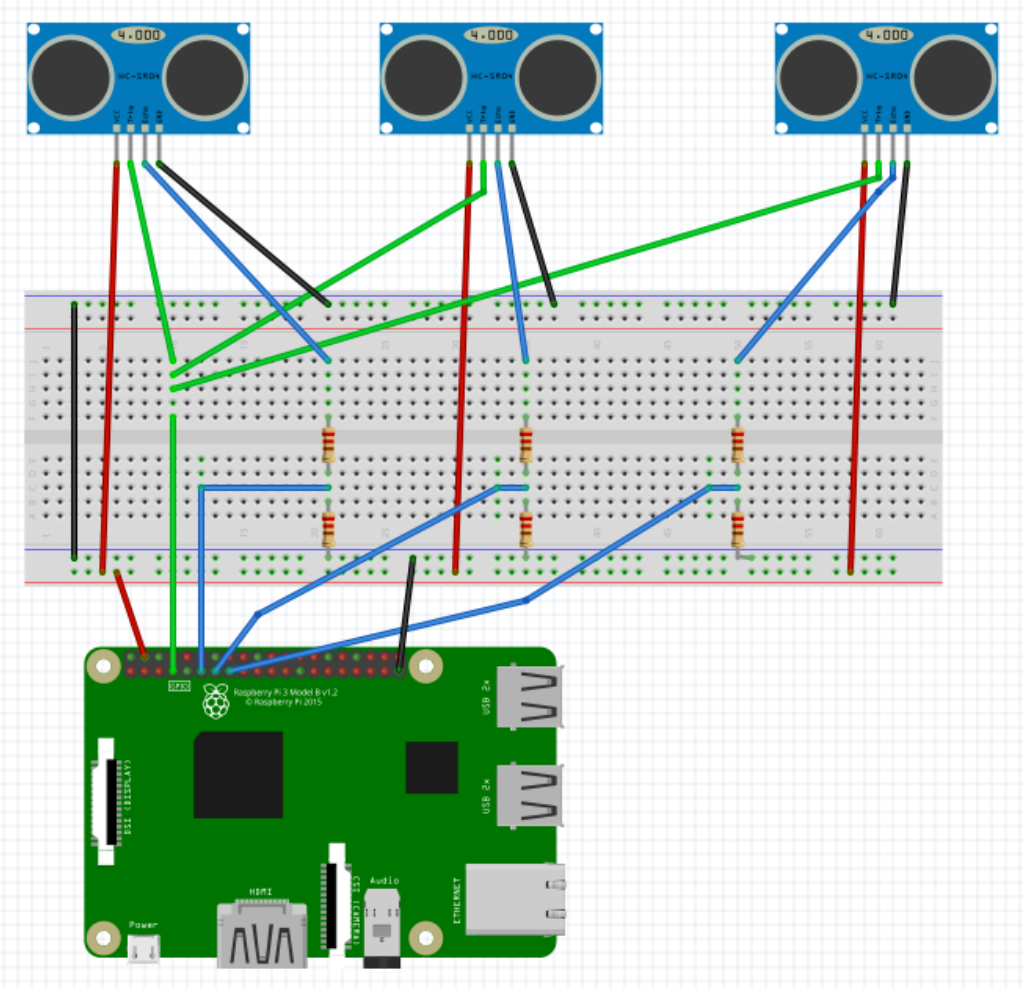
Phase 3 : project report

Smart parking management

Introduction:

Smart parking management systems have revolutionized urban transportation by leveraging sensor technology to optimize parking space utilization. This introduction explores the deployment of sensors in the context of smart parking management, shedding light on the transformative impact of these technologies on urban mobility and the numerous benefits they offer, from reduced traffic congestion to enhanced user experiences.

Circuit stimulation:



Circuit explanation:

The circuit above illustrates the stimulation of the Raspberry Pi with ultrasonic sensors, where various datasets are collected during the stimulation of the ultrasonic sensors. These data define whether an object can be placed and determine the range of the object (vehicle) from the sensors. The data collected from the sensors provide insights into the concept of parking systems.

- **Object Placement:** The first function of the collected data is to determine whether an object, such as a vehicle, can be placed in a specific location. The ultrasonic sensors likely measure distances and can help identify if there's enough space for the object to be accommodated without any obstructions.

- **Object Range Measurement:** The second function of the data is to ascertain the range of the object from the sensors. This range measurement is crucial for understanding how far an object is from the sensors, which is often critical for parking systems to ensure safe and accurate parking.

Dataset:

distance from	sensor 1(s1)	sensor 2(s2)	sensor 3(s3)	led
4-10 cm	slot not available	slot not available	slot not available	OFf
>10 cm	slot available	slot available	slot available	ON

Dataset explanation:

- **The first data** point informs us that a vehicle is currently occupying a parking slot, and furthermore, it indicates that there are no vacant slots available at that moment. This suggests that the parking area is fully occupied, and no additional vehicles can be accommodated.

- **The second data** point provides a different scenario. It indicates that there is an available parking slot for a vehicle. This means that a parking space has become vacant and is ready for use by another vehicle.

You can use this dataset for testing and experimentation, but please keep in mind that real-world sensor data might exhibit variations and noise that a simulation like this doesn't capture.

Code:

```
import RPi.GPIO as GPIO
import time

# Define the pins for the ultrasonic sensor
trigPin = 9  # Trigger pin
echoPin = 10  # Echo pin

def setup():
    GPIO.setmode(GPIO.BCM)
    GPIO.setup(trigPin, GPIO.OUT)
    GPIO.setup(echoPin, GPIO.IN)

def loop():
    try:
        while True:
            GPIO.output(trigPin, False)
            time.sleep(0.02)  # 2 microseconds
            GPIO.output(trigPin, True)
            time.sleep(0.00001)  # 10
microseconds
            GPIO.output(trigPin, False)

            while GPIO.input(echoPin) == 0:
                pulse_start = time.time()
            while GPIO.input(echoPin) == 1:
                pulse_end = time.time()

            duration = pulse_end - pulse_start
            distance = (duration * 34300) / 2
# Speed of sound in cm/s

            print(f"Distance: {distance:.2f}
cm")
            time.sleep(1)  # Wait for a moment
before the next reading
```

```
    except KeyboardInterrupt:
        GPIO.cleanup()

if __name__ == '__main__':
    setup()
    loop()
```

Code explanation:

```
import RPi.GPIO as GPIO
import time
```

- Import the RPi.GPIO library for working with Raspberry Pi's GPIO pins.
- Import the time module for timing-related functions.

```
trigPin = 9  # Trigger pin
echoPin = 10  # Echo pin
```

- Define variables trigPin and echoPin to hold the GPIO pin numbers connected to the ultrasonic sensor's trigger and echo pins.

```
def setup():
    GPIO.setmode(GPIO.BCM)
    GPIO.setup(trigPin, GPIO.OUT)
    GPIO.setup(echoPin, GPIO.IN)
```

- Define a function setup() for initializing the GPIO pins. It sets the GPIO mode to BCM (Broadcom SOC channel numbering) and configures trigPin as an output pin and echoPin

as an input pin.

```
def loop():
    try:
        while True:
```

- Define the main loop using a while True statement, which means it will run continuously until you manually stop the script

```
GPIO.output(trigPin, False)
            time.sleep(0.02)  # 2 microseconds
            GPIO.output(trigPin, True)
            time.sleep(0.00001)  # 10
microseconds
            GPIO.output(trigPin, False)
```

- Generate an ultrasonic pulse by setting the trigPin LOW, waiting for 2 microseconds (the corrected version of your original 2 microseconds), setting it HIGH for 10 microseconds, and then setting it LOW again. This pulse is sent to the ultrasonic sensor to trigger a distance measurement.

```
while GPIO.input(echoPin) == 0:
                pulse_start = time.time()
            while GPIO.input(echoPin) == 1:
                pulse_end = time.time()
```

- Measure the duration of the echo pulse by monitoring the echoPin. It records the time when the echo pulse starts (pulse_start) and when it ends (pulse_end).

```
duration = pulse_end - pulse_start
```

```
            distance = (duration * 34300) / 2
# Speed of sound in cm/s
```

- Calculate the distance from the measured duration. It uses the speed of sound in air, which is approximately 34300 cm/s, to calculate the distance the ultrasonic wave traveled and back. The result is divided by 2 since the signal has to travel to the object and back.

```
print(f"Distance: {distance:.2f} cm")
            time.sleep(1)  # Wait for a moment
before the next reading
```

- Print the calculated distance to the console with 2 decimal places.
- Introduce a 1-second delay before taking the next distance measurement.

```
except KeyboardInterrupt:
        GPIO.cleanup()
```

- The code handles a KeyboardInterrupt exception (usually caused by Ctrl+C) to ensure that it cleans up the GPIO pins using GPIO.cleanup() before exiting.

```
if __name__ == '__main__':
    setup()
    loop()
```

- This code block checks if the script is being run as the main program (not imported as a module). If so, it calls the setup() function to initialize the GPIO pins and then starts the main loop by calling loop().

Conclusion:

The provided code, adapted for Raspberry Pi, facilitates efficient management of parking spaces in the context of smart parking systems. By utilizing ultrasonic sensors and a Python script, this technology allows for real-time monitoring of parking spot availability. It offers advantages such as improved user experiences, reduced fuel consumption and emissions, data for analytics, and scalability. When integrated into smart parking solutions, this code enhances parking efficiency and contributes to sustainable urban mobility.