Experiment title – Time of flight measurements using an ultrasonic sensor

Week no. – 3

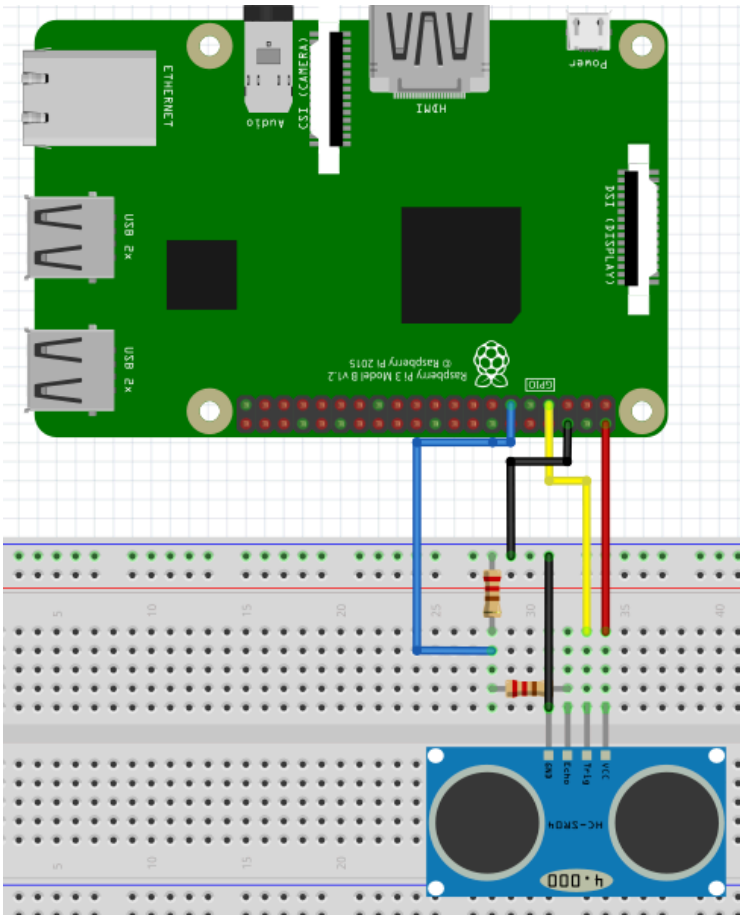
Date of experiment – 05/09/2023

Group name – Group 6

Group members – Binduthraya Matta (B21203), Saransh Duharia (B21021), Jasnoor Tiwana (B21194), Rawal Ram(B21219), Adarsh Santoria (B21176).

---

1. Experiment layout



Software Used Fritzing

2. Results (in order of the indicated task completion criteria)

Task 4(LAB 2) – Start the daemon with a sample rate of 4 µs. .Complete the code by adding the missing snippets. Which is the maximal frequency you can set? How many possible levels for duty cycle and frequency are there?Repeat task 1 and mark your observations.

Command executed to initiate the pigpiod daemon with a sample rate of 4 µs.-

```
sudo killall pigpiod
sudo pigpiod -s 4 -b 200 -f
```

The number of maximal frequencies we can set are- **18**

Maximum frequency - **10000 Hz**

Minimum frequency - **13 Hz**

```
                    Hertz

      1: 40000 20000 10000 8000 5000 4000 2500 2000 1600
          1250  1000   800  500  400  250  200  100   50

      2: 20000 10000  5000 4000 2500 2000 1250 1000  800
           625   500   400  250  200  125  100   50   25

      4: 10000  5000  2500 2000 1250 1000  625  500  400
           313   250   200  125  100   63   50   25   13
sample
 rate
 (us) 5:  8000  4000  2000 1600 1000  800  500  400  320
           250   200   160  100   80   50   40   20   10

      8:  5000  2500  1250 1000  625  500  313  250  200
           156   125   100   63   50   31   25   13    6

     10:  4000  2000  1000  800  500  400  250  200  160
           125   100    80   50   40   25   20   10    5
```

Possible Levels for Duty Cycle:
By default the range of Duty Cycle is from 0-255

```
pi.set_PWM_dutycycle(4,   0) # PWM off
pi.set_PWM_dutycycle(4,  64) # PWM 1/4 on
pi.set_PWM_dutycycle(4, 128) # PWM 1/2 on
pi.set_PWM_dutycycle(4, 192) # PWM 3/4 on
pi.set_PWM_dutycycle(4, 255) # PWM full on
```

But the default range can be changed using "*set_PWM_range*"

pi.set_PWM_range(userGPIO, range)

```
pi.set_PWM_range(9, 100)  # now  25 1/4,   50 1/2,   75 3/4 on
pi.set_PWM_range(9, 500)  # now 125 1/4,  250 1/2,  375 3/4 on
pi.set_PWM_range(9, 3000) # now 750 1/4, 1500 1/2, 2250 3/4 on
```

Completed Code Snippet

```
#Distance measurement using HC-SR04 using software triggered pulses.
import RPi.GPIO as GPIO
import time

#Preparing the RPi-----------------------------------------
#GPIO Mode (BOARD / BCM)
GPIO.setmode(GPIO.BCM)

#set GPIO Pins
TRIG = 4
ECHO = 17

#set GPIO direction (IN / OUT)
GPIO.setup(TRIG, GPIO.OUT)
GPIO.setup(ECHO, GPIO.IN)

#Create single trigger pulse with 10 ms on the TRIG pin
#Send trigger signal
GPIO.output(TRIG, GPIO.HIGH)
time.sleep(10E-6)
GPIO.output(TRIG, GPIO.LOW)
#-------------------------------------------------------------

# save StartTime
while GPIO.input(ECHO) == 0:
    StartTime = time.time()
```

```
# save time of arrival

while GPIO.input(ECHO) == 1:

    StopTime = time.time()


TimeElapsed = (StopTime - StartTime)

print(TimeElapsed)


GPIO.cleanup()
```

Software used: Visual Studio Code

Task 1(LAB 3) – Connect the sensor according to the figure. Why are the resistors required? What are the resistor values that you can use?

The Raspberry Pi GPIO pins operate at 3.3V, but the HC-SR04 normally operates at 5V logic levels. We need voltage dividers to step down the 5V signals to 3.3V in order to ensure secure and dependable communication between the HC-SR04 and the Raspberry Pi.

Resistors can offer defense against improper connections or short circuits. They help to minimize harm in the event of incorrect wiring by limiting the current that passes between the HC-SR04 and the Raspberry Pi GPIO pins.

Any combination of resistors is possible which can divide the voltage and get it to the level of 3.3V or less and also provide enough power to drive the sensor's reading.

example: we can use a 2.2kΩ resistor in series with the echo pin and a 4.7kΩ resistor in parallel with the echo pin to ground.

Task 2(LAB 3) – Place an object 10 cm in front of the sensor.

a. Connect the Trigger pin to channel 1 (GPIO4 of Raspberry Pi) and the Echo pin to channel 2 (GPIO17 of Raspberry Pi through a resistor) the oscilloscope.

In this step, the ultrasonic sensor's Trigger pin is connected to channel 1 of the oscilloscope, and the Echo pin is connected to channel 2. This setup allows us to monitor the signals generated by the sensor.

b. Send a trigger signal of 10 ms (using RPi.GPIO library) to the sensor.

Using the RPi.GPIO library, a trigger signal of 10 milliseconds is sent to the ultrasonic sensor. This signal initiates the process where the sensor emits an ultrasonic wave.

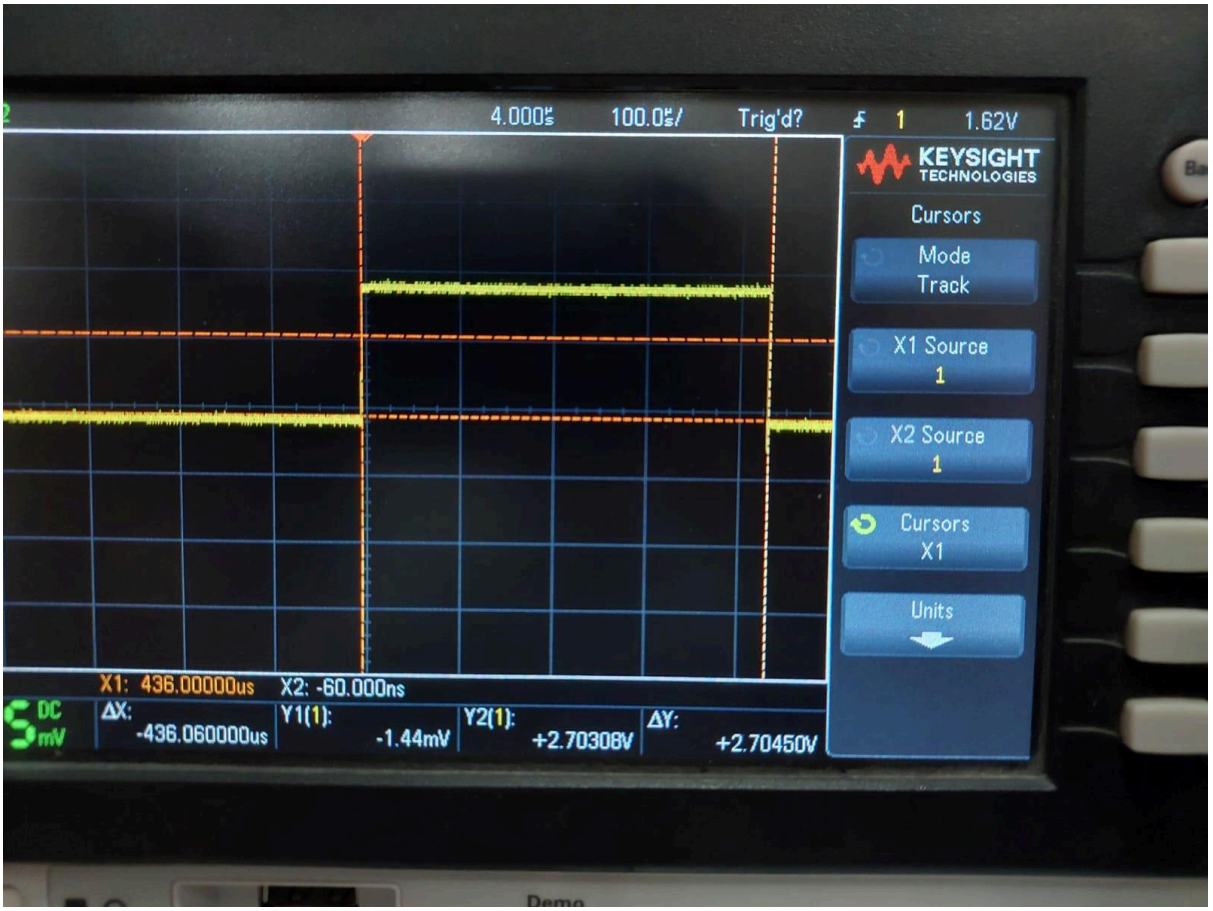
c. Adjust the oscilloscope such that you can see the signals on both channels and set the persistence mode. Acquire a single shot signal.

To visualize the signals on both channels (Trigger and Echo), the oscilloscope is adjusted. Persistence mode is set to retain the signal traces for better observation. A single-shot signal is acquired, which captures the entire process of the ultrasonic wave emission and reception.

d. Change the distance of the object to 20 cm and repeat the acquisition.

The initial distance of the object from the sensor is changed to 20 cm. This modification allows us to observe how the time delay between the trigger and echo signals changes with the distance.

e. Explain your observations for both channels. For your report, save the .csv files and plot the curves and explain how the time-of-flight signal is encoded in your observations.



At a distance of 7.49cm causing Time difference 436µs

Observations for Channel 1 (Trigger): Channel 1 displays a trigger signal with a sharp rising edge, indicating the moment the sensor sends the ultrasonic wave. The trigger signal remains consistent in terms of shape and timing.

Observations for Channel 2 (Echo): Channel 2 shows the echo signal, which begins with a rising edge after the trigger signal is sent. The time delay between the rising edge of the trigger signal and the rising edge of the echo signal represents the time it takes for the ultrasonic wave to travel to the object and back to the sensor. As the distance between the sensor and the object increases (from 10 cm to 20 cm), the time delay also increases. This is evident from the shift in the position of the rising edge of the echo signal. Time-of-Flight Signal Encoding: The time delay observed in Channel 2 is directly proportional to the distance between the sensor and the object.

Task 3(Lab 3) – Use the pigpio library to measure the time of flight (Measurement of the Echo Pin) using the Raspberry Pi using the callback functions. Download the code "lab3_3_software".

a. Download the code and explain the logic. How is the time-of-flight of the sound wave measured? Which pin is used for the measurement? How are the timing events triggered?

Logic of the Code: The provided code uses the pigpio library to measure the time-of-flight of an ultrasonic signal. It utilizes callback functions to handle timing events efficiently.
Measurement of Time-of-Flight: Time-of-flight is measured by calculating the time delay between sending a trigger signal to the ultrasonic sensor and receiving the echo signal when it bounces back from an object. The code uses the difference between the rising edge and falling edge timestamps of the Echo pin to calculate this time delay.

Used Pin for Measurement: The Echo pin of the ultrasonic sensor is used for measurement. When the sensor receives the echo signal after the ultrasonic wave hits an object, the Echo pin goes high, and this signal is used to trigger the timing events.

Triggering Timing Events: Timing events are triggered using callback functions. When the Echo pin experiences a rising edge (it goes from low to high), a callback function records the timestamp of the rising edge, marking the time when the ultrasonic wave was sent. When the Echo pin experiences a falling edge (it goes from high to low), another callback function records the timestamp of the falling edge, marking the time when the echo signal was received. The time difference between these two timestamps represents the time-of-flight, which is then used to calculate the distance to the object.

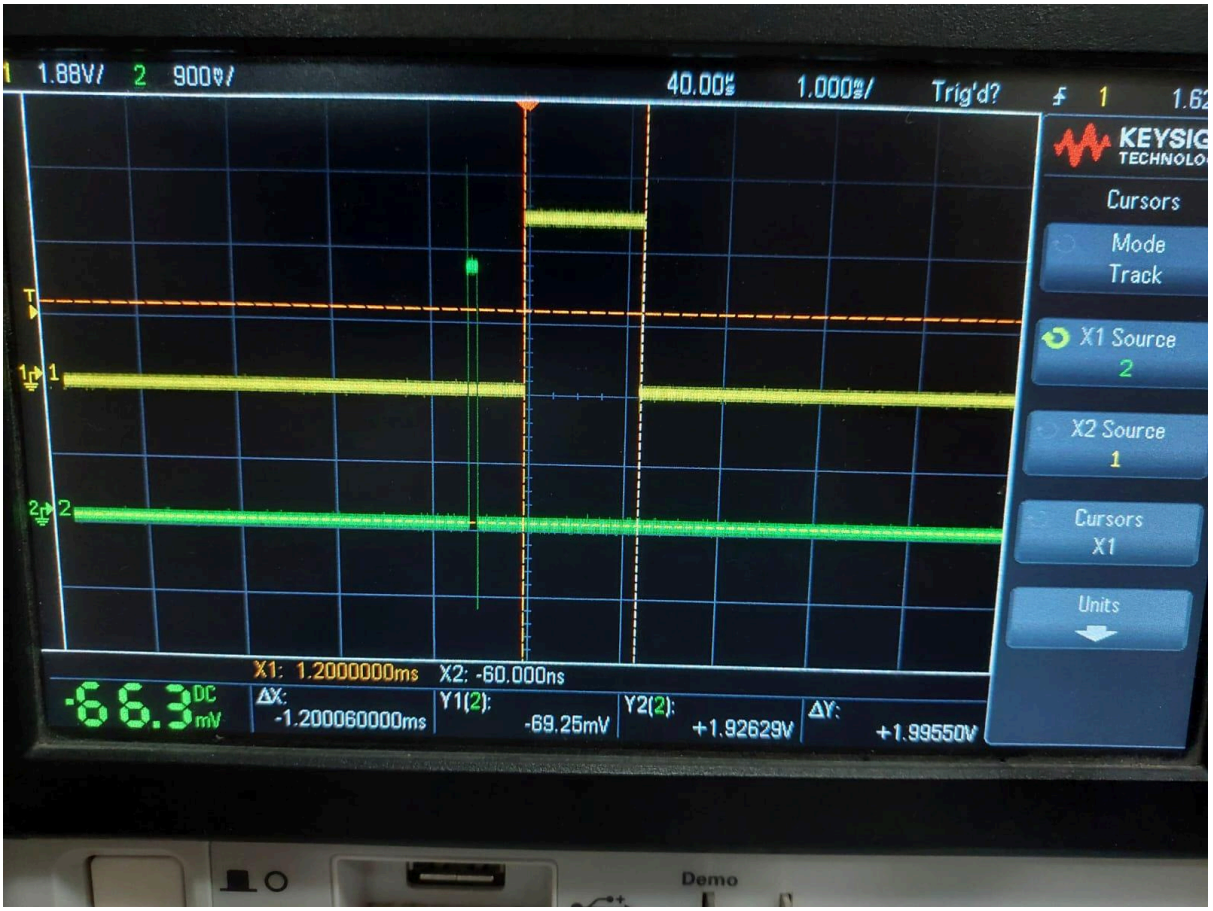
b. Complete the code by replacing the triple question marks with the correct snippet.

```
# IMPORTANT - Before running this program, execute the following command in the terminal - sudo pigpiod


import pigpio

import time


pi = pigpio.pi()


# Set GPIO Pins

TRIG = 4   # Replace with the actual GPIO pin number connected to TRIG

ECHO = 17  # Replace with the actual GPIO pin number connected to ECHO

pi.set_mode(TRIG, pigpio.OUTPUT)  # TRIG is an output pin

pi.set_mode(ECHO, pigpio.INPUT)   # ECHO is an input pin
```

```
def ownrise(gpio, level, tick):
    global high
    high = tick

def ownfall(gpio, level, tick):
    global low
    low = tick

# Use this pin to generate a 10 mu-s trigger signal
pi.gpio_trigger(TRIG, 10, 1)   # (channel number, trigger pulse width in mu-s, channel NOT level after trigger)

pi.callback(ECHO, pigpio.FALLING_EDGE, ownfall)
pi.callback(ECHO, pigpio.RISING_EDGE, ownrise)

if pi.wait_for_edge(ECHO, pigpio.FALLING_EDGE, 5.0):
    tof = high - low
    print(f"There is an object in {tof} microseconds distance.")
else:
    print('Timeout')

# Stop pigpio
pi.stop()
```

c. If you have filled the code correctly you receive a numerical output in the shell when you execute the code. Modify the existing code so you receive the output: “There is an object in xx cm distance.”

The output in the code gives the time of flight of the sound wave. For calculating the distance from the object, we can divide it by 2 (as it gives both outgoing and incoming time summation) and after that multiply it with speed of sound (343 m/s), also ensuring the correct unit conversion. The multiplying factor will be 17150.



Trigger and Echo Wave (Green and Yellow respectively)

Task 4(Lab 3) – The next goal is to compare the accuracy using hardware and software time-based measurements. Place the object at 2 cm, 5 cm and 10 cm and take for each distance 1000 measurement samples. Compute the average and the standard deviation of each distance and compare your results obtained by software-timed code and hardware-timed code. (You may use further libraries of Python numpy or matplotlib).

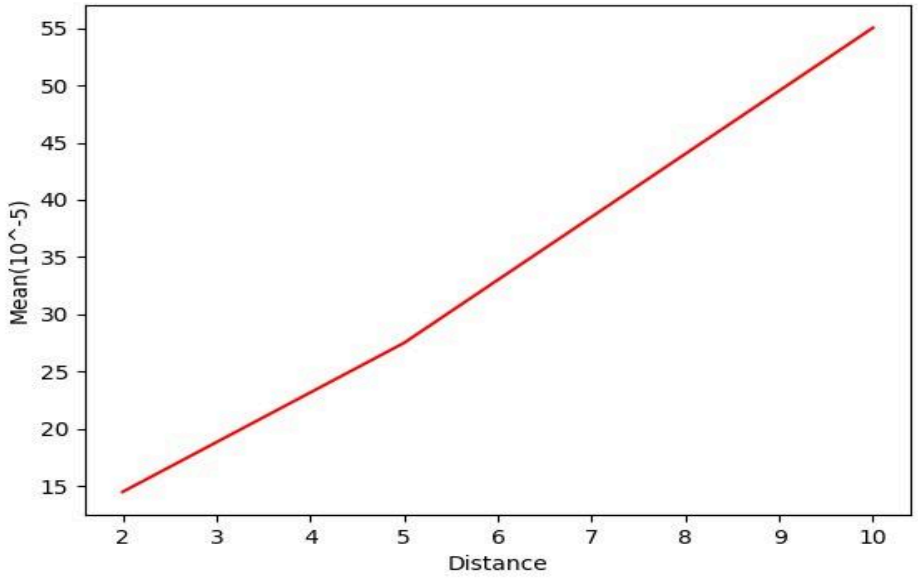
a. For the hardware-based measurement you can use the previous code and modify it to automate the acquisition of 1000 measurements.

```
import pigpio

import time

import statistics

import matplotlib.pyplot as plt


def main():

    pi = pigpio.pi()
```

```
TRIG = 4   # Replace with your TRIG pin number
ECHO = 17  # Replace with your ECHO pin number
pi.set_mode(TRIG, pigpio.OUTPUT)
pi.set_mode(ECHO, pigpio.INPUT)

num_samples = 1000  # Number of measurements per distance
distances = [2, 5, 10]  # Distances in centimeters to measure

for distance in distances:
    measurements = []
    for _ in range(num_samples):
        # Send trigger pulse
        pi.gpio_trigger(TRIG, 10, 1)  # Trigger for 10 microseconds

        # Measure time-of-flight
        if pi.wait_for_edge(ECHO, pigpio.FALLING_EDGE, 5.0):
            tof = (pi.read_bank_1() * 1000000) / 1000000.0  # microseconds
            measurements.append(tof)
        else:
            print('Timeout')
            measurements.append(None)

    average_tof = sum(measurements) / num_samples
    std_deviation = (sum((x - average_tof) ** 2 for x in measurements) / num_samples) **
0.5

    print(f"Distance: {distance} cm")
    print(f"Average Time-of-Flight: {average_tof} microseconds")
    print(f"Standard Deviation: {std_deviation} microseconds")
```

```
    pi.stop()
   # Plot the results
    plot_results(distances, average_tof_values)


def plot_results(distances, average_tof_values):
        plt.plot(distances, average_tof_values)
        plt.xlabel('Distance ')
        plt.ylabel('Mean(10^-5)')


if __name__ == '__main__':
    main()
```

b. For the software-based code you can download the template “lab3_4_software” and modify it to automate the acquisition of 1000 measurements (Careful: set the TRIG and ECHO pins according to your setup.)



The goal of this task is to compare the accuracy of hardware-based and software-based distance measurements using an HC-SR04 ultrasonic sensor. To do this, measurements are taken at object distances of 2 cm, 5 cm, and 10 cm, with each distance being measured 1000 times. For the hardware-based measurement, a code snippet is modified to automate the acquisition of these measurements. Similarly, for the software-based measurement, a provided template is adjusted for the same purpose. Average and standard deviation values are computed for each distance, and the results obtained from both methods are compared. Libraries like NumPy or Matplotlib can be used for data analysis and visualization. This comparison helps evaluate the precision and consistency of distance measurements using hardware and software approaches.

3. Conclusion

1. **What in your opinion is the most important thing you learnt from the experiment?**

First of all we learnt how the ultrasound distance measurement module (HC-SR04) works and measures the distances of the objects from the sensor. We learnt to connect the sensor to the Raspberry Pi and ways to operate it to generate waveforms on the Oscilloscope. We also learnt how to use pigpio commands and did the distance measurement for the hardware triggered timer. Finally, we did the comparison between the hardware and software-based timers, and measured the accuracies of both.

2. **What was interesting about the experiment?**

One of the interesting things which we observed were the waveforms generated on the Oscilloscope when the voltage levels of trigger and Echo pins were recorded. This was the first time we came to know that we could control the sensor triggers through the hardware-based trigger, and that was really interesting.

3. **What was challenging about the experiment?**

The major part of the experiment went quite smoothly. The major things we had to take care of were that connections were tight, to prevent any noise from being shown in the waveforms. Although we faced some problems with this in the beginning, we quickly caught up to and resolved the noise issues.

4. **Were there any drawbacks of the way the experiment was done? How would you do it better?**

The experiment was done with the best of efforts, and we ensured that there weren't any drawbacks from our part this time.

5. **Contribution statement** – In this experiment, each team member played a specific role to ensure its success.

Adarsh Santoria: Conducted circuit analysis, helped build the circuit, and assisted in data collection and coding.
Saransh Duharia: Assisted in setting up the oscilloscope, capturing data, and categorizing observations regarding jitter strength.
Jasnoor: Helped in Circuit execution, did coding of Raspberry Pi and Analyzed the collected data to visualize the results.
Rawal Ram: Setup the circuit and Helped in Oscilloscope execution.
Binduthraya: Managed the Raspberry Pi setup and executed the code,while ensuring the sensor was in working condition.
Collectively, our team worked together to design and execute the experiment, ensuring a comprehensive analysis of the results and a well-rounded understanding of the experiment's outcomes.