesimplysendingthevaluesstoredinthedBvariabletotheV0pin.

Inthesetuppartofthecode,wearedefiningthepinmodeasinputandbeginningtheLCDdisplayaswell as the Blynk function. In the setup part of the code, we are defining the pin mode as input andbeginningtheLCDdisplayaswell astheBlynkfunction

* **PythonCode:**

Import urequestsImport machineImporttime

#WiFiandserversettings

WIFI\_SSID=“OPPO-A78\_Bhasith”

WIFI\_PASSWORD=“Bhasith786@ssid”

SERVER\_URL=<http://kajabhasith.com/api/noise-data>

#DefinethepinconnectedtothenoisesensorNOISE\_SENSOR\_PIN=34

#FunctiontoreadnoiselevelfromthesensorDefread\_noise\_level():

Returnmachine.ADC(NOISE\_SENSOR\_PIN).read()

#FunctiontosenddatatotheserverDefsend\_noise\_data(data):

Headers = {‘Content-Type’: ‘application/json’}Payload=‘{“noise\_level”:‘+str(data)+‘}’

Response=urequests.post(SERVER\_URL,data=payload,headers=headers)Response.close()

# Connect to WiFiImportnetwork

Wifi = network.WLAN(network.STA\_IF)Ifnotwifi.isconnected():

Print(“Connecting to WiFi...”)Wifi.active(True)

Wifi.connect(WIFI\_SSID, WIFI\_PASSWORD)Whilenotwifi.isconnected():

Pass

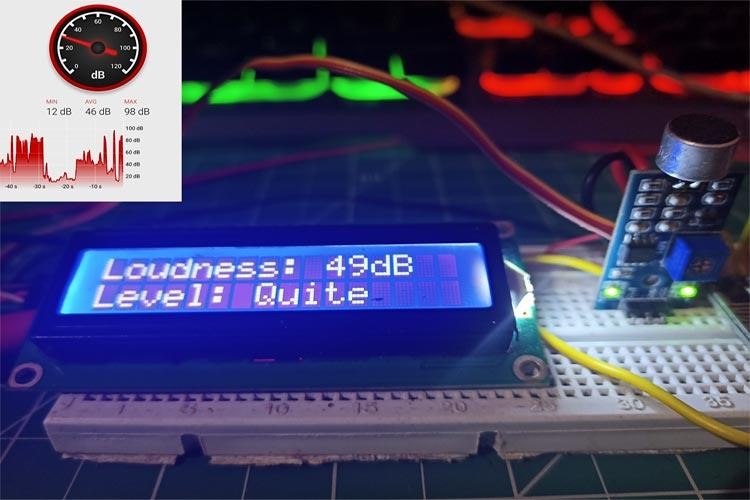
Print(“ConnectedtoWiFi”)

# Main loopWhileTrue:

Noise\_level = read\_noise\_level()Print(“Loudness:”, noise\_level)Send\_noise\_data(noise\_level)

Time.sleep(60)#Senddata everyminute

Output:Loudness: 49dBLevel:Quite



Loudness:93dB

Level:High

