NOISE POLLUTION MONITORING PHASE\_3

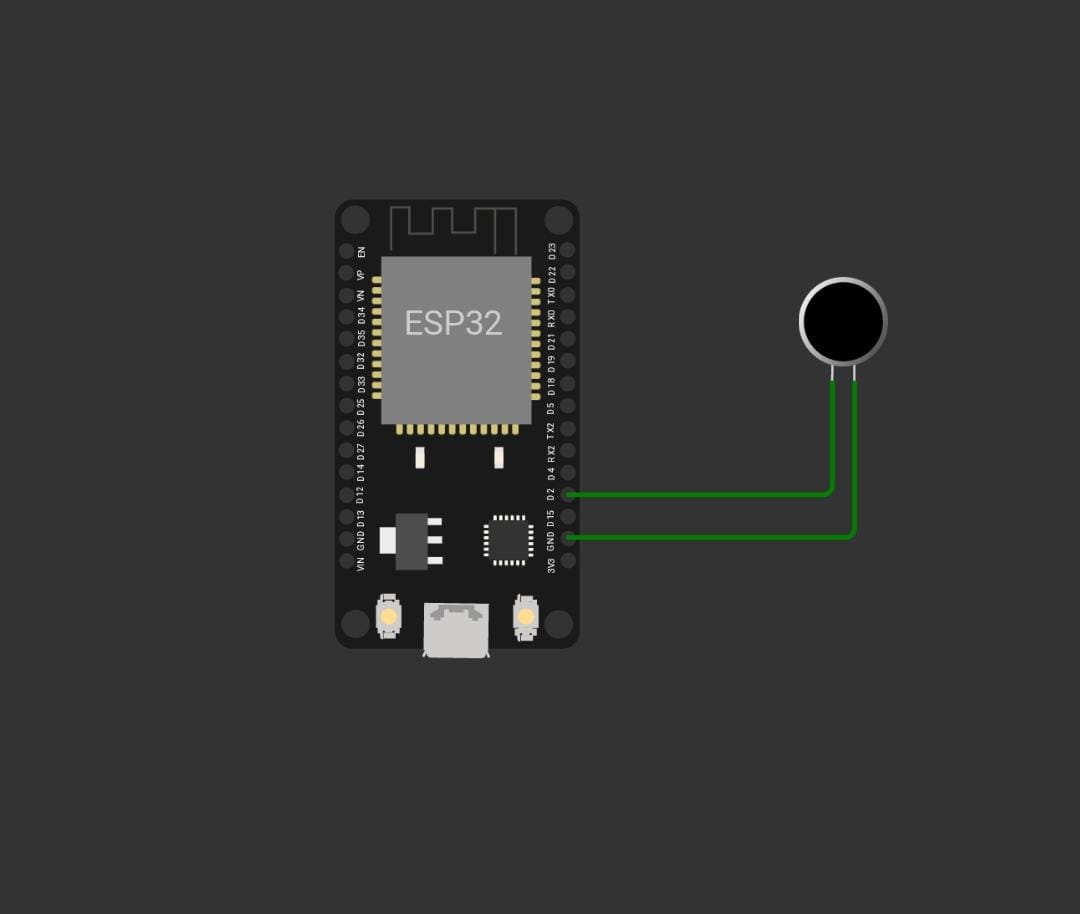
PHASE\_3 Development part 1

Noise pollution is a prevalent issue in urban environments, and with the help of Arduino and Python, we can develop a cost-effective solution to monitor noise levels and analyze noise pollution statistics.We'll utilize the Arduino Uno for data collection as it offers an easy interface for various sensors.To collect noise data, we'll integrate a sound sensor (e.g., KY-038) with the Arduino Uno.The Arduino Uno will be responsible for collecting noise data through the sound sensor.

Programming

In the code, given below we are using MicroPython to read analog input from a pin using the machine.ADC module and printing the readings to the console.

the code is written in MicroPython, a version of Python designed for microcontroller and embedded systems programming.



CODE

'''

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

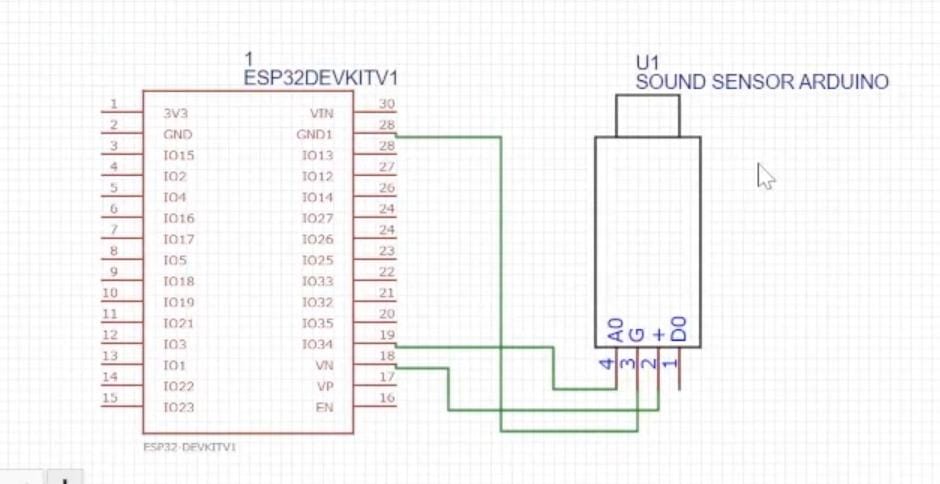
a = machine.ADC(machine.Pin(32))

while True:

sample = a.read() # we want 16 bits, a.read() returns 10 bits

print(sample)

time.sleep(1/44100)



Explanation of the above code

Code Snippet 1:

from machine import Pin, ADC

from time import sleep

pot = ADC(Pin(2))

pot.atten(ADC.ATTN\_11DB) # Full range: 3.3v

from machine import Pin, ADC: This line imports the Pin and ADC classes from the machine module. These classes are typically used in MicroPython to work with GPIO pins and analog-to-digital conversion.

from time import sleep: This line imports the sleep function from the time module, which is used to introduce delays in the program.

pot = ADC(Pin(2)): This line creates an ADC (Analog-to-Digital Converter) object called pot by initializing it with Pin 2. In MicroPython, Pin 2 is used as an input for analog measurements.

pot.atten(ADC.ATTN\_11DB): This line sets the attenuation level of the ADC to ADC.ATTN\_11DB, which corresponds to the maximum voltage range of 3.3 volts. This adjustment ensures that the ADC can handle the full voltage range.

Comments are included to explain the different attenuation levels (ADC.ATTN\_0DB, ADC.ATTN\_2\_5DB, ADC.ATTN\_6DB, and ADC.ATTN\_11DB) and their associated maximum voltage ranges.

while True:

pot\_value = pot.read()

print(pot\_value)

sleep(0.1)

while True:: This starts an infinite loop, which means the following code will keep executing indefinitely.

pot\_value = pot.read(): This line reads the analog voltage from the pot ADC object and stores it in the variable pot\_value. The value represents the analog voltage level at Pin 2.

print(pot\_value): This line prints the pot\_value to the console, showing the analog voltage value.

sleep(0.1): This introduces a 100 ms (0.1-second) delay, causing the program to pause briefly before repeating the loop. This effectively sets the sampling rate.

Code Snippet 2:

import machine, time

a = machine.ADC(machine.Pin(32))

import machine, time: This line imports the machine module for working with hardware, and the time module for handling time-related functions.

a = machine.ADC(machine.Pin(32)): This line creates an ADC object called a by initializing it with Pin 32. In this snippet, the ADC is used to read analog data from Pin 32.

while True:

sample = a.read() # we want 16 bits, a.read() returns 10 bits

print(sample)

time.sleep(1/44100)

while True:: This initiates an infinite loop, indicating that the following code will run continuously.

sample = a.read(): This line reads an analog value from the a ADC object and stores it in the variable sample. The comment suggests that you want a 16-bit value, but the a.read() function, by default, returns a 10-bit value.

print(sample): This line prints the sample value to the console.

time.sleep(1/44100): This introduces a delay of 1/44100 seconds, which appears to be intended to emulate a sampling rate of 44.1 kHz, which is commonly used in audio applications. The time.sleep function is used to control the sampling rate.

Output link

<https://wokwi.com/projects/378856648020956161>

