IOT_Phase 5
Proj_228488_Team_2
(Noise Pollution Monitoring)

Team Members:

1. Kishore.M (210621106029)
2. Jedance (2106211060027)
3. Nathanial V (210621106036)
4. Edwin Inbaraj (210621106019)

Introduction:

A noise monitoring system is designed to capture, analyze, and manage environmental noise levels. It helps measure and control noise pollution in various settings, including urban areas, industrial sites, and public spaces.

Hardware Components:

First Code Snippet (MicroPython for ESP8266):

- An analog-to-digital converter (ADC) connected to Pin 2, which is used to read analog data.
- The code snippet reads a value from the ADC and prints it.

Second Code Snippet (MicroPython for ESP32):

- The component information is defined in the JSON configuration provided. Here are the components specified in the JSON configuration:
- An ESP32 development kit (type: "wokwi-esp32-devkit-v1") with the ID "esp."This represents the ESP32 microcontroller.
- A microphone component with the ID "mic." This represents a microphone sensor.

Software Components:

- Arduino IDE: The Arduino Integrated Development Environment is used for writing and uploading code to MicroPython for ESP8266 and MicroPython for ESP32.

System Design:

ESP8266/ESP32:

Low-cost microcontrollers with Wi-Fi capabilities, ideal for IoT applications.

IoT Connectivity:

Implementing connectivity to transmit noise data to Azure IoT Hub.

Options include:

Wi-Fi:

Use onboard Wi-Fi capabilities of microcontrollers.

Cellular:

If remote locations are involved, consider using cellular IoT modules.

LoRa:

For long-range communication in low-power scenarios, LoRaWAN can be suitable.

Power Source:

Deployment scenario, power your IoT device with a suitable power source. This could be a standard power outlet, battery, or energy harvesting solutions for low-power, remote deployments.

Traffic Data Integration :

Incorporate traffic data sources such as traffic cameras, GPS, or traffic flow sensors to obtain information about vehicle counts, speeds, and congestion in the same areas.

Data analytics is crucial for noise monitoring as it helps in extracting valuable insights from the collected noise data.

Alert Response:

Plan for appropriate responses to alerts, such as dispatching noise control teams, notifying residents of possible disturbances, or making real-time traffic management adjustments.

Understanding the Problem:

1. **Stakeholder Analysis:**
    - **Citizens:** Suffer from the negative impact of noise pollution.
    - **Local Authorities:** Responsible for enforcing noise regulations.
    - **Urban Planners:** Need data for sustainable city development.
    - **Environmentalists:** Advocate for reduced noise pollution.
2. **Current Limitations:**
    - Lack of real-time noise data.
    - Inefficient noise source identification.
    - Limited community involvement in noise reduction.
3. **Impact Assessment:**
    - Health issues (e.g., stress, sleep disturbance).
    - Reduced property values.
    - Environmental degradation.
    - Legal disputes due to noise complaints.

Design Thinking Approach:

Empathize:

- Conduct surveys and interviews with citizens affected by noise pollution to understand their pain points.
- Analyze existing noise monitoring systems and their shortcomings.
- Engage with local authorities to comprehend their data needs and challenges.

Define:

- Define the project scope, objectives, and success criteria.

- Develop user personas representing citizens, local authorities, urban planners, and environmentalists.
- Establish clear constraints and resources available for the project.

Ideate:

- Brainstorm solutions for real-time noise monitoring.
- Explore sensor technologies (microphones, sound level meters).
- Consider communication protocols (Wi-Fi, LoRa, cellular).
- Ideate on data visualization techniques.
- Think about user-friendly alerting mechanisms.

Prototype:

- Create a prototype of the IoT noise monitoring system.
- Test different noise sensors for accuracy and reliability.
- Develop a basic data transmission mechanism.
- Design a simple data visualization interface for real-time monitoring.

Test:

- Collect test data from sensor prototypes in different urban locations.
- Evaluate the accuracy of noise measurements.
- Gather feedback from potential users (citizens, local authorities).
- Identify issues and areas for improvement in the prototype.

Iterate:

- Based on test results and feedback, refine the prototype.
- Explore advanced features like noise source identification using machine learning.
- Continuously improve the user interface and data visualization.
- Optimize the system for scalability.

Next Steps:

- Proceed with the development of the IoT noise monitoring system based on the refined prototype.
- Implement advanced features like noise source identification.
- Conduct pilot tests in selected urban areas to gather real-world data.
- Collaborate with stakeholders for feedback and further enhancements.

A real-time noise level monitoring system promotes public awareness of noise pollution in several ways:

- It provides real-time data on noise levels in the community. This data can be used to identify areas that are experiencing high levels of noise pollution, and to track changes in noise levels over time. This information can be shared with the public through a variety of channels, such as social media, websites, and mobile apps.
- It can be used to create noise maps. Noise maps are visual representations of noise levels in a community. They can be used to identify noise hotspots and to track the spread of noise pollution. Noise maps can be a powerful tool for raising public awareness of noise pollution and for advocating for noise pollution mitigation measures.
- It can be used to develop and implement noise pollution mitigation strategies. For example, a real-time noise level monitoring system can be used to identify areas that need noise barriers or to enforce noise ordinances. The system can also be used to track the effectiveness of noise pollution mitigation measures.

By raising public awareness of noise pollution, a real-time noise level monitoring system can contribute to noise pollution mitigation in the following ways:

- It can empower individuals to take action to reduce their own exposure to noise pollution. For example, people can use the data from the monitoring system to avoid noisy areas or to take steps to reduce noise levels in their own homes and businesses.
- It can support policies and programs that address noise pollution. For example, public awareness of noise pollution can help to build support for stricter noise ordinances or for investments in noise pollution mitigation measures.

- It can encourage people to be more mindful of the noise they generate. By knowing that their noise levels are being monitored, people may be more likely to take steps to reduce their noise levels.
- Overall, a real-time noise level monitoring system can be a valuable tool for promoting public awareness of noise pollution and contributing to noise pollution mitigation.
- Here are some specific examples of how a real-time noise level monitoring system can be used to promote public awareness and contribute to noise pollution mitigation:
- A city could use a real-time noise level monitoring system to create a public website that shows the current noise levels at different locations throughout the city. Residents could use this information to plan their activities and avoid noisy areas.
- A construction company could use a real-time noise level monitoring system to ensure that their construction activities are not exceeding noise limits. The system could also be used to inform nearby residents of upcoming construction activities and to provide them with contact information in case they have any concerns.
- A school could use a real-time noise level monitoring system to identify areas of the school campus where noise levels are excessive. The school could then take steps to reduce noise levels in those areas, such as installing soundproofing materials or moving noisy activities to less populated areas.
- By using real-time noise level monitoring systems in these and other ways, we can create a more peaceful and healthy environment for everyone.

Visibility of Noise Levels:

Real-time monitoring systems provide up-to-the-minute data on noise levels in specific areas. This data is often displayed on public platforms, websites, or smartphone apps, making it easily accessible to the general public. When people can see the noise levels in their surroundings, they become more aware of the issue.

Alerts and Notifications:

These systems can send alerts or notifications to residents when noise levels exceed certain thresholds, helping individuals understand when noise pollution

is particularly high and prompting them to take action or make lifestyle adjustments.

Community Engagement:

Real-time monitoring systems encourage community engagement. Residents can actively participate in monitoring noise levels by contributing data from their own locations using smartphones or dedicated monitoring devices. This involvement fosters a sense of ownership over the issue.

Education and Information:

When people have access to real-time noise data, they become better informed about the sources of noise pollution in their community. This information often includes details about specific noise sources, their decibel levels, and their potential health and quality of life impacts. This educational aspect raises awareness about the problem and its consequences.

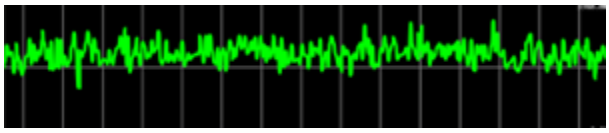
Evidence-Based Advocacy:

Real-time data provides concrete evidence for advocacy efforts. Communities, activists, and local authorities can use this data to support arguments for stricter noise regulations, changes in urban planning, or noise mitigation measures. It empowers residents and organizations to lobby for noise reduction solutions.

Simulation 1 :

```
{
  "version": 1,
  "author": "Kishore M",
  "editor": "wokwi",
  "parts": [
    {
      "type": "wokwi-esp32-devkit-v1",
      "id": "esp",
      "top": -52.9,
      "left": 62.2,
      "attrs": { "env": "micropython-20231005-v1.21.0" }
    },
    { "type": "wokwi-microphone", "id": "mic", "top": -
16.98, "left": 263.79, "attrs": {} }
  ],
  "connections": [
    [ "esp:TX0", "$serialMonitor:RX", "", [] ],
    [ "esp:RX0", "$serialMonitor:TX", "", [] ],
```

```
    [ "mic:1", "esp:D2", "green", [ "v0" ] ],
    [ "mic:2", "esp:GND.1", "green", [ "v0" ] ]
  ],
  "serialMonitor": { "display": "plotter" },
  "dependencies": {}
}



'''
from machine import Pin, ADC
from time import sleep

pot = ADC(Pin(2))
pot.atten(ADC.ATTN_11DB)       #Full range: 3.3v
#ADC.ATTN_0DB: Maximum voltage of 1.2V
#ADC.ATTN_2_5DB: Maximum voltage of 1.5V
#ADC.ATTN_6DB: Maximum voltage of 2.0V
#ADC.ATTN_11DB: Maximum voltage of 3.3V
while True:
  pot_value = pot.read()
  print(pot_value)
  sleep(0.1)
'''
import machine, time
a = machine.ADC(machine.Pin(32))
while True:
    sample = a.read()  # we want 16 bits, a.read() returns
10 bits
    print(sample)
    time.sleep(1/44100)
```



The first snippet is for reading analog input from a potentiometer using an ESP8266 (MicroPython), and the second snippet appears to be for reading analog input from an ADC (Analog-to-Digital Converter) on a MicroPython compatible board.
The second snippet reads analog data using the machine.ADC class and prints the value
Link :

https://wokwi.com/projects/378851393730889729

The conclusion of such a project would typically involve further development, testing, and interaction between various virtual components to achieve a specific goal or test various electronic circuits.

It represents a virtual circuit with an ESP32 and a microphone sensor. It reads analog data from the microphone and prints it to the console.

Simulation Code 2:

```
#include <LiquidCrystal.h> // include the LiquidCrystal library
const int micPin1 = A0; // define the pin for the first microphone
const int micPin2 = A1; // define the pin for the second microphone
const int micPin3 = A2; // define the pin for the third microphone
const int buzzerPin = 9; // define the pin for the buzzer
const int ledPin = 6; // define the pin for the LED
const int contrast = 50; // define the LCD contrast
```

```
LiquidCrystal lcd(12, 11, 5, 4, 3, 2); // initialize the LCD display

void setup() {
  pinMode(buzzerPin, OUTPUT); // set the buzzer pin as output
  pinMode(ledPin, OUTPUT); // set the LED pin as output
  lcd.begin(16, 2); // initialize the LCD display
  analogWrite(6,contrast); // set the LCD contrast
  Serial.begin(9600); // initialize the serial monitor
}

void loop() {
  // read the values from the microphones
  int micValue1 = analogRead(micPin1);
  int micValue2 = analogRead(micPin2);
  int micValue3 = analogRead(micPin3);

  // calculate the sound levels in dB for each microphone
  float voltage1 = micValue1 * 5.0 / 1024.0; // convert the first microphone value to voltage (5V reference)
  float voltage2 = micValue2 * 5.0 / 1024.0; // convert the second microphone value to voltage (5V reference)
  float voltage3 = micValue3 * 5.0 / 1024.0; // convert the third microphone value to voltage (5V reference)
  float dB1 = 20 * log10(voltage1/0.0063); // calculate the sound level in dB for the first microphone
  float dB2 = 20 * log10(voltage2/0.0063); // calculate the sound level in dB for the second microphone
  float dB3 = 20 * log10(voltage3/0.0063); // calculate the sound level in dB for the third microphone

  // calculate the average sound level in dB for all microphones
  float averageDB = (dB1 + dB2 + dB3) / 3;

  // display the sound level on the LCD display and the serial monitor
  lcd.setCursor(0, 0); // set the cursor to the first row of the LCD display
  lcd.print("Sound Level: "); // print the text "Sound Level: " on the LCD display
  lcd.setCursor(0, 1); // set the cursor to the second row of the LCD display
  lcd.print(averageDB); // print the average sound level on the LCD display
  Serial.print("Sound Level: "); // print the text "Sound Level: " on the serial monitor
```

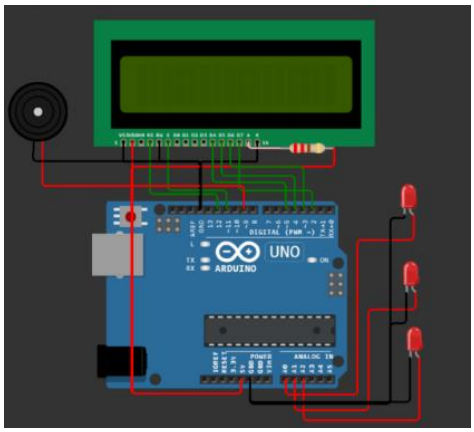
```
  Serial.println(averageDB); // print the average sound level on the serial monitor

  // control the LED and the buzzer based on the sound level
  if (averageDB > 70) { // if the sound level is higher than 70 dB
    digitalWrite(ledPin, HIGH); // turn the LED on
    tone(buzzerPin, 1000, 500); // turn the buzzer on
  } else { // if the sound level is lower than 70 dB
    digitalWrite;
  }
}
```

Diagram.json :

```
{
  "version": 1,
  "author": "Kishore M",
  "editor": "wokwi",
  "parts": [
    { "type": "wokwi-arduino-uno", "id": "uno", "top": 202.2, "left": 18.6, "attrs": {} },
    { "type": "wokwi-lcd1602", "id": "lcd", "top": 8, "left": 20, "attrs": {} },
    { "type": "wokwi-resistor", "id": "r1", "top": 140, "left": 220, "attrs": { "value": "220" } },
    {
      "type": "wokwi-buzzer",
      "id": "bz1",
      "top": 66.16,
      "left": -72.28,
      "attrs": { "volume": "0.1" }
    },
    {
      "type": "wokwi-led",
      "id": "led1",
      "top": 175.61,
      "left": 339.36,
      "attrs": { "color": "red" }
    },
    {
      "type": "wokwi-led",
      "id": "led2",
```

```
    "top": 259.31,
    "left": 341.33,
    "attrs": { "color": "red" }
   },
   {
    "type": "wokwi-led",
    "id": "led3",
    "top": 329.23,
    "left": 345.27,
    "attrs": { "color": "red" }
   }
  ],
  "connections": [
   [ "uno:GND.1", "lcd:VSS", "black", [ "v-51", "*", "h0", "v18" ] ],
   [ "uno:GND.1", "lcd:K", "black", [ "v-51", "*", "h0", "v18" ] ],
   [ "uno:GND.1", "lcd:RW", "black", [ "v-51", "*", "h0", "v18" ] ],
   [ "uno:5V", "lcd:VDD", "red", [ "v16", "h-16" ] ],
   [ "uno:5V", "r1:2", "red", [ "v16", "h-118", "v-244", "h50" ] ],
   [ "r1:1", "lcd:A", "pink", [] ],
   [ "uno:12", "lcd:RS", "green", [ "v-16", "*", "h0", "v20" ] ],
   [ "uno:11", "lcd:E", "green", [ "v-20", "*", "h0", "v20" ] ],
   [ "lcd:D4", "uno:5", "green", [ "v43.53", "h76.86" ] ],
   [ "lcd:D5", "uno:4", "green", [ "v36.63", "h75.24" ] ],
   [ "lcd:D6", "uno:3", "green", [ "v26.79", "h78.54" ] ],
   [ "lcd:D7", "uno:2", "green", [ "v52.39", "h79.87" ] ],
   [ "bz1:2", "uno:9", "red", [ "v36.28", "h220.75" ] ],
   [ "bz1:1", "uno:GND.1", "black", [ "v9.69", "h180.53", "v54.16" ] ],
   [ "led1:A", "uno:A0", "red", [ "v26.92", "h-60.72", "v166.42", "h-77.79" ] ],
   [ "led2:A", "uno:A1", "red", [ "v18.06", "h-50.87", "v108.32", "h-81.73" ] ],
   [ "led3:A", "uno:A2", "red", [ "v67.3", "h-125.71" ] ],
   [ "led3:C", "uno:GND.2", "black", [ "v47.6", "h-169.87" ] ],
   [ "led2:C", "uno:GND.2", "black", [ "v32.84", "h-17.24", "v86.66", "h-
147.71" ] ],
   [ "led1:C", "uno:GND.2", "black", [ "v3.29", "h-15.27", "v198.92", "h-
150.66" ] ]
  ],
  "dependencies": {}
```

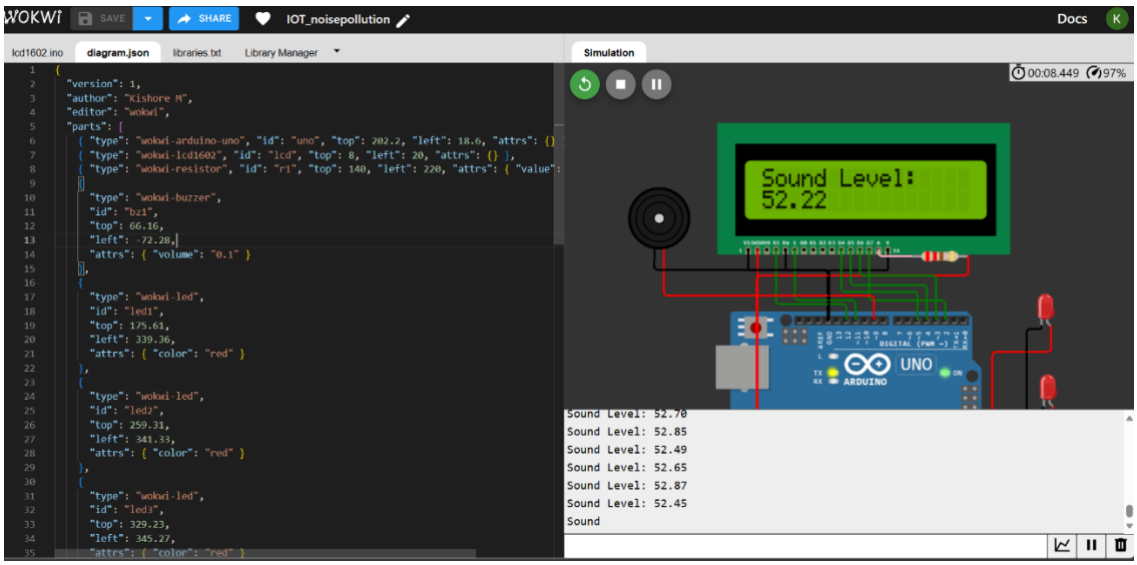
Diagram.json output :

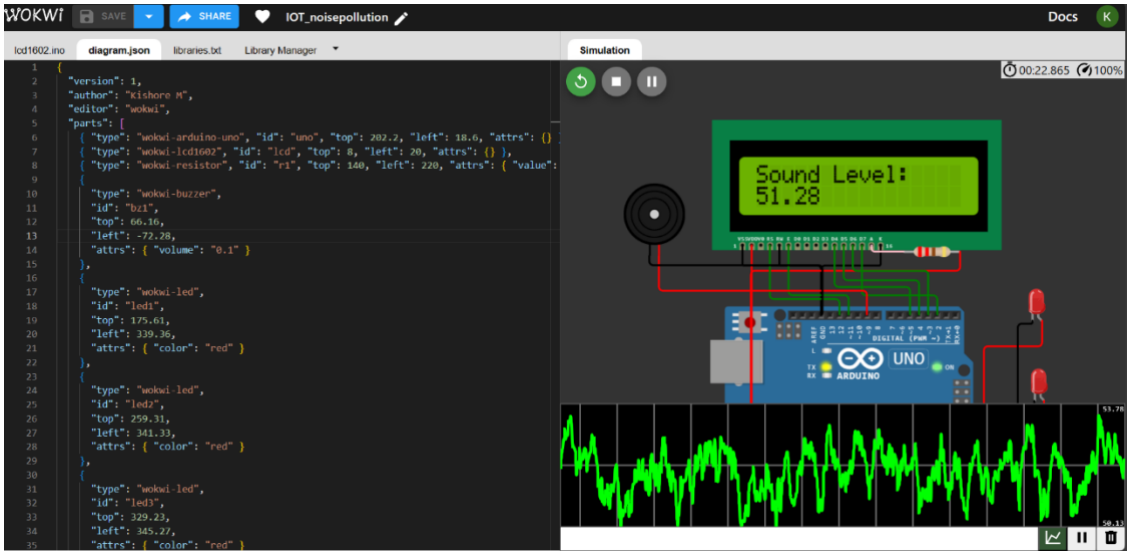


Simulation link:

https://wokwi.com/projects/379572007533323265

Simulation Output:





App for noise monitoring system :

Conclusion:

There are several sources of noise such as automobiles, construction works, tools, and industries. Noise Pollution is becoming a serious concern due to the increase in vehicles, heavy industries, power tools, etc. In conclusion, noise pollution monitoring systems play a pivotal role in modern society by providing critical insights into the levels and sources of noise pollution in our environment. In summary, real-time noise level monitoring systems provide an effective way to engage the public, raise awareness, and contribute to noise pollution mitigation by making noise data readily available, promoting community involvement, and facilitating evidence-based advocacy and solutions.