IOT NOISE POLLUTION MONITOR

Group 5 (Electrical/Electronics Engineering)

MEET THE TEAM

APAU, Kwadwo
1845322

HAYIBOR, Irene Eyram Ama
1853522

APPIAH, Andrew Benedict
1845522

AHENE, Emmanuel Adjei
1842822

ANUMEL, Henry Kwesi
1845122

AYAMGA, Peter Atanga
1848722

KWAO NORTEY, Andrews
7134521

APEKE, Ebenezer Elorm
1845422

DANSO, Carl Kwarteng
1850922

AIKINS, Olivia
1843022

PROBLEM STATEMENT

Noise pollution is a growing environmental concern, especially in urban areas, industrial zones, and public spaces. Excessive noise levels can lead to serious health issues, including stress, hearing loss, sleep disturbances, and reduced productivity. Traditional noise monitoring methods are either manual, expensive, or lack real-time tracking and alert mechanisms.

There is a need for an automated, real-time noise pollution monitoring system that can continuously track noise levels and alert authorities or individuals when predefined limits are exceeded. By leveraging IoT technology, this system can provide accurate, remote, and real-time monitoring, helping to enforce noise regulations and promote a healthier environment.

OUR SOLUTION

SIMULATION-BASED IMPLEMENTATION

The project was simulated using Wokwi, an online simulation tool, instead of being physically built. Key components used in the simulation include a potentiometer, a buzzer, an ESP32, and an OLED display.

ESP32

The ESP32's Wi-Fi connectivity enabled real-time noise monitoring by transmitting data to ThingSpeak, allowing remote visualization and analysis.

MONITORING NOISE LEVELS

The OLED display showed real-time noise levels in dB.

A buzzer was programmed to sound whenever the noise level exceeded a predefined threshold.

WHY A POTENTIOMETER?

GPIO34 is an ADC (Analog-to-Digital Converter) pin, allowing the ESP32 to read variable voltage levels from the potentiometer.

The potentiometer's middle pin outputs a variable voltage between 0V and 3.3V, which mimics how a real sound sensor would produce an analogue signal based on noise intensity.

WHY ESP32?

Built-in Wi-Fi for real-time ThingSpeak data transmission.

Multiple ADC pins (12-bit resolution) for accurate noise level readings.

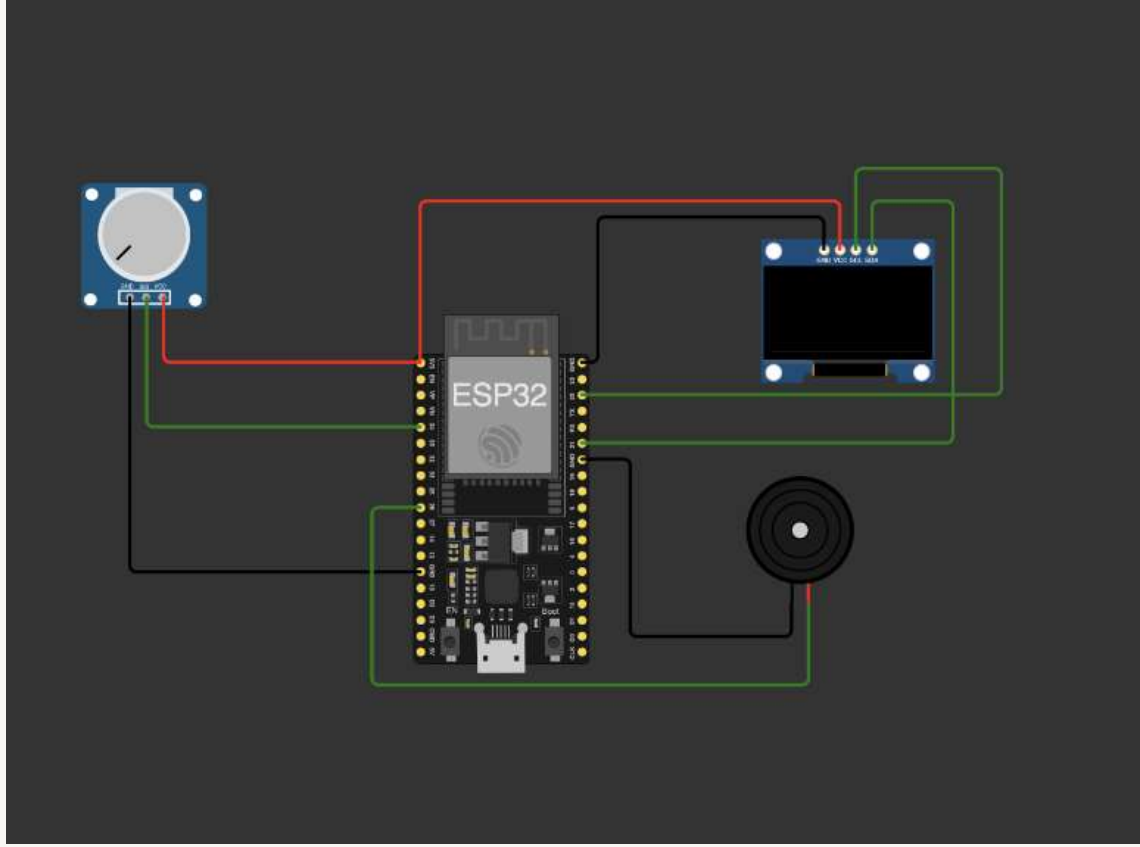
I2C support for seamless OLED display integration.

Low power consumption, ideal for continuous monitoring.

Fast dual-core 240 MHz CPU for handling multiple tasks efficiently.

Better than Arduino Uno (no Wi-Fi) & ESP8266 (fewer ADC pins, lower power).

SCHEMATIC BREAKDOWN



POTENTIOMETER

Middle pin to GPIO34 (ADC Input): Reads voltage changes.

One side to 3.3V: Provides maximum voltage.

Other side to GND: Completes the circuit.

OLED DISPLAY

SDA (GPIO21): Transfers data between ESP32 and OLED.

SCL (GPIO22): Synchronizes data transmission.

VCC (3.3V): Powers the display.

GND: Completes the circuit.

BUZZER

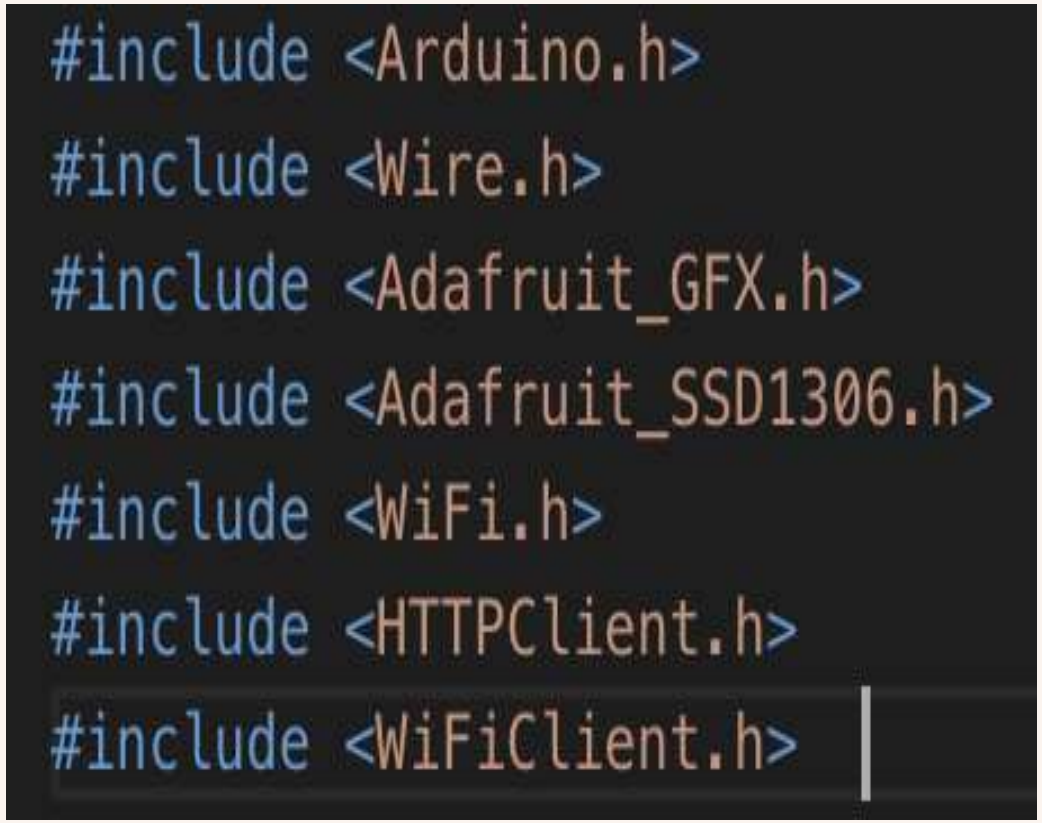
Positive pin (GPIO26): Controls the buzzer.

Negative pin (GND): Completes the circuit.

GPIO26 set to HIGH when noise exceeds the threshold, activating the buzzer.

CODE BREAKDOWN

LIBRARIES AND SUPPORTING FILES



- Includes the core Arduino functions like pinMode(), digitalWrite(), and analogRead().
- Enables I2C communication, used for interfacing with the OLED display.
- A graphics library that provides drawing functions (text, shapes) for the OLED display.
- A library specifically for SSD1306-based OLED displays, allowing easy control of the screen.
- Enables ESP32's Wi-Fi functionality, allowing it to connect to a network and send data.
- Allows the ESP32 to send HTTP requests, needed for transmitting noise data to ThingSpeak.
- Facilitates Wi-Fi-based client-server communication, ensuring stable network connections.

PART II : SETUP

```
void setup() {
    Serial.begin(115200);
    pinMode(SOUND_SENSOR_PIN, INPUT);
    pinMode(BUZZER_PIN, OUTPUT);

    // Initialize OLED Display
    if (!display.begin(SSD1306_SWITCHCAPVCC, 0x3C)) {
        Serial.println("OLED initialization failed");
        while (1);
    }
    display.clearDisplay();
    display.setTextSize(1);
    display.setTextColor(WHITE);

    // Connect to WiFi
    WiFi.begin(ssid, password);
    Serial.print("Connecting to WiFi");
    while (WiFi.status() != WL_CONNECTED) {
        delay(500);
        Serial.print(".");
    }
    Serial.println("Connected!");
}
```

Serial.begin(115200) Enables debugging output in the Serial Monitor. The two lines following it define the capabilities of the buzzer and the potentiometer (sound sensor).

display.begin() → Starts the OLED display using I2C.If initialization fails, print error and stop execution. Clears display, sets text size, and enables white text.

Attempts to connect to Wi-Fi. While trying to connect it prints “.” every 500ms. It will then print "Connected!" once connected to the internet.

PART III : LOOP

Reads analogue voltage from the potentiometer and maps the value (0-4095) to dB range (30-100 dB) using map().

Clears the screen, updates noise level, and displays it.

If noise exceeds the threshold (60 dB) a buzzer activates and should a noise below the threshold be recorded, the buzzer turns off

Checks if Wi-Fi is connected. Sends noise level data to the IoT platform via HTTP request.

Delay (2s) before next reading.

```
void loop() {
    int sensorValue = analogRead(SOUND_SENSOR_PIN);
    float noiseLevel = map(sensorValue, 0, 4095, 30, 100); // Convert

    Serial.print("Noise Level: ");
    Serial.print(noiseLevel);
    Serial.println(" dB");

    // Display on OLED
    display.clearDisplay();
    display.setCursor(0, 10);
    display.print("Noise: ");
    display.print(noiseLevel);
    display.print(" dB");
    display.display();

    // Trigger Alert if Noise Exceeds Threshold
    if (noiseLevel > NOISE_THRESHOLD) {
    Serial.println("Buzzer ON: Noise Too High!");
    tone(BUZZER_PIN, 1000);
    }
    else {
    Serial.println("Buzzer OFF: Noise Normal");
    noTone(BUZZER_PIN);
    }


    // Send Data to IoT Platform
    if (WiFi.status() == WL_CONNECTED) {
        HTTPClient http;
        String url = String(server) + "&field1=" + String(noiseLevel);
        http.begin(url);
        int httpResponseCode = http.GET();
        http.end();
    }

    delay(2000); // Wait before next reading
}
```

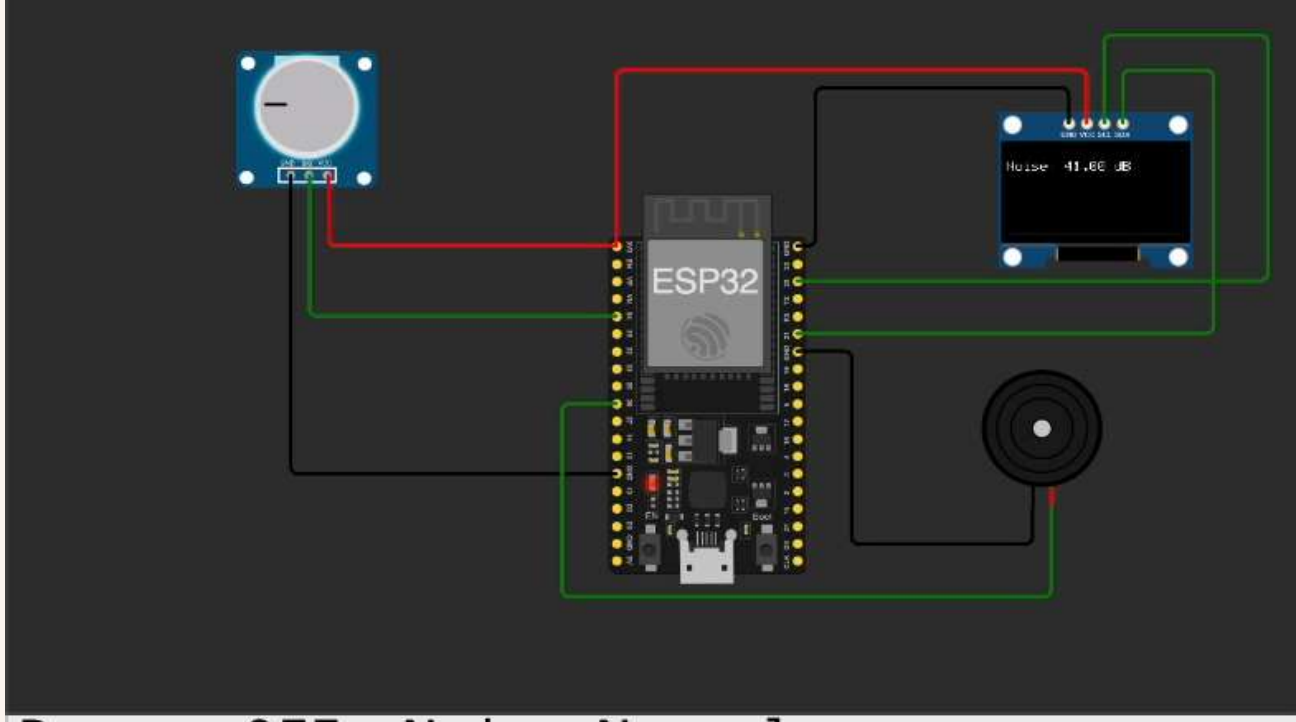
SIMULATION

1. ESP32 Boots Up: Initializes the serial monitor, OLED display, and Wi-Fi connection.
2. Reads Noise Levels: The potentiometer simulates a sound sensor, and the ESP32 converts its voltage to dB.
3. Displays Noise Levels: The OLED screen continuously updates with the noise level in dB.
4. Buzzer Alert: If the noise exceeds 60 dB, the buzzer activates; otherwise, it stays off.
5. Data Transmission: The ESP32 sends noise data to ThingSpeak for remote monitoring.
6. Repeats Every 2 Seconds: The system continuously measures, alerts, and updates data.

Simulation: https://wokwi.com/projects/426250817586979841

Data Visualization:

https://thingspeak.mathworks.com/channels/2889099/private_show




```
Buzzer OFF: Noise Normal
Noise Level: 41.00 dB
Buzzer OFF: Noise Normal
Noise Level: 41.00 dB
Buzzer OFF: Noise Normal
Noise Level: 41.00 dB
Buzzer OFF: Noise Normal
```

KEY TAKEAWAYS

The System Effectively Simulated Noise Level Detection

The OLED Display Provides Clear and Readable Outputs

The Buzzer Alert Works as Expected

The IoT Feature Successfully Sent Data to the Cloud

Possible Improvements for a Real Deployment

As the Voltage level from the potentiometer increases, the noise level increases

CONCLUSION

CHALLENGES FACED

1. Inconsistent Wi-Fi Connectivity: Poor signal strength may cause data transmission failures and cause the system to cease operation.

2. Inaccurate Sensor Readings: Electrical noise or interference may affect analogue readings.

3. Limited Power Supply: Continuous Wi-Fi and sensor operation can drain power quickly.

4. Delayed Data Updates: Network latency can slow down data transmission to ThingSpeak.

5. Fixed Noise Threshold Limitation: A static 60 dB threshold may not be suitable for all environments.

IMPROVEMENTS THAT CAN BE MADE

1. Use a More Reliable Wi-Fi Network structure: Implement reconnection logic in case of disconnections.

2. Replace Potentiometer with Real Sound Sensor: Use an actual microphone module like MAX9814 for better accuracy.

3. Optimize Power Consumption: Implement deep sleep mode when not actively measuring noise.

4. Use Edge Computing: Process noise data locally before sending only significant changes to the cloud.

THANK YOU