```
#include <Adafruit_MPU6050.h> // Include MPU6050 library
#include <SPI.h>
#include <WiFi.h>
#include <Wire.h> // Include Wire library for I2C(Inter-Integrated Circuit)
communication
#include <esp_now.h>

Adafruit_MPU6050 g_mpuSensor; ///< Create an instance of the
                              ///< MPU6050 sensor

esp_now_peer_info_t
    g_peerInfoFirst; ///< Peer information for ESP-NOW
                     ///< communication with the first peer
esp_now_peer_info_t
    g_peerInfoSecond; ///< Peer information for ESP-NOW
                      ///< communication with the second
                      ///< peer

uint8_t g_arrBroadcastAddress[2][6] = {
    {0xEC, 0x64, 0xC9, 0x86, 0x13, 0xC0},
    {0xEC, 0x64, 0xC9, 0x82, 0x7E,
     0x10}}; ///< MAC addresses for broadcast

String g_arrSSID[2] = {
    "BUS_STOP_A", "BUS_STOP_B"}; ///< SSIDs of the bus stops

int g_iVertex = 0; ///< Vertex index

const int g_iButtonPin = 0; ///< Pin number for the button

int g_iOOSFlag = 0; ///< Flag for out of service status

int g_iHopeFlag = 0; ///< Flag for hope status

volatile bool g_bButtonPressed =
    false; ///< Flag to indicate if the button is pressed

bool g_bPassengerFlag =
    false; ///< Flag for passenger status

int g_iHaltFlag = 0; ///< Flag for halt status

int g_iLatchFlag = 0; ///< Flag for latch status
/**
 * @struct tagMESSAGE
 * @brief Structure for sending and receiving data.
 *
 * @var tagMESSAGE::strMessage
 * @brief A string message.
 * @var tagMESSAGE::iBusStopIndex
 * @brief The index of the bus stop.
 * @var tagMESSAGE::iBusIndex
```

```
 * @brief The index of the bus.
 * @var tagMESSAGE::iDirection
 * @brief The direction of the bus.
 *
 * This structure must match the sender structure.
 */
typedef struct tagMESSAGE
{
    String strMessage;
    int iBusStopIndex;
    int iBusIndex;
    int iDirection;
} MESSAGE;
// Create a struct_message called myData
MESSAGE g_myData; ///< Instance of MESSAGE to hold data.

const int g_iLedPin = 2; ///< Pin number for the LED.

/**
 * @struct tagMESSAGE_RECV
 * @brief Structure to receive data.
 *
 * @var tagMESSAGE_RECV::bButtonPressed
 * @brief Flag to indicate if the button is pressed.
 */
typedef struct
{
    bool bButtonPressed;
} MESSAGE_RECV;

/**
 * @brief Callback function that is executed when data is
 * received.
 *
 * This function handles the incoming data and sets the
 * passenger flag to true if a message is received.
 *
 * @param pucMac The MAC address of the sender.
 * @param pucIncomingData The incoming data.
 * @param iLen The length of the incoming data.
 */
void onDataRecv(const uint8_t *pucMac,
                const uint8_t *pucIncomingData, int iLen)
{
    MESSAGE_RECV msgRecv;
    memcpy(&msgRecv, pucIncomingData, sizeof(msgRecv));
    Serial.println("passengers waiting");

    g_bPassengerFlag = true;
}
//////////////////////////////////////////
/**
 * @brief Sends a service message to the bus stops.
 *
```

```
 * This function sends a service message (either "BUS OOS"
 * or "BUS NOOS") to the bus stops.
 *
 * @param iIndex The index of the message to be sent.
 */
void send_data_service(int iIndex)
{
    if (iIndex == 0)
    {
        g_myData.strMessage = "BUS OOS";
        g_myData.iBusIndex = 1;
        g_myData.iBusStopIndex = 0;
        g_myData.iDirection = 0;
        esp_err_t eResult = esp_now_send(
            g_arrBroadcastAddress[0], (uint8_t *)&g_myData,
            sizeof(g_myData));

        if (eResult == ESP_OK)
        {
            Serial.println("Sent with success");
        }
        else
        {
            Serial.println("Error sending the data");
        }

        eResult = esp_now_send(g_arrBroadcastAddress[1],
                               (uint8_t *)&g_myData,
                               sizeof(g_myData));

        if (eResult == ESP_OK)
        {
            Serial.println("Sent with success");
        }
        else
        {
            Serial.println("Error sending the data");
        }
    }

    else
    {
        g_myData.strMessage = "BUS NOOS";
        g_myData.iBusIndex = 1;
        g_myData.iBusStopIndex = 0;
        g_myData.iDirection = 0;
        esp_err_t eResult = esp_now_send(
            g_arrBroadcastAddress[0], (uint8_t *)&g_myData,
            sizeof(g_myData));

        if (eResult == ESP_OK)
        {
            Serial.println("Sent with success");
        }
```

```
        else
        {
            Serial.println("Error sending the data");
        }

        eResult = esp_now_send(g_arrBroadcastAddress[1],
                               (uint8_t *)&g_myData,
                               sizeof(g_myData));

        if (eResult == ESP_OK)
        {
            Serial.println("Sent with success");
        }
        else
        {
            Serial.println("Error sending the data");
        }
    }
}
/**
 * @brief Interrupt service routine for handling button
 * press.
 *
 * This function is called when an interrupt is triggered.
 * It toggles the g_bButtonPressed flag.
 */
void IRAM_ATTR handleInterrupt()
{
    g_bButtonPressed = !g_bButtonPressed;
}

/**
 * @brief Callback function that is executed when data is
 * sent.
 *
 * This function prints the status of the last packet sent.
 * It prints "Delivery Success" if the packet was sent
 * successfully, otherwise it prints "Delivery Fail".
 *
 * @param pucMacAddr The MAC address of the receiver.
 * @param eStatus The status of the send operation.
 */
void OnDataSent(const uint8_t *pucMacAddr,
                esp_now_send_status_t eStatus)
{
    Serial.print("\r\nLast Packet Send Status:\t");
    Serial.println(eStatus == ESP_NOW_SEND_SUCCESS
                       ? "Delivery Success"
                       : "Delivery Fail");
    if (eStatus == ESP_NOW_SEND_SUCCESS)
    {
    }
    else
    {
```

```
    }
}
/**
 * @brief Setup function for the Arduino sketch.
 *
 * This function initializes the serial communication, sets
 * the button pin as input and attaches an interrupt to it,
 * initializes the MPU6050 sensor, sets the accelerometer
 * range and filter bandwidth, initializes ESP-NOW,
 * registers the send callback function, adds peers for
 * ESP-NOW communication, registers the receive callback
 * function, and sets the LED pin as output.
 */
void setup(void)
{
    Serial.begin(115200); // Initialize serial communication
    while (!Serial)
        delay(10); // Wait for serial port to connect

    pinMode(g_iButtonPin, INPUT_PULLUP);
    attachInterrupt(digitalPinToInterrupt(g_iButtonPin),
                    handleInterrupt, FALLING);
    Serial.println(
        "Adafruit MPU6050 test!"); // Print a message to
                                   // serial monitor

    if (!g_mpuSensor.begin())
    { // Try to initialize MPU6050 sensor
        Serial.println("Failed to find MPU6050 chip");
        while (1)
        {
            delay(10);
        }
    }
    Serial.println(
        "MPU6050 Found!"); // Print a message indicating
                           // MPU6050 is found

    g_mpuSensor.setAccelerometerRange(
        MPU6050_RANGE_8_G); // Set accelerometer range to
                            // +/- 8G
    Serial.print("Accelerometer range set to: ");
    switch (g_mpuSensor.getAccelerometerRange())
    {
    case MPU6050_RANGE_8_G:
        Serial.println("+-8G");
        break;
    }

    g_mpuSensor.setFilterBandwidth(
        MPU6050_BAND_21_HZ); // Set filter bandwidth to 21
                             // Hz
    Serial.print("Filter bandwidth set to: ");
    switch (g_mpuSensor.getFilterBandwidth())
```

```
    {
    case MPU6050_BAND_21_HZ:
        Serial.println("21 Hz");
        break;
    }

    Serial.println(""); // Print empty line for readability
    delay(100);         // delay for stabilization

    WiFi.mode(WIFI_AP_STA);
    WiFi.softAP("BUS", NULL, 4);

    // Init ESP-NOW
    if (esp_now_init() != ESP_OK)
    {
        Serial.println("Error initializing ESP-NOW");
        return;
    }

    // Once ESPNow is successfully Init, we will register
    // for Send CB to get the status of Trasnmitted packet
    esp_now_register_send_cb(OnDataSent);

    memcpy(g_peerInfoFirst.peer_addr,
           g_arrBroadcastAddress[0], 6);
    g_peerInfoFirst.channel = 4;
    g_peerInfoFirst.encrypt = false;
    if (esp_now_add_peer(&g_peerInfoFirst) != ESP_OK)
    {
        Serial.println("Failed to add peer");
        return;
    }

    memcpy(g_peerInfoSecond.peer_addr,
           g_arrBroadcastAddress[1], 6);
    g_peerInfoSecond.channel = 4;
    g_peerInfoSecond.encrypt = false;
    if (esp_now_add_peer(&g_peerInfoSecond) != ESP_OK)
    {
        Serial.println("Failed to add peer");
        return;
    }

    ////////////////////////////////////////////////////
    esp_now_register_recv_cb(onDataRecv);

    pinMode(g_iLedPin, OUTPUT);   // Set LED pin as output
    digitalWrite(g_iLedPin, LOW); // Start with LED off
    ////////////////////////////////////////////////////
}
/**
 * @brief Gets the RSSI value for a specific bus stop.
 *
 * This function scans the networks and returns the RSSI
```

```
 * value for a specific bus stop.
 *
 * @param iBusStopIndex The index of the bus stop.
 * @return The RSSI value. Returns 0 if the RSSI is greater
 * than or equal to -50, 2 if the SSID matches the bus
 * stop, and 1 otherwise.
 */
int get_rssi(int iBusStopIndex)
{
    Serial.println("scan start");

    // WiFi.scanNetworks will return the number of networks
    // found
    int iNetworkCount =
        WiFi.scanNetworks(false, false, false, 50, 4);
    Serial.println("scan done");
    if (iNetworkCount == 0)
    {
        Serial.println("no networks found");
    }
    else
    {
        Serial.print(iNetworkCount);
        Serial.println(" networks found");
        for (int i = 0; i < iNetworkCount; ++i)
        {
            // Print SSID and RSSI for each network found
            Serial.print(i + 1);
            Serial.print(": ");
            Serial.print(WiFi.SSID(i));
            Serial.print(" (");
            Serial.print(WiFi.RSSI(i));
            Serial.print(")");
            Serial.println(
                (WiFi.encryptionType(i) == WIFI_AUTH_OPEN)
                    ? " "
                    : "*");

            if (WiFi.SSID(i) == g_arrSSID[iBusStopIndex])
            {
                Serial.println(WiFi.RSSI(i));
                if (WiFi.RSSI(i) >= -50)
                {
                    // esp_err_t eResult =
                    // esp_now_send(g_arrBroadcastAddress,
                    // (uint8_t *) &g_myData,
                    // sizeof(g_myData));
                    return (0);
                }
                return (2);
            }
        }
        return (1);
    }
```

```
    Serial.println("");
}
int g_iAvgSample = 0;
int g_arrSamples[10];

/**
 * @brief Sends a data message to the bus stops.
 *
 * This function sends a data message (either "BUS
 * STOPPED!" or "BUS STARTED!") to the bus stops.
 *
 * @param iMessageIndex The index of the message to be sent.
 * If iMessageIndex is 0, "BUS STOPPED!" is sent. Otherwise,
 * "BUS STARTED!" is sent.
 * @param iBusStopIndex The index of the bus stop to send
 * the message to.
 */
void send_data(int iMessageIndex, int iBusStopIndex)
{
    if (iMessageIndex == 0)
    {
        g_myData.strMessage = "BUS STOPPED!";
        g_myData.iBusIndex = 1;
        g_myData.iBusStopIndex = 0;
        g_myData.iDirection = 0;
        esp_err_t eResult = esp_now_send(
            g_arrBroadcastAddress[iBusStopIndex],
            (uint8_t *)&g_myData, sizeof(g_myData));

        if (eResult == ESP_OK)
        {
            Serial.println("Sent with success");
        }
        else
        {
            Serial.println("Error sending the data");
        }
    }
    else
    {
        g_myData.strMessage = "BUS STARTED!";
        g_myData.iBusIndex = 1;
        g_myData.iBusStopIndex = 0;
        g_myData.iDirection = 0;
        esp_err_t eResult = esp_now_send(
            g_arrBroadcastAddress[iBusStopIndex],
            (uint8_t *)&g_myData, sizeof(g_myData));

        if (eResult == ESP_OK)
        {
            Serial.println("Sent with success");
        }
        else
        {
```

```
            Serial.println("Error sending the data");
        }
    }
}
/**
 * @brief Sends a halt message to the bus stops.
 *
 * This function sends a halt message (either "BUS HALTED"
 * or "BUS MOVING") to the bus stops.
 *
 * @param iMessageIndex The index of the message to be sent.
 * If iMessageIndex is 0, "BUS HALTED" is sent. Otherwise,
 * "BUS MOVING" is sent.
 * @param iBusStopIndex The index of the bus stop to send
 * the message to.
 */
void send_data_halt(int iMessageIndex, int iBusStopIndex)
{
    if (iMessageIndex == 0)
    {
        g_myData.strMessage = "BUS HALTED";
        g_myData.iBusIndex = 1;
        g_myData.iBusStopIndex = 0;
        g_myData.iDirection = 0;
        esp_err_t eResult = esp_now_send(
            g_arrBroadcastAddress[iBusStopIndex],
            (uint8_t *)&g_myData, sizeof(g_myData));

        if (eResult == ESP_OK)
        {
            Serial.println("Sent with success");
        }
        else
        {
            Serial.println("Error sending the data");
        }
    }
    else
    {
        g_myData.strMessage = "BUS MOVING";
        g_myData.iBusIndex = 1;
        g_myData.iBusStopIndex = 0;
        g_myData.iDirection = 0;
        esp_err_t eResult = esp_now_send(
            g_arrBroadcastAddress[iBusStopIndex],
            (uint8_t *)&g_myData, sizeof(g_myData));

        if (eResult == ESP_OK)
        {
            Serial.println("Sent with success");
        }
        else
        {
            Serial.println("Error sending the data");
```

```
        }
    }
}
/**
 * @brief Main loop function for the Arduino sketch.
 *
 * This function continuously checks if the button is
 * pressed and sends a service message if it is. It also
 * checks if a passenger signal is received and turns on
 * the LED if it is. It calculates the average sample of
 * the accelerometer readings and determines if the bus is
 * moving or stopped based on the average. If the bus is
 * moving, it sends a "BUS MOVING" message. If the bus is
 * stopped, it sends a "BUS STOPPED" message. It also
 * checks the RSSI values and sends a halt message if the
 * bus is near a bus stop.
 */
void loop()
{
    /* Get new sensor events with the readings */

    if (g_bButtonPressed)
    {
        if (g_iHopeFlag == 0)
        {
            Serial.println("BUS OUT OF SERVICE!");
            send_data_service(0);
            g_iOOSFlag = 1;
        }
        g_iHopeFlag = 1;
    }
    if (g_iOOSFlag == 1 && !g_bButtonPressed)
    {
        Serial.println("CONTINUING SERVICE");
        send_data_service(1);
        g_iOOSFlag = 0;
        g_iHopeFlag = 0;
    }

    if (g_bPassengerFlag)
    {
        Serial.println("Received signal, turning on LED");
        for (int i = 0; i < 2; i++)
        {
            digitalWrite(g_iLedPin, HIGH); // Turn on LED
            delay(250);
            digitalWrite(g_iLedPin, LOW);
            delay(250);
        }
        g_bPassengerFlag = false;
    }

    if (g_iAvgSample == 10)
    {
```

```
        int iSum = 0;
        for (int j = 0; j < 10; j++)
        {
            iSum = iSum + g_arrSamples[j];
        }
        g_iAvgSample = 0;
        if (iSum >= 7)
        {
            Serial.println("Bus Moving");
            if (g_iLatchFlag == 1)
            {
                if (g_iHaltFlag == 0)
                {
                    send_data(1, g_iVertex % 2);
                    g_iLatchFlag = 0;
                    g_iVertex++;
                }
                else
                {
                    send_data_halt(1, g_iVertex % 2);
                    send_data_halt(1, (g_iVertex + 1) % 2);
                    g_iLatchFlag = 0;
                    g_iHaltFlag = 0;
                }
            }
        }
        else
        {
            Serial.println("Bus stopped");
            int iRSSI1 = get_rssi(g_iVertex % 2);
            int iRSSI2 = get_rssi((g_iVertex + 1) % 2);
            if (iRSSI1 == 0 && g_iLatchFlag == 0)
            {
                Serial.println("bus arrived at");
                Serial.println(g_arrSSID[g_iVertex % 2]);
                send_data(0, g_iVertex % 2);
                g_iLatchFlag = 1;
            }
            else if ((iRSSI1 == 2 || iRSSI2 == 2) &&
                     g_iLatchFlag == 0)
            {
                send_data_halt(0, g_iVertex % 2);
                send_data_halt(0, (g_iVertex + 1) % 2);
                g_iLatchFlag = 1;
                g_iHaltFlag = 1;
            }
        }
    }

    sensors_event_t a, g, temp;
    g_mpuSensor.getEvent(&a, &g, &temp);
    int arrFlags[5]; // Array to store motion detection
                     // flags
    double dNet;
```

```
    // Loop to check motion for 10 iterations
    for (int i = 0; i < 5; i++)
    {
        double dAx =
            a.acceleration.x; // Acceleration along x-axis
        double dAy =
            a.acceleration.y; // Acceleration along y-axis
        double dAz =
            a.acceleration.z; // Acceleration along z-axis
        dNet =
            sqrt((dAx * dAx) + (dAy * dAy) +
                 (dAz * dAz)); // Calculate net acceleration

        // Check if net acceleration falls within a certain
        // range from the offset (acceleration when bus is
        // stopped)
        if (dNet > 9.4 and dNet < 10.3)
        {
            arrFlags[i] =
                0; // Set flag to 0 indicating bus stopped
        }
        else
        {
            arrFlags[i] =
                1; // Set flag to 1 indicating bus moving
        }
        delay(
            100); // Delay betrween iterations for stability
    }
    int iSum = 0;
    // Calculate sum of motion detection flags
    for (int i = 0; i < 5; i++)
    {
        iSum = iSum + arrFlags[i];
    }
    // Check if sum indicates bus stopped or moving
    if (iSum == 0)
    {
        Serial.println(
            "BS"); // Print message indicating bus stopped
        g_arrSamples[g_iAvgSample] = 0;
    }
    else
    {
        Serial.println(
            "BM"); // Print message indicating bus is moving
        g_arrSamples[g_iAvgSample] = 1;
    }
    g_iAvgSample = g_iAvgSample + 1;
    Serial.println(""); // Print empty line for readability
}
```