PHASE -3

PUBLIC TRANSPORT OPTIMIZATION

Deployed IOT devices:

1. **GPS Module**

GPS (Global Position System) used for positioning and tracking buses based on satellite communication. GPS satellites cover the entire earth at all times. To get accurate GPS location data, there should be a minimum of three satellites. The NEO-6M GPS module used in the proposed system is small and works on very low power, making it ideal for tracking applications. The GPS module operates at 3.3 V, as a result, powered by connecting the GPS module to the 3.3 V pin of the ESP32.

2. **ESP32 Microcontroller**

The ESP32 is a microcontroller with a Wi-Fi module, an open-source IoT platform that is characterized by low-cost and low-power system-on-a-chip (SOC). An ESP32 has a dual-core structure and internal modules such as Wi-Fi, Bluetooth, and many Peripheral Interfaces such as IR, SPI, CAN, Ethernet, and temperature sensors

3. **Ultrasonic Sensor**

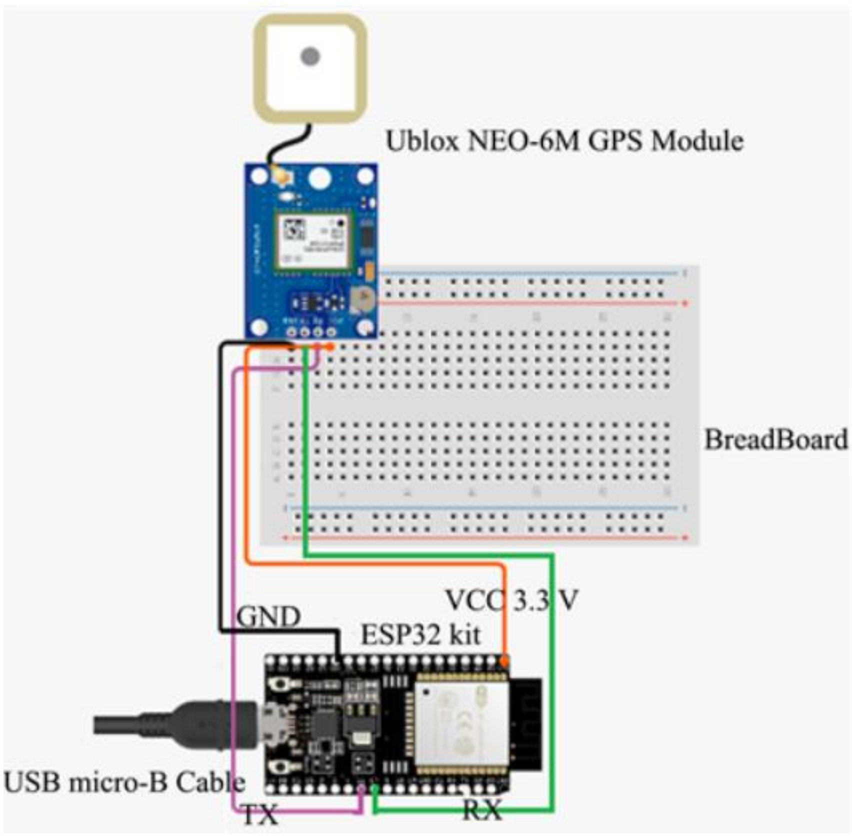
Ultrasonic sensors are commonly used with Arduino to measure distance by sending and receiving ultrasonic waves. Connect the VCC pin of the ultrasonic sensor to the 5V pin on the Arduino. Connect the GND pin of the ultrasonic sensor to the GND pin on the Arduino. Connect the TRIG pin of the ultrasonic sensor to a digital pin (e.g., Pin 7) on the Arduino. Connect the ECHO pin of the ultrasonic sensor to another digital pin (e.g., Pin 6) on the Arduino.

ARDUINO CODE FOR ESP32 MICROCONTROLLER:

This code enables an ESP32 to interact with Blynk, read distances from two ultrasonic sensors, and update the Blynk application with information on people entering and leaving a defined area. It also controls an LED based on these conditions. The distances are measured using the Haversine formula, which is common for calculating distances between latitude and longitude coordinates.

```
#define BLYNK_TEMPLATE_ID "TMPL26V4fGv5q"
#define BLYNK_TEMPLATE_NAME "Test"
#define BLYNK_AUTH_TOKEN "XEHxNF_Ur1Nt2p7wB5B20dNI1ZUwj34P"

#include <WiFi.h>
#include <WiFiClient.h>
```

```
#include <BlynkSimpleEsp32.h>

int duration1 = 0;
int distance1 = 0;
int duration2 = 0;
int distance2 = 0;
int dis1 = 0;
int dis2 = 0;
int dis_new1 = 0;
int dis_new2 = 0;
int entered = 0;
int left = 0;
int inside = 0;
#define LED 2
#define PIN_TRIG1 15
#define PIN_ECHO1 14
#define PIN_TRIG2 13
#define PIN_ECHO2 12
BlynkTimer timer;

char auth[] = BLYNK_AUTH_TOKEN;
char ssid[] = "Wokwi-GUEST";   // your network SSID (name)
char pass[] = "";
#define BLYNK_PRINT Serial

long get_distance1() {
  // Start a new measurement:
  digitalWrite(PIN_TRIG1, HIGH);
  delayMicroseconds(10);
  digitalWrite(PIN_TRIG1, LOW);

  // Read the result:
  duration1 = pulseIn(PIN_ECHO1, HIGH);
  distance1 = duration1 / 58;
  return distance1;
}

long get_distance2() {
  // Start a new measurement:
  digitalWrite(PIN_TRIG2, HIGH);
  delayMicroseconds(10);
  digitalWrite(PIN_TRIG2, LOW);

  // Read the result:
  duration2 = pulseIn(PIN_ECHO2, HIGH);
  distance2 = duration2 / 58;
  return distance2;
}
int count = 0;
void myTimer() {
  dis_new1 = get_distance1();
  dis_new2 = get_distance2();
  if (dis_new1<100){
    count++;
  Serial.println("Number of passengers inside the bus:");
  Serial.println(count);
  }
  if (dis_new2<100){
    if(count<=0){
      count=0;
    }
    else{
      count--;
    }
  Serial.println("Number of passengers inside the bus:");
  Serial.println(count);

  }
}

 void setup() {
  Serial.begin(115200);
  pinMode(LED, OUTPUT);
  pinMode(PIN_TRIG1, OUTPUT);
  pinMode(PIN_ECHO1, INPUT);
  pinMode(PIN_TRIG2, OUTPUT);
  pinMode(PIN_ECHO2, INPUT);
  Blynk.begin(auth, ssid, pass, "blynk.cloud", 8080);
  timer.setInterval(1000L, myTimer);

}

void loop() {
  Blynk.run();
  timer.run();
}
```



ARRIVAL TIME CALCULATION:

Python script:

```
class ArrivalTimeCalculator:
    def __init__(self):
        self.prev_coordinate = (0, 0)  # Initialize previous coordinate to (0, 0)

    def calculate_arrival_time(self, distance, average_speed):
        # Calculate arrival time in minutes
        arrival_time = (distance / average_speed) * 60
        return arrival_time

    def process_new_coordinates(self, new_coordinate, distance, average_speed):
        if self.prev_coordinate == new_coordinate:
            # Start timer and calculate delay time
            delay_time = self.start_timer_and_get_delay()

            # Calculate final arrival time
            final_arrival_time = self.calculate_arrival_time(distance, average_speed) +
delay_time
            self.prev_coordinate = new_coordinate  # Update prev coordinate
            return final_arrival_time
        else:
            # Process new coordinates and calculate estimated arrival time
            estimated_arrival_time = self.calculate_arrival_time(distance, average_speed)

            # Update prev coordinate
            self.prev_coordinate = new_coordinate
            return estimated_arrival_time

    def start_timer_and_get_delay(self):
        # Placeholder function for starting timer and getting delay time
        # You should implement a timer function based on your specific environment
        delay_time = 5  # Placeholder value, replace with actual delay time calculation
        return delay_time


# Example Usage
arrival_time_calculator = ArrivalTimeCalculator()

# Example coordinates
new_coordinate1 = (12.34, 56.78)  # Example new coordinate (latitude, longitude)
new_coordinate2 = (12.34, 56.78)  # Example new coordinate (latitude, longitude)

distance = 10  # Example distance in kilometers
average_speed = 40  # Example average speed in km/h

final_arrival_time1 = arrival_time_calculator.process_new_coordinates(new_coordinate1,
distance, average_speed)
print(f"The final arrival time for first coordinate is approximately {final_arrival_time1}
minutes.")

final_arrival_time2 = arrival_time_calculator.process_new_coordinates(new_coordinate2,
distance, average_speed)
print(f"The final arrival time for second coordinate is approximately
{final_arrival_time2} minutes.")
```

Python script on the IoT sensors to send real-time location and ridership data to the transit information platform:

```
import paho.mqtt.client as mqtt
import json
import time
from your_sensor_module import get_real_sensor_data

# MQTT broker information
broker_address = "mqtt.eclipse.org"
broker_port = 1883
topic = "transit_data"

def generate_real_data():
    location_data = gps_module.get_location_data()
    ridership_data = ultrasonic_sensor_module.get_ridership_data()

    return {
        "location": location_data,
        "ridership": ridership_data
    }
```

```python
# MQTT client setup
client = mqtt.Client()
client.connect(broker_address, broker_port, 60)

try:
    while True:
        # Generate real sensor data
        data = generate_real_data()

        # Convert data to JSON
        payload = json.dumps(data)

        # Publish data to the topic
        client.publish(topic, payload)

        # Print for verification
        print("Published:", payload)

        # Adjust the frequency based on your requirements
        time.sleep(10)  # Wait for 10 seconds before sending the next data

except KeyboardInterrupt:
    print("Script terminated by user.")
finally:
    # Disconnect from the broker
    client.disconnect()
```