Phase 4

NOISE POLLUTION MONITORING

INTRODUCTION

Noise pollution is a growing problem in many parts of the world. It can have a negative impact on human health and well-being, and can also disturb wildlife. There are a number of ways to monitor noise pollution, and one of the most effective is to use an ESP32 microcontroller and a sound sensor.

COMPONENTS REQUIRED:

- ESP32 Board
- LCD Display
- Microphone Sensor
- Buzzer
- LED

ESP32

The ESP32 is a low-cost, low-power microcontroller that is well-suited for a variety of IoT applications. It has a built-in Wi-Fi and Bluetooth radio, which makes it easy to connect to the internet and other devices. The ESP32 also has a number of analog and digital pins, which can be used to connect a variety of sensors and actuators.

SOUND SENSOR

A sound sensor is a device that can measure the level of sound in the environment. There are a number of different types of sound sensors available, but one of the most common is the microphone. A microphone converts sound waves into electrical signals, which can then be measured by the ESP32.

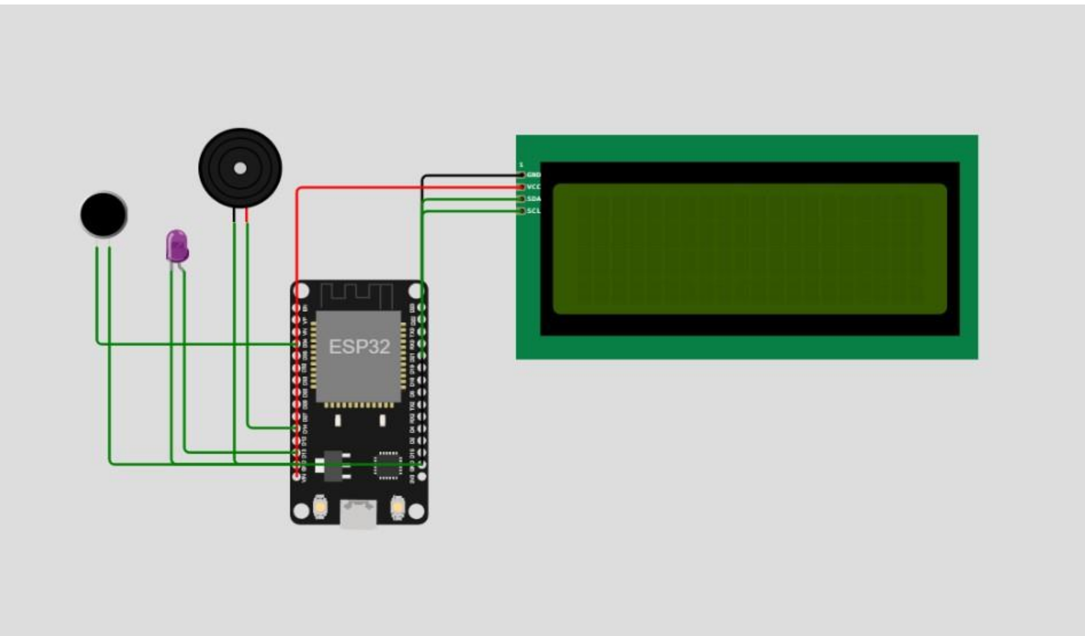
WEB DEVELOPMENT PYTHON

Python is a popular programming language that is well-suited for web development. There are a number of Python frameworks and libraries that can be used to create web applications, such as Django and Flask.

REAL TIME NOISE LEVEL

The ESP32 can be used to measure the real-time noise level in the environment. The sensor data can be sent to a web server using Python, and then displayed on a web page. This allows users to monitor the noise level remotely, and to take action if necessary.

CIRCUIT DIAGRAM:



PYTHON CODE:

```
import time
import math
from machine import ADC, Pin, I2C, PWM
from lcd_api import LcdApi
from pico_i2c_lcd import I2cLcd
import network
import urequests as requests

# Define ADC pin for the microphone
mic_pin = ADC(Pin(34))

# Initialize I2C
i2c = I2C(0, sda=Pin(21), scl=Pin(22), freq=400000)

# Initialize LCD display with your specific settings
I2C_ADDR = 39
I2C_ROWS = 4
I2C_COLS = 20
lcd = I2cLcd(i2c, I2C_ADDR, I2C_ROWS, I2C_COLS)

# Define the buzzer and LED pins
buzzer_pin = Pin(14)
led_pin = Pin(13, Pin.OUT)

# Define the noise threshold in dB
noise_threshold = 60  # Adjusted to 60 dB
```

```
# Microphone sensitivity (in dB per Volt) - replace with your microphone's sensitivity
MIC_SENSITIVITY = 3.0

# Define your Wi-Fi credentials
WIFI_SSID = "Wokwi-GUEST"
WIFI_PASS = ""

# Initialize Wi-Fi
wifi = network.WLAN(network.STA_IF)
wifi.active(True)
wifi.connect(WIFI_SSID, WIFI_PASS)

# Wait until Wi-Fi connection is established
while not wifi.isconnected():
    pass

# Create a PWM object for the buzzer
buzzer_pwm = PWM(buzzer_pin)
buzzer_pwm.deinit()  # Turn off the buzzer at the beginning

# Define a function to calculate sound pressure level (SPL)
def calculate_noise_level(adc_value, reference_voltage, sensitivity):
    # Calculate noise level in decibels (dB) based on sensitivity and voltage
    noise_db = 20 * math.log10(adc_value / (reference_voltage * sensitivity))
    return noise_db
```

```
# Define a function to update noise level and control the buzzer and LED
def update_noise_level():
    global noise_level

    # Read ADC values from the microphone
    mic_value = mic_pin.read()

    # Set a constant reference voltage based on your system's maximum voltage
    reference_voltage = 5.0  # Assuming 5V maximum reference voltage

    # Calculate noise level
    noise_level = calculate_noise_level(mic_value, reference_voltage, MIC_SENSITIVITY)  # Update noise_level directly

    # Control the buzzer and LED based on the noise level
    if noise_level > noise_threshold:
        # Turn on the buzzer and LED
        buzzer_pwm.freq(1000)
        led_pin.on()
    else:
        # Turn off the buzzer and LED
        buzzer_pwm.deinit()
        led_pin.off()

# Define a function to display noise level status
def display_noise_status():
    if noise_level < noise_threshold - 10:
        status = "Quiet"
```

```
    elif noise_level >= noise_threshold - 10 and noise_level < noise_threshold + 10:
        status = "Normal"
    else:
        status = "High"

    # Display noise level status and the actual noise level on the LCD and serial monitor
    lcd.clear()
    lcd.move_to(0,0)
    lcd.putstr("Loudness: {:.2f}dB".format(noise_level))
    lcd.move_to(0,2)
    lcd.putstr("Level: {}".format(status))

    print("Loudness: {:.2f} dB".format(noise_level))
    print("Level: {}".format(status))

# Main loop with continuous data transmission
first_data_sent = False  # Track if the first data is sent

while True:
    # Update noise level
    update_noise_level()

    # Display noise level status
    display_noise_status()

    # Send data to your server continuously after the first data
```

```
    if first_data_sent:
        data = {
            "noise_level": noise_level
        }
        response = requests.post(SERVER_URL, json=data)

    if not first_data_sent:
        first_data_sent = True
# Mark the first data as sent
```

OUTPUT OF PROGRAM

The output of a noise pollution monitoring project using an ESP32 can be in a variety of formats, depending on the specific needs of the project. Some common output formats include:

- Real-time noise level data: This is the most common type of output, and it consists of a continuous stream of noise level data sent to a server or other device. The data can be in any format, but it is typically sent in JSON or CSV format.
- Noise level alerts: This type of output is generated when the noise level exceeds a certain threshold. The alert can be sent to a variety of devices, such as a smartphone, email, or SMS.
- Noise level reports: This type of output is generated over a period of time, such as a day, week, or month. The report can include statistics such as the average noise level, the maximum noise level, and the number of noise level alerts that were generated.

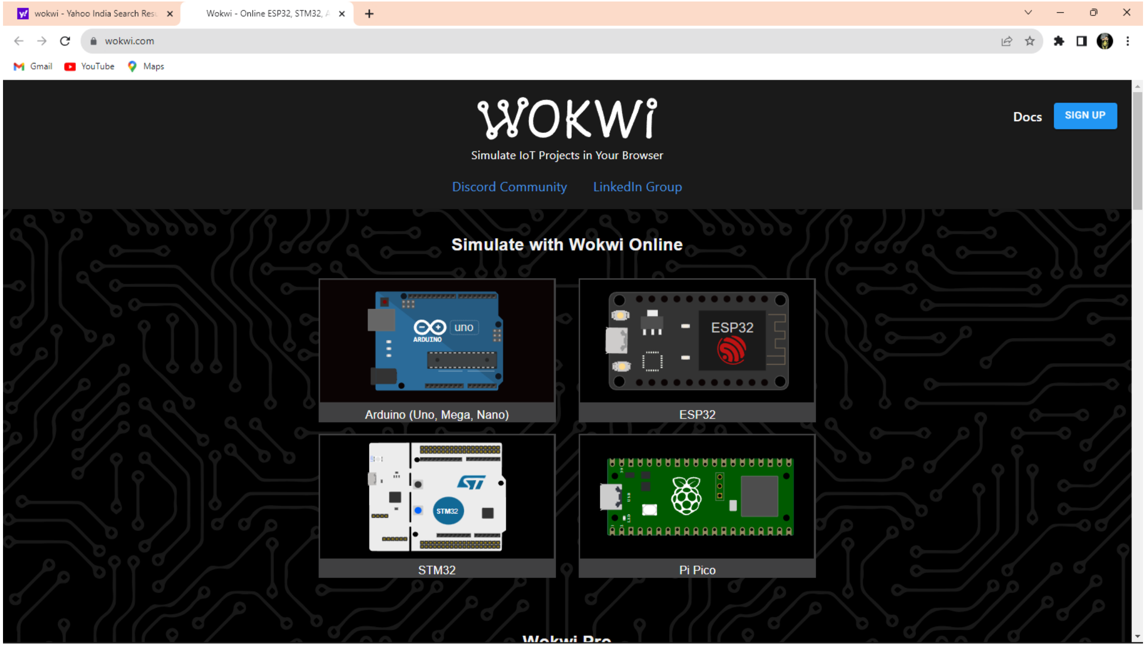
The specific output format that you choose will depend on the needs of your project and the devices that you are using. For example, if you are building a noise pollution monitoring system for a city, you might want to send the real-time noise level data to a server that can be used to generate a noise level map. If you are building a noise pollution monitoring system for a home, you might want to receive noise level alerts on your smartphone.

WOKWI SIMULATION PLATFORM:

An IoT simulation platform is a software tool that allows you to create and test IoT devices and systems without having to use physical hardware. This can be useful for a variety of reasons, such as:

- Testing new IoT devices and systems: You can use an IoT simulation platform to test new IoT devices and systems before you deploy them in the real world. This can help you to identify and fix any problems before they cause any disruptions.
- Prototyping IoT solutions: An IoT simulation platform can be used to prototype IoT solutions before you develop the actual hardware and software. This can help you to validate your ideas and to get feedback from users early on.
- Training and education: An IoT simulation platform can be used to train and educate people about IoT devices and systems. This can be useful for students, engineers, and other professionals who need to learn about IoT.

IoT simulation platforms typically work by creating a virtual representation of the IoT device or system that you want to test. This virtual representation can then be used to simulate the behaviour of the device or system in the real world.



WEBSITE LINK:

Web: https://wokwi.com/

CONCLUSION

Noise pollution monitoring is an important issue, and the ESP32 microcontroller is a powerful tool that can be used to monitor noise levels in real time. By using an ESP32 and a sound sensor, you can create a noise pollution monitoring system that can be used to improve the quality of life in your community.