Appendix A

Methodology

I. Revisions from EEE 196

A. Removal of Outdoor Sensor Node and Driver Alert System
Lack of approval from a bus company to allow the installation of an outdoor node on the exterior of the bus. With this, the driver alert system for the air-conditioning controls was also removed due to the lack of outdoor air quality data.

B. EDKs/SDKs Used

1. ESP-IDE w/ ESP-IDF (https://dl.espressif.com/dl/esp-idf/)

C. Libraries Used

Library	Link to Source
ESP-Zigbee-SDK	https://github.com/espressif/esp-zigbee-sdk
Wi-Fi MAC Sniffer	https://www.hackster.io/p99will/esp32-wifi-mac-scanner-sniffer-promiscuous-4c12f4 https://docs.espressif.com/projects/esp-idf/en/stable/esp32/api-reference/network/esp_wifi.html?highlight=wifi_promiscuous_pkt_t#_CPPv422wifi_promiscuous_pkt_t
Openthread C API	https://github.com/openthread/openthread/tree/main/include/openthread
ESP-BLE-MESH API	https://docs.espressif.com/projects/esp-idf/en/latest/esp32/api-reference/bluetooth/esp-ble-mesh.html#api-reference

D. Hardware Components

Type	**Component/Model Name**
MCU	ESP32-H2-DevKitM-1 ESP Thread Border Router/Zigbee Gateway v1.2
Sensors	SEN55 + CAB-18079 (Cable) CJMCU-4541 (MiCS-4514) Adafruit 3709 (SGP30)
Supplementary	Neo6MV2 GPS Module

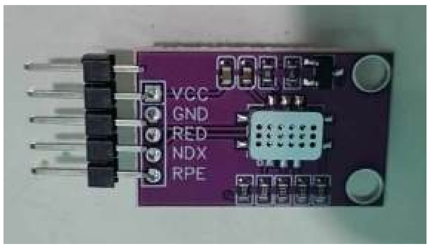
	SD Card Module 32GB SD Card 10mm LEDs MB102 Breadboard Power Supply PCB printed by Elecrow 10000mAh Powerbank

Appendix B

Problems Encountered in Project Implementation

I. Espressif IDE & IDF
The ZigBee sample code provided from the installed ESP-IDE was outdated (v0.7) while the the esp-zigbee-sdk GitHub repository was updated to ~v1.3. This resulted in limitations like string length. The solution was to update the IDF to v5.2.1, however the IDE encounters errors when doing so, at the time of development. Maximum string length was tested to be 75 characters/bytes with first byte being "K".

II. MiCS-4514 Pinouts
The MiCS-4514 ordered from the UP CARE components database had through-holes that were not spaced for compatibility with breadboards (2.54mm pitch). Although, it had provided 2.54mm pitch header pins.



The delivered MiCS-4514 and header pins

III. ESP32-H2 Analog-to-Digital Converter
The output of all ADC channels did not range from the typical range of 0-4095, instead it ranged from 2160-4081. All ESP32-H2 development kits resulted in the same behavior. Tests with a digital power supply resulted in the following ranges:

ADC Output Attenuation	Voltage Input Range	ADC Output Range
0 dB	0-500 mV	2160-4081
12 dB	0-1900 mV	

Testing an ordinary ESP32 (NodeMCU-ESP32S) resulted in normal ADC behavior.

IV. Neo6M V2 GPS Polling Rate & Antenna
Since the planned upload rate was 10 seconds, the planned GPS polling rate would also be similar. However, its output would be very inconsistent/inaccurate at any polling rate other than every 1 second. Antenna needs to be placed upward for fast signal reception.

V. BLE-MESH Number of Simultaneous Incoming/Outgoing Message Segmentations
Since the string lengths consisted around 300-400 bytes, an increase in the allowed number of simultaneous incoming/outgoing message segmentations was needed since it would cause errors after some time. This was changed from the default value of 10 to 20 in the sdkconfig of the project.

VI. SEN55 Grounding
One of the SEN55 sensors encountered a grounding problem where the metal chassis was not grounded (open circuit with GND pin) which caused inflated PM levels. Solution was to stick an additional GND wire to the chassis.

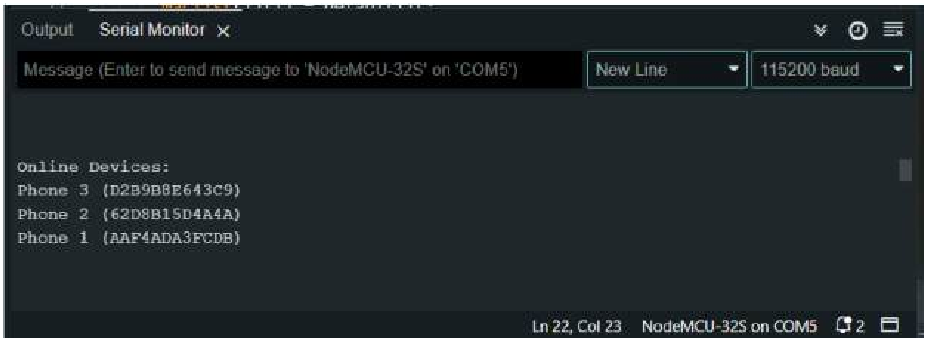
Appendix C

Other Results and Discussion

I. Occupancy Tests

There are several tests performed in order to evaluate the performance of the Occupancy Counter via WiFi Scanning.

MAC Address Verification. The goal of this test was to check whether the ESP32 sensor was reading actual MAC addresses. This was done by collecting the MAC address of the researcher's mobile devices and then comparing the resulting data printed on the laptop screen. As seen in the photo below, the device is able to successfully detect the correct MAC address of all three control phones.

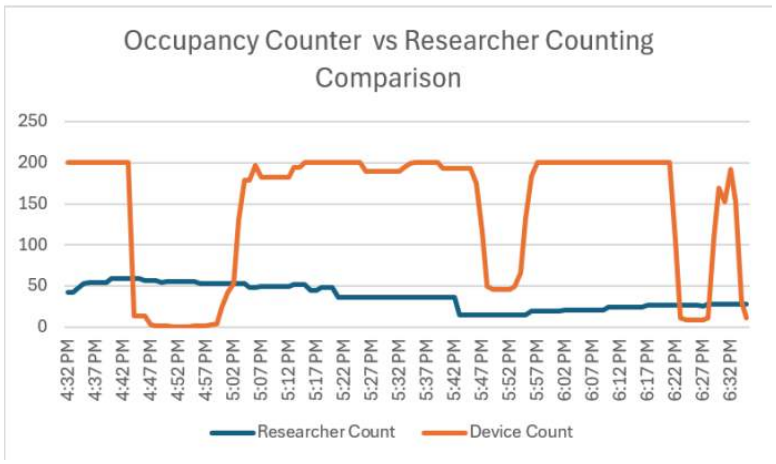


```
Online Devices:
Phone 3 (D2B9B8E643C9)
Phone 2 (62D8B15D4A4A)
Phone 1 (AAF4ADA3FCDB)
```

Stationary Counting Test. This test is conducted in different isolated locations where the researcher has a clear line of sight to all people or devices in the area. The accuracy of the Occupancy Counter is compared to the Manual Count of the researcher. As seen in the table below, the device has an average accuracy of ~80% in different areas. It is able to successfully estimate the number of devices in an area.

Setting	Accuracy
Outdoor Gazebo	88%
Outdoor Parking Lot	80%
Large Lecture Classroom	74%

Bus Counting Test. This test is conducted on the EDSA Carousel Bus with the Occupancy Sensor placed at the center. According to the results, there was almost no correlation between the Occupancy Sensor and Manual Count. According to the time plot, the occupancy count spikes to max (200) when near malls and train stations, then drops to near 0 on highways or roads isolated from large buildings. This happens despite the researchers counting a consistent number of 40 to 50 passengers using social media on their mobile devices. The likely conclusion is that passengers use mobile data instead of pocket WiFi devices in order to access the internet. Therefore using WiFi as an passenger estimation in the context of commuting in a Public Utility Bus is not reliable.

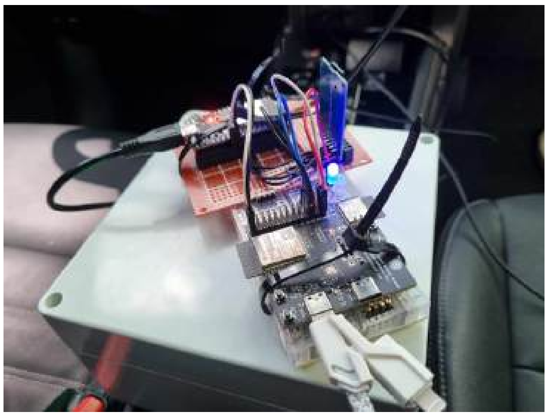


II. AQM System Field Tests

A. Car and Bike Field Test

Outdoor Node Installation on Bike

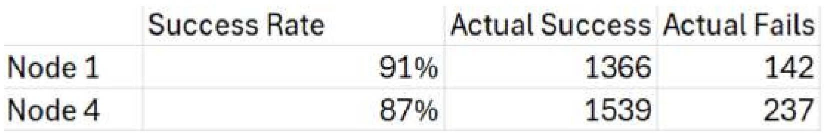
The system was field tested using a car for indoor monitoring and a bike for outdoor monitoring. The outdoor node was attached to the bicycle as shown in the figures above.


Indoor Node and Border Router Node

On the other hand, one indoor node and border router node was placed inside the car. The field test ran for 100 minutes (1 hour and 40 minutes) with the air conditioning settings inside the car set to level 2 at 23 degrees Celsius.

Powerbank Capacity After Field Test

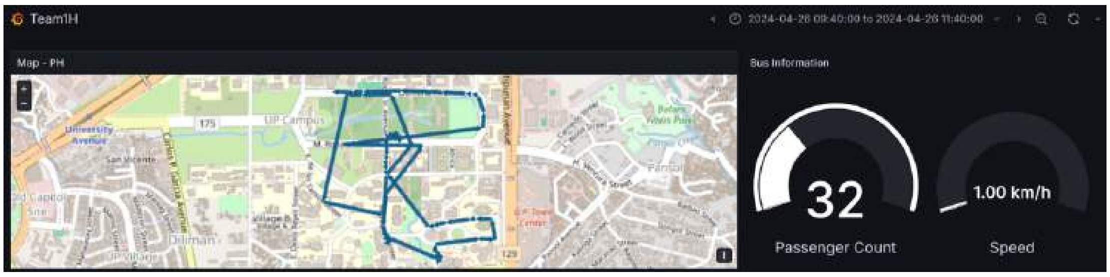
After 100 minutes of deployment, the powerbank's 10000 mAh capacity changed to around 13%. With this, the system was computed to consume around 290 mA and the estimated runtime of the system was 34.6 hours using the 10000 mAh powerbank.



	Success Rate	Actual Success	Actual Fails
Node 1	91%	1366	142
Node 4	87%	1539	237

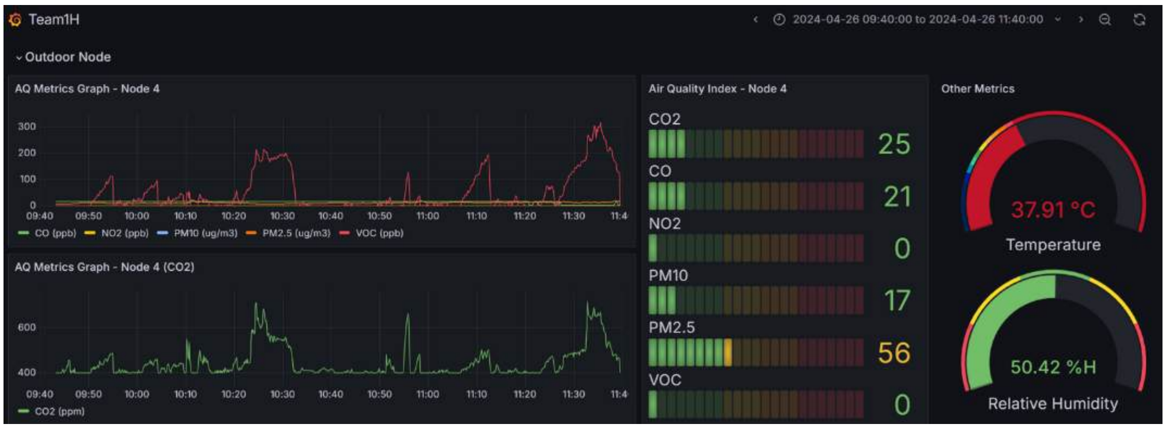
Upload Success Analysis

Shown above the success rate of upload comparing the data on the database with the logs on the SD card. The upload success rate on the indoor was found to be 91% while the outdoor node was found to be 87%.

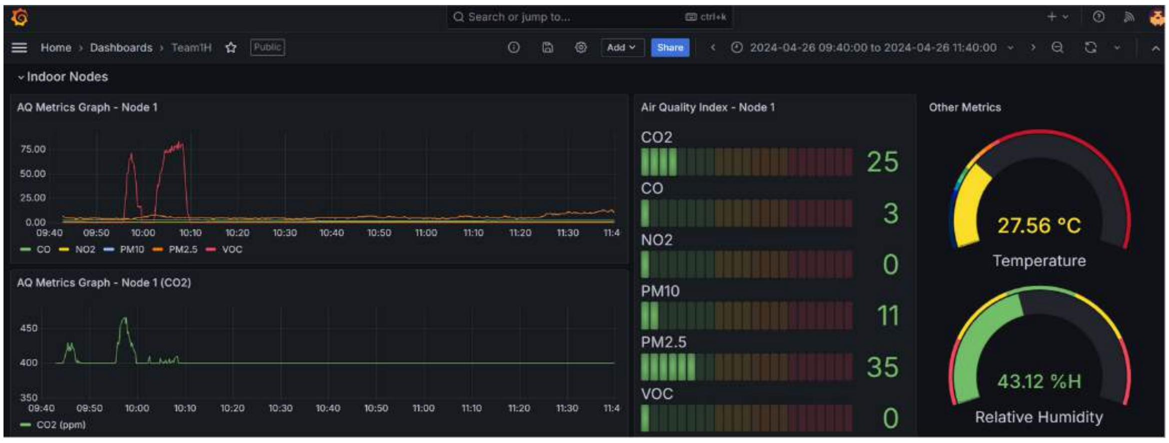


Occupancy and Vehicle Information - Grafana Dashboard

Shown above is the vehicle and occupancy information displayed on the Grafana dashboard. The GPS module struggled in getting a GPS signal mainly due to wiring issues of the actual hardware and polling rate (5s), hence the inaccuracies of the vehicle routing shown. However, both the coordinates and speed was found to be accurate when the GPS module was able to find a signal. For the occupancy counting, readings were high whenever the node was within the vicinity of buildings.

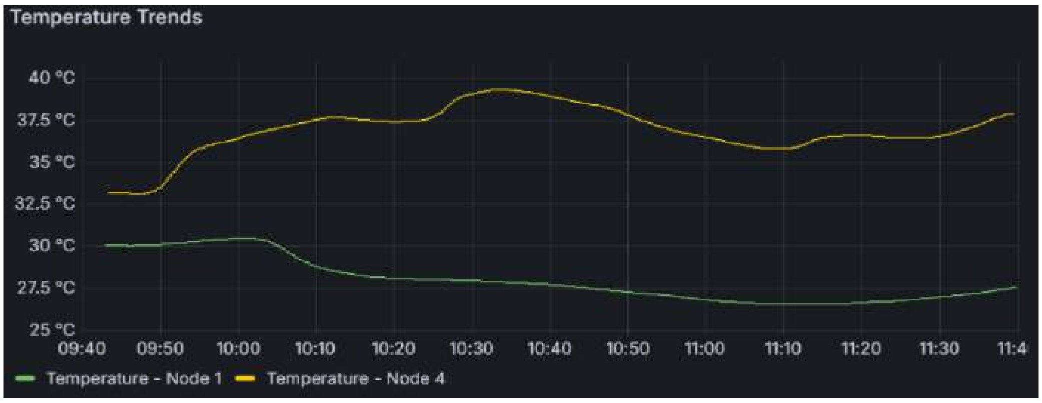


Outdoor Node Grafana Dashboard



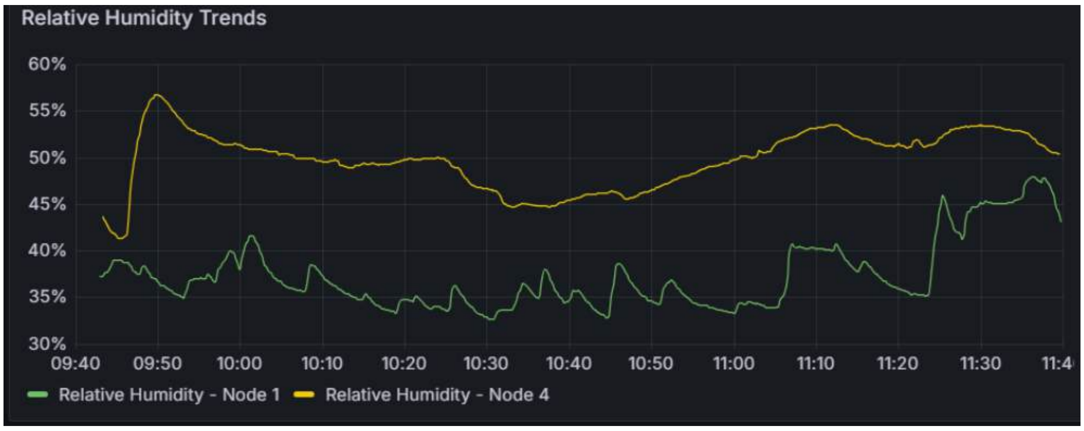
Indoor Node Grafana Dashboard

For the air quality readings, it was found that CO and NO2 readings from the MiCS-4514 were unreliable. Outdoor particulate matter (PM2.5/PM10) readings rise at the presence of other cars while indoor PM2.5/PM10 rise also at the presence of traffic when doors are opened.



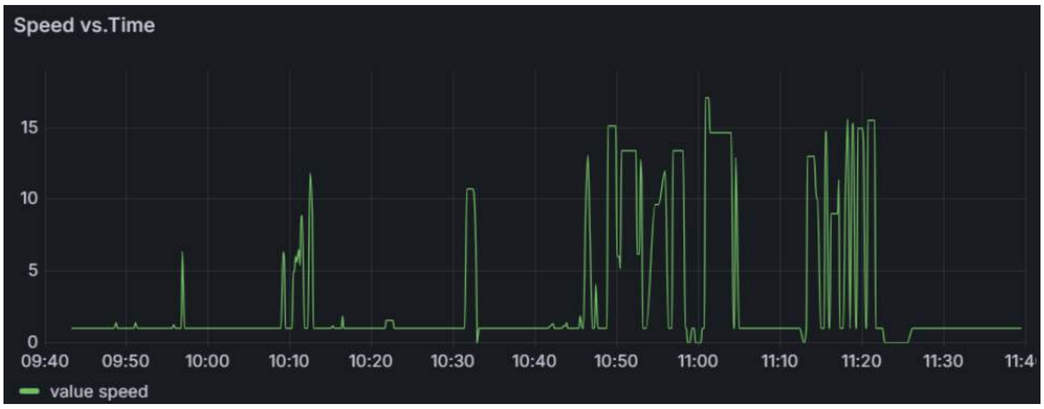
Temperature Trends - Grafana Dashboard

For the temperature trends, it is evident that the outdoor environment had higher temperature readings than that of inside the car. The outdoor temperature was found to be around 36 to 38 degrees Celsius while indoor temperature settled to around 27 to 28 degrees Celsius 20 minutes after car startup.



Relative Humidity Trends - Grafana Dashboard

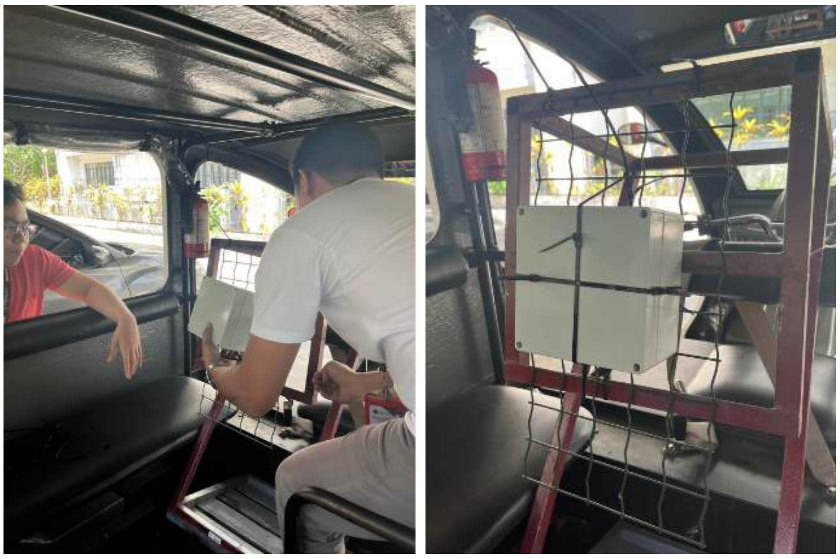
Relative humidity is higher also for the outdoor environment as opposed to that of inside the car. Outdoor relative humidity was around 50% while indoor relative humidity was found to be around 35 to 40 %.



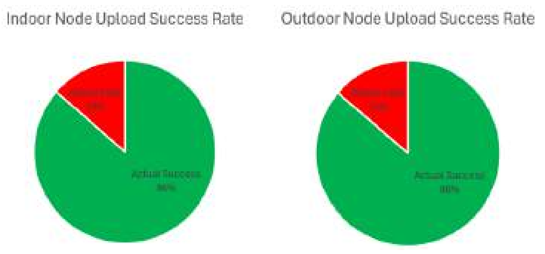
Speed Readings - Grafana Dashboard

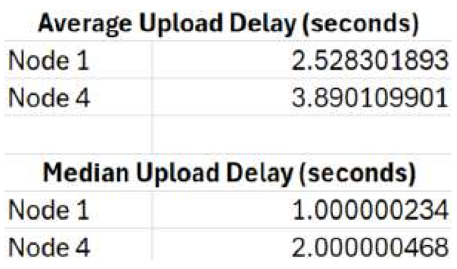
Speed readings from the GPS module were accurate when compared to the speed from the car’s dash cam. However, due to difficulty in finding and maintaining a GPS signal, speed readings show 0 whenever the GPS signal was lost.

B. Car and E-Trike Field Test


Outdoor Node Installation on E-Trike

The field test was done on the car and e-trike with the help of the EEEI admin office. The system setup was the same as the previous car and bike field test, with the outdoor node attached inside the ew-trike as shown above. Testing was done for 140 minutes (2 hours 20 minutes) with two stops in between. The first stop is to fix the loss of signal issue of the GPS module by fixing the wirings and make the antenna have a more secure placement. The second stop is to charge the e-trike due to low battery. As such, system characterization was divided into three stages.

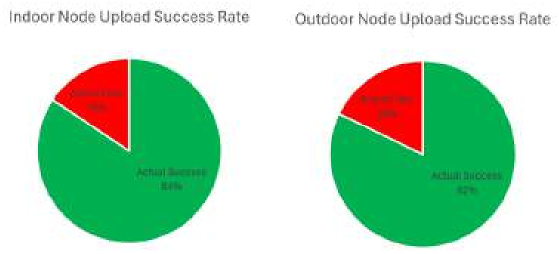


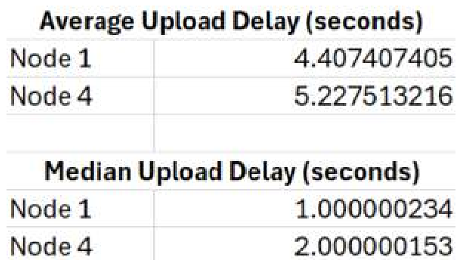


Average Upload Delay (seconds)	
Node 1	2.528301893
Node 4	3.890109901

Median Upload Delay (seconds)	
Node 1	1.000000234
Node 4	2.000000468

Stage 1 Upload Success Rate and Delay

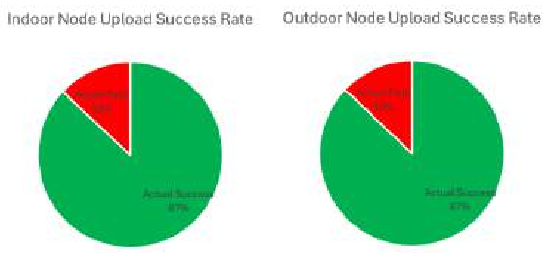




Average Upload Delay (seconds)	
Node 1	4.407407405
Node 4	5.227513216

Median Upload Delay (seconds)	
Node 1	1.000000234
Node 4	2.000000153

Stage 2 Upload Success Rate and Delay



Average Upload Delay (seconds)	
Node 1	5.421428564
Node 4	5.875739612

Median Upload Delay (seconds)	
Node 1	1.999999839
Node 4	3.000000073

Stage 3 Upload Success Rate and Delay

Shown above is the upload success rate and delay of the system. The outdoor node had lower success rate and higher delay compared to the indoor node. This may be attributed to the outdoor node having to transmit additional data from GPS such as longitude, latitude, and speed, which resulted to more data loss and longer time to process data into a JSON payload before uploading via MQTT protocol.

Stage 1

	Success Rate
Grafana	61%
Offline	50%

Stage 1

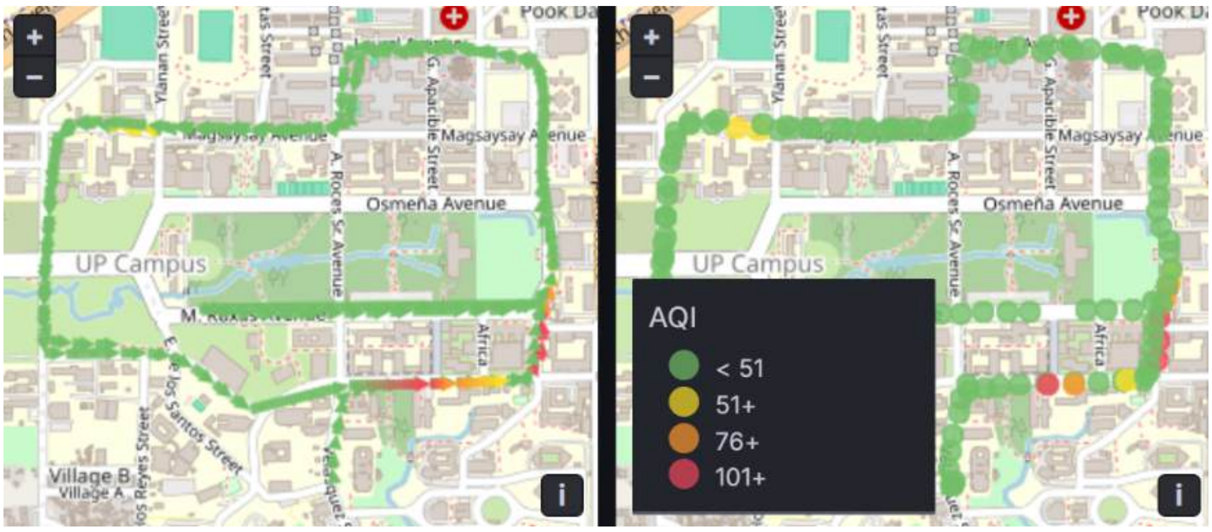
	Success Rate
Grafana	97%
Offline	96%

Stage 1

	Success Rate
Grafana	89%
Offline	87%

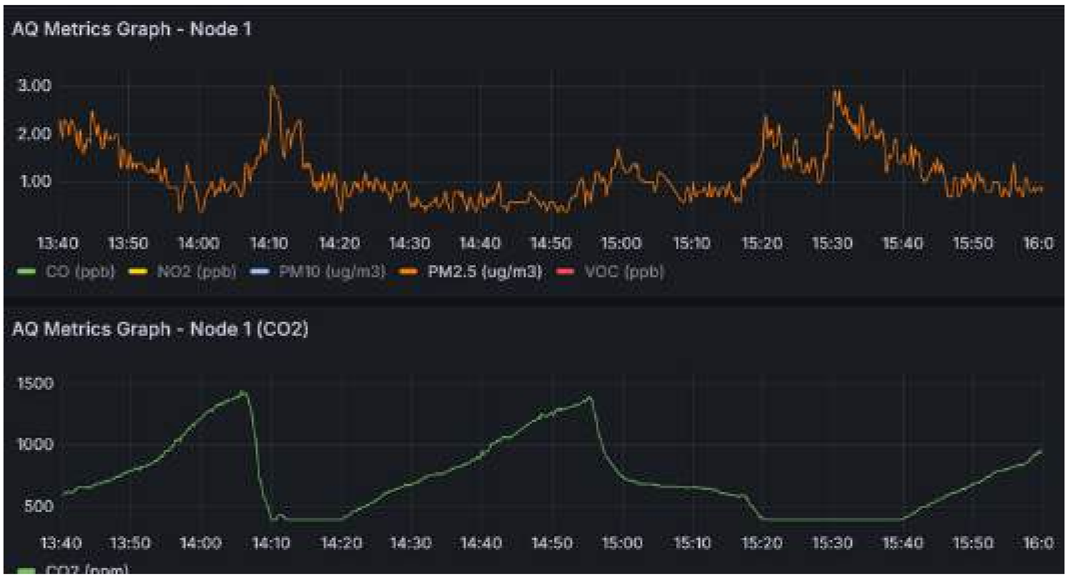
GPS Data Upload Success Rate

Shown above is the GPS data upload success rate. It was shown that stage 1 had low success due to the loss of GPS signal, prompting the researchers to halt the field test to address the issue. Upong fixing the wirings and making the antenna more secure, the success rate was much better with the data also being accurate in location-tracking.



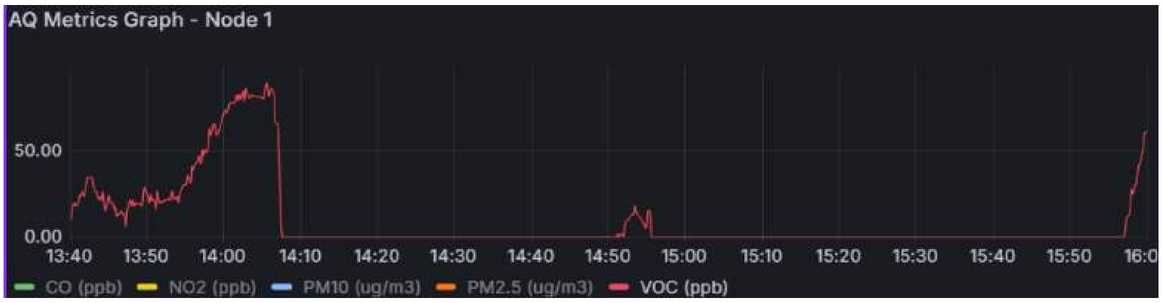
Outdoor AQI map

The figure above shows the map of the outdoor air quality index (AQI). Outdoor AQI tends to rise due to smoke coming from jeeps and cars, leading to higher outdoor particulate matter readings.



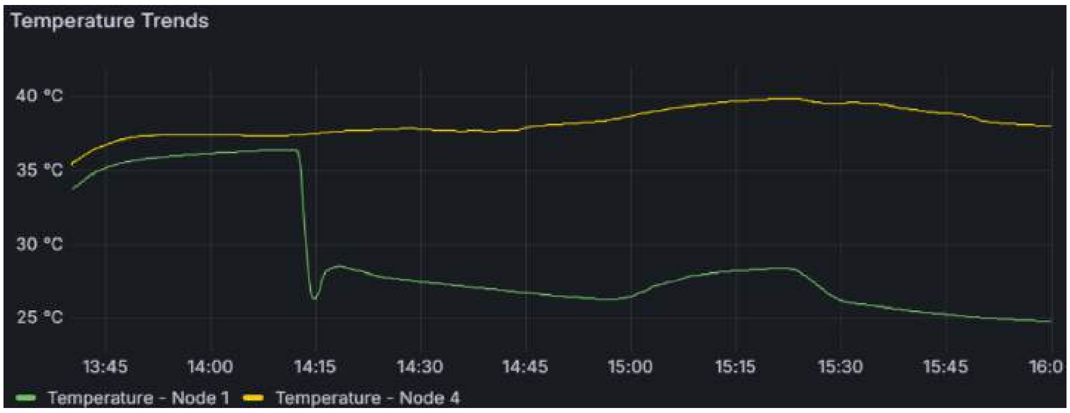
Indoor PM and CO2 Readings

It was observed that PM2.5/PM10 and CO2 readings have an inverse relationship with one another. As CO2 levels rise, PM2.5/PM10 readings tend to decrease.



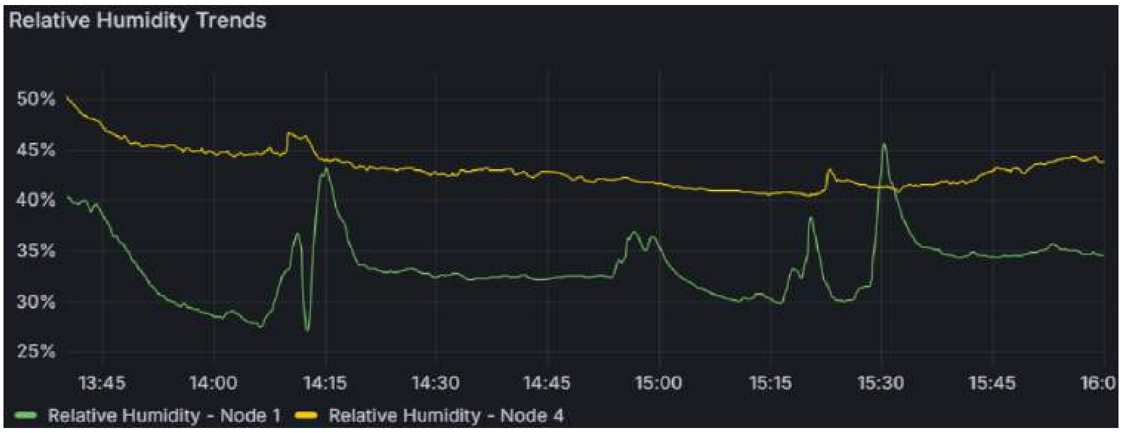
Indoor VOC Readings

It was found that VOC readings are associated with distinct smell such as food smell and alcohol sprays. VOC readings may also be attributed to when the interior plastics in the car or the leather seat were exposed to high temperatures, especially on car startup where temperature does not yet settle down.



Temperature Trend

The temperature readings were still consistent from the previous car and bike test where readings were higher outside with temperatures of around 37 to 39 degrees Celsius as opposed to indoor temperatures ranging from 25 to 27 degrees Celsius once it settled down after car start up.



Relative Humidity Trend

Relative humidity trends were also the same, with outdoor being higher at around 45% relative humidity while the car cabin was around 33% relative humidity. The sudden spikes in indoor readings were attributed to the car being stopped and turned off to fix the sensor nodes and charge the e-trike’s battery.

Appendix D

Expanded Conclusions and Recommendations

The Indoor Air Quality Monitoring System was able to acquire and display air quality and occupancy information on the website through the use of the MQTT protocol. Passengers who use the website are informed about the air quality status in all sections of the PUB. They may expect delays of up to 30 seconds, but will still be updated at least once every minute regarding the status of the PUB. The wireless mesh network allows for seamless data transfer across multiple nodes.

Trends in air pollutant concentrations inside the PUB may be attributed to numerous factors such as bus capacity and opening of doors. PM2.5 levels may be attributed to opening of doors at bus stops and heavy traffic which led to prolonged exposure to dust and smoke. CO2 and VOC levels were also observed to spike with increased passenger capacity and tighter spaces. This is especially true for CO2 levels where overcrowding occurs at different areas of the bus. In addition, VOC levels were also observed to increase with the presence of perfumes, alcohol sprays, and other strong scents.

As for the tested network protocols, out of the 3 networks, Thread and BLE-MESH performed with good to excellent data reliability occurring minimal losses with transmission from node to gateway. On the other hand, Zigbee was only able to achieve reliable data transmission when the upload period was increased to 30 seconds. However, this is primarily due to the implementation of the Zigbee protocol on the ESP platform and may be optimized with updates to the ESP-Zigbee-SDK. As for their latency, Thread and Bluetooth were able to achieve latencies lower than 10 seconds, with Thread achieving the lowest latency. Meanwhile, due to the limitations of data transmission lengths, Zigbee experienced significantly higher latency of up to 10 to 17 seconds as it needed to extract and reformat the transmitted data from the sensor nodes before uploading to the web application. Overall, the networks were able to transmit data successfully from each node to the gateway and the mesh network was able to keep uploading data to the web application even if an outage occurred in one or two nodes. In addition, reconnection of nodes is automatic with BLE-MESH being the quickest out of the three.

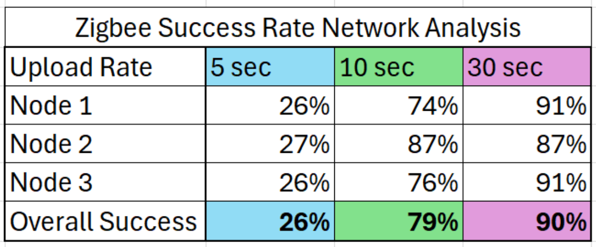
The Wi-Fi based occupancy counter is not a reliable method of measuring and estimating the number of people on the PUB. It was able to perform well in a controlled environment with accessible Wi-Fi, proving that it is capable of counting devices. However, in a public transportation setting, the concept of Wi-Fi is inapplicable to the current culture of technology where users primarily use mobile data to access the internet. It was found that the interference from approaching populated buildings resulted in overcounting. On the other hand, locations without interference lead to a count of nearly 0. This could mean that everyone on the PUB keeps their Wi-Fi off in favor of mobile data.

This may be the first real-time measurement campaign done to assess indoor air quality in PUBs in Metro Manila and the Philippines. The information from the website will help them make decisions beneficial to their health. This includes choosing where to sit for minimal exposure, or choosing whether to leave the PUB if air pollutants become too high. The proposed system also allows to gather air quality and other relevant data to urge the government for better policy-making to push for a better transportation system. This includes better maintenance on buses or other public utility vehicles, investment on more vehicles to lessen crowd congestion, and

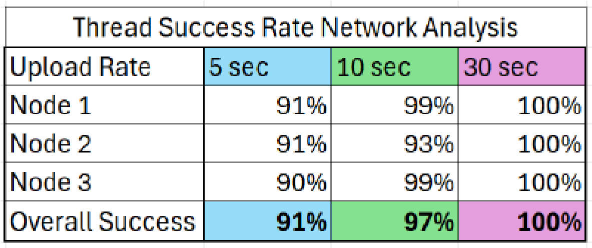
For future works, an implementation of an outdoor node for environmental monitoring would be ideal for comparison with indoor readings for the driver to conduct measures such as switching the air conditioning mode and opening of windows. A partnership or agreement with a bus company or the government to deploy is recommended for more secure installation and longer deployment period. The longer deployment would allow for more meaningful data, especially in correlation AQ data with rush hours. The AQM system can also be expanded by deploying stationary AQM systems on bus stops for further analysis and comparison.

Appendix E

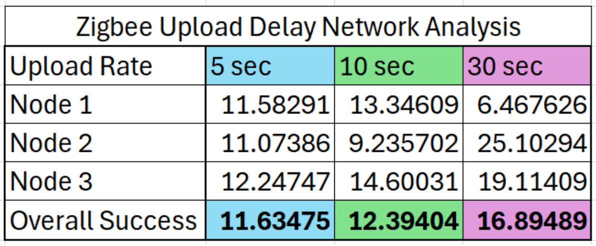
System Reliability Tests



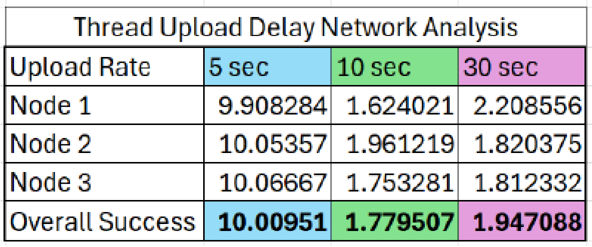
Zigbee Success Rate Network Analysis			
Upload Rate	5 sec	10 sec	30 sec
Node 1	26%	74%	91%
Node 2	27%	87%	87%
Node 3	26%	76%	91%
Overall Success	**26%**	**79%**	**90%**



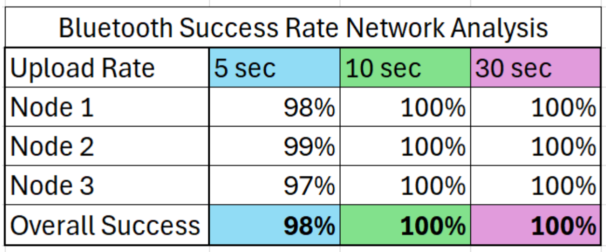
Thread Success Rate Network Analysis			
Upload Rate	5 sec	10 sec	30 sec
Node 1	91%	99%	100%
Node 2	91%	93%	100%
Node 3	90%	99%	100%
Overall Success	**91%**	**97%**	**100%**



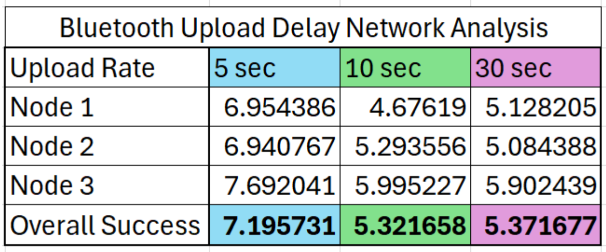
Zigbee Upload Delay Network Analysis			
Upload Rate	5 sec	10 sec	30 sec
Node 1	11.58291	13.34609	6.467626
Node 2	11.07386	9.235702	25.10294
Node 3	12.24747	14.60031	19.11409
Overall Success	**11.63475**	**12.39404**	**16.89489**



Thread Upload Delay Network Analysis			
Upload Rate	5 sec	10 sec	30 sec
Node 1	9.908284	1.624021	2.208556
Node 2	10.05357	1.961219	1.820375
Node 3	10.06667	1.753281	1.812332
Overall Success	**10.00951**	**1.779507**	**1.947088**



Bluetooth Success Rate Network Analysis			
Upload Rate	5 sec	10 sec	30 sec
Node 1	98%	100%	100%
Node 2	99%	100%	100%
Node 3	97%	100%	100%
Overall Success	**98%**	**100%**	**100%**



Bluetooth Upload Delay Network Analysis			
Upload Rate	5 sec	10 sec	30 sec
Node 1	6.954386	4.67619	5.128205
Node 2	6.940767	5.293556	5.084388
Node 3	7.692041	5.995227	5.902439
Overall Success	**7.195731**	**5.321658**	**5.371677**

Appendix F

Code for Zigbee System - Node

Template Source Code:
https://github.com/espressif/esp-zigbee-sdk/tree/main/examples/esp_zigbee_customized_devices/customized_server

```
#include <stdio.h>
#include <string.h>
#include "esp_check.h"
#include "esp_err.h"
#include "esp_log.h"
#include "nvs_flash.h"
#include "freertos/FreeRTOS.h"
#include "freertos/task.h"
#include "zcl/esp_zigbee_zcl_common.h"
#include "esp_zigbee_core.h"
#include "driver/i2c.h"
#include "driver/gpio.h"
#include <sys/unistd.h>
#include <sys/stat.h>
#include "esp_vfs_fat.h"
#include "sdmmc_cmd.h"
#include "driver/sdspi_host.h"
#include <stdlib.h>
#include "soc/soc_caps.h"
#include "esp_adc/adc_oneshot.h"
#include "esp_adc/adc_cali.h"
#include "esp_adc/adc_cali_scheme.h"
#include <math.h>
#include "driver/uart.h"

#include "led_strip.h"
#define BLINK_GPIO 8
static uint8_t s_led_state = 1;
static led_strip_handle_t led_strip;

uint16_t SENSOR_DATA_CLUSTER_ID = 0xFFF1;
uint16_t SENSOR_DATA_ATTR_ID = 0;
uint16_t SENSOR_DATA_ATTR_ID_2 = 1;
uint16_t SENSOR_DATA_ATTR_ID_3 = 2;
char Current_Date_Time[20];
int latitude_decimal;
int longitude_decimal;
SemaphoreHandle_t semaphore;
TaskHandle_t sen_handle, mics_handle, dummy, sgp_handle, finish_handle, uart_handle, zb_handle;

static const char *TAG_SDCARD = "SD_CARD";
int heated = 0;
#define TASK_COUNT 3

//GPIO Pins for LED
#define RED_LED 22
#define ORANGE_LED 23
#define YELLOW_LED 27
#define GREEN_LED 26

//I2C0 for SGP30
#define I2C_PORT_0 0
#define I2C0_SCL 10
#define I2C0_SDA 11

//I2C1 for SEN55 and MiCS-4514
#define I2C_PORT_1 1
#define I2C1_SCL 12
#define I2C1_SDA 8

//For Sensor I2C Addresses
#define SEN55_ADDR 0x69
#define SGP30_ADDR 0x58

//General I2C parameters
#define I2C_FREQUENCY 100000

//Global Variables for Sensor Readings
static float pm25, pm10, rh, temp, co, no2, co2, voc;
int no2_adc, co_adc;
float pm25_score, pm10_score, co_score, no2_score, co2_score, voc_score, iaq;
float speed_kmh=0;
char* location[4];
char* velocity[10];
double lat_deg = 0;
double long_deg = 0;
```

```
char* speed_str = "0";

//SPI for SD Card Module
#define PIN_NUM_MISO  0
#define PIN_NUM_MOSI  5
#define PIN_NUM_CLK   4
#define PIN_NUM_CS    1
#define MAX_SIZE 220
#define MOUNT_POINT "/sdcard"
int first_write = 1;

//ADC1 for MiCS-4514
adc_channel_t ADC1_CH1_RED = ADC_CHANNEL_1; // For RED ADC Readings
adc_channel_t ADC1_CH2_NOX = ADC_CHANNEL_2; // For NOX ADC Readings
#define PREHEAT_PIN 13 // Pin to pre-heat the module

//UART1 for Ublox Neo 6M V2
#define TX_pin 26
#define RX_pin 27

static const char *TAG = "ESP_ZB_CLIENT_4";
char data1[80] = "K150,1000.00,150,1000.00,85.00,100.00,150,60000.00,150,10000.00,150,1000.00,150,10.00";
char data2[80] = "K150,10.00,00.000000,000.000000,00.00,2020-01-01T12:59:59";
char data3[50] = "00.000000,000.000000,00.00,";
//https://github.com/espressif/esp-zigbee-sdk/issues/244

/* Zigbee configuration */
/* T H I S     I S      F O R      E N D      D E V I C E S*/
#define INSTALLCODE_POLICY_ENABLE       false             /* enable the install code policy for security */
#define ED_AGING_TIMEOUT                ESP_ZB_ED_AGING_TIMEOUT_64MIN
#define ED_KEEP_ALIVE                   3000     /* 3000 millisecond */
#define ZB_CLIENT_ENDPOINT_1                     1          /* esp device endpoint */
#define ZB_CLIENT_ENDPOINT_2                     2
#define ZB_CLIENT_ENDPOINT_3                     3
#define ZB_CLIENT_ENDPOINT_4                     4
#define ESP_ZB_PRIMARY_CHANNEL_MASK     (1l << 11)  /* Zigbee primary channel mask use in the example */
#define ESP_ZB_SECONDARY_CHANNEL_MASK   (1l << 13)  /* Zigbee primary channel mask use in the example */

uint16_t GATEWAY_TIME_CLUSTER_ID = 0xFFF2;
uint8_t GATEWAY_TIME_ATTR_ID = 3;

#define ESP_ZB_ZED_CONFIG()                                       \
    {                                                             \
        .esp_zb_role = ESP_ZB_DEVICE_TYPE_ED,                     \
        .install_code_policy = INSTALLCODE_POLICY_ENABLE,         \
        .nwk_cfg.zed_cfg = {                                      \
            .ed_timeout = ED_AGING_TIMEOUT,                       \
            .keep_alive = ED_KEEP_ALIVE,                          \
        },                                                        \
    }
#define ESP_ZB_DEFAULT_RADIO_CONFIG()                             \
    {                                                             \
        .radio_mode = RADIO_MODE_NATIVE,                          \
    }
#define ESP_ZB_DEFAULT_HOST_CONFIG()                              \
    {                                                             \
        .host_connection_mode = HOST_CONNECTION_MODE_NONE,        \
    }

typedef struct zb_coordinator_s {
    esp_zb_ieee_addr_t ieee_addr;
    uint8_t  endpoint;
    uint16_t short_addr;
} zb_coordinator_t;
zb_coordinator_t zb_coord_info;

typedef struct zdo_info_ctx_s {
    uint8_t endpoint;
    uint16_t short_addr;
} zdo_info_user_ctx_t;

void ReportAttribute (uint16_t attr_id){
    //DELIERY DETAILS
    esp_zb_zcl_report_attr_cmd_t report_here;
    report_here.zcl_basic_cmd.dst_addr_u.addr_short = zb_coord_info.short_addr;
    report_here.zcl_basic_cmd.dst_endpoint = zb_coord_info.endpoint;
    report_here.zcl_basic_cmd.src_endpoint = ZB_CLIENT_ENDPOINT_4;
    report_here.address_mode = ESP_ZB_APS_ADDR_MODE_16_ENDP_PRESENT;
    report_here.clusterID = SENSOR_DATA_CLUSTER_ID;
    report_here.cluster_role = ESP_ZB_ZCL_CLUSTER_SERVER_ROLE; //SERVER ACTUALLY MEANS I HAVE THE DATA,
    NOT CLIENT WILL SEND DATA
    report_here.attributeID = attr_id;
    esp_zb_zcl_attr_t *value_r = esp_zb_zcl_get_attribute(ZB_CLIENT_ENDPOINT_4, SENSOR_DATA_CLUSTER_ID,
    ESP_ZB_ZCL_CLUSTER_SERVER_ROLE, attr_id);
    if (attr_id == SENSOR_DATA_ATTR_ID){
            memcpy(value_r->data_p, &data1, sizeof(data1));
    }
    else if (attr_id == SENSOR_DATA_ATTR_ID_2){
            memcpy(value_r->data_p, &data2, sizeof(data2));
     }
     else if (attr_id == SENSOR_DATA_ATTR_ID_3){
            memcpy(value_r->data_p, &data3, sizeof(data3));
     }
     esp_zb_zcl_report_attr_cmd_req(&report_here);
}

void ReadAttribute (void){
     esp_zb_zcl_read_attr_cmd_t read_here;
     read_here.zcl_basic_cmd.dst_addr_u.addr_short = zb_coord_info.short_addr;
     read_here.zcl_basic_cmd.dst_endpoint = zb_coord_info.endpoint;
     read_here.zcl_basic_cmd.src_endpoint = ZB_CLIENT_ENDPOINT_4;
     read_here.address_mode = ESP_ZB_APS_ADDR_MODE_16_ENDP_PRESENT;
     read_here.clusterID = GATEWAY_TIME_CLUSTER_ID;
     read_here.attributeID = GATEWAY_TIME_ATTR_ID;
     esp_zb_zcl_read_attr_cmd_req(&read_here);
}

static void blink_led(void)
{
     /* Set the LED pixel using RGB from 0 (0%) to 255 (100%) for each color */
     led_strip_set_pixel(led_strip, 0, 255, 16, 16);
     /* Refresh the strip to send data */
     led_strip_refresh(led_strip);
}

static void configure_led(void)
{
    ESP_LOGI(TAG, "Example configured to blink addressable LED!");
    /* LED strip initialization with the GPIO and pixels number*/
    led_strip_config_t strip_config = {
        .strip_gpio_num = BLINK_GPIO,
        .max_leds = 1, // at least one LED on board
    };
    led_strip_rmt_config_t rmt_config = {
        .resolution_hz = 10 * 1000 * 1000, // 10MHz
    };
    ESP_ERROR_CHECK(led_strip_new_rmt_device(&strip_config, &rmt_config, &led_strip));
    /* Set all LED off to clear all pixels */
    led_strip_clear(led_strip);
}

//SD Card R/W Functions
static esp_err_t sd_card_write_file(const char *path, char *data)
{
    ESP_LOGI(TAG_SDCARD, "Opening file %s", path);
    FILE *f = fopen(path, "a+");
    if (f == NULL) {
        ESP_LOGE(TAG_SDCARD, "Failed to open file for writing");
        return ESP_FAIL;
    }
    fprintf(f, data);
    fclose(f);
    ESP_LOGI(TAG_SDCARD, "File written");

    return ESP_OK;
}
static esp_err_t sd_card_read_file(const char *path)
{
    ESP_LOGI(TAG_SDCARD, "Reading file %s", path);
    FILE *f = fopen(path, "r");
    if (f == NULL) {
        ESP_LOGE(TAG_SDCARD, "Failed to open file for reading");
        return ESP_FAIL;
    }
    char line[MAX_SIZE];
    fgets(line, sizeof(line), f);
    fclose(f);

    // strip newline
    char *pos = strchr(line, '\n');
    if (pos) {
        *pos = '\0';
    }
    ESP_LOGI(TAG_SDCARD, "Read from file: '%s'", line);

    return ESP_OK;
}
//SD Card Main Functions
void sd_card_write(void){
    esp_err_t ret;
    // Options for mounting the filesystem.
    // If format_if_mount_failed is set to true, SD card will be partitioned and
    // formatted in case when mounting fails.
    esp_vfs_fat_sdmmc_mount_config_t mount_config = {
#ifdef CONFIG_EXAMPLE_FORMAT_IF_MOUNT_FAILED
        .format_if_mount_failed = true,
#else
        .format_if_mount_failed = false,
#endif // EXAMPLE_FORMAT_IF_MOUNT_FAILED
        .max_files = 5,
        .allocation_unit_size = 16 * 1024
    };
    sdmmc_card_t *card;
    const char mount_point[] = MOUNT_POINT;
    ESP_LOGI(TAG_SDCARD, "Initializing SD card");

    // Use settings defined above to initialize SD card and mount FAT filesystem.
    ESP_LOGI(TAG_SDCARD, "Using SPI peripheral");

    // For setting a SD card frequency, use host.max_freq_khz (range 400kHz - 20MHz for SDSPI)
    sdmmc_host_t host = SDSPI_HOST_DEFAULT();

    spi_bus_config_t bus_cfg = {
        .mosi_io_num = PIN_NUM_MOSI,
        .miso_io_num = PIN_NUM_MISO,
        .sclk_io_num = PIN_NUM_CLK,
        .quadwp_io_num = -1,
        .quadhd_io_num = -1,
        .max_transfer_sz = 4000,
    };
    ret = spi_bus_initialize(host.slot, &bus_cfg, SPI_DMA_CH_AUTO);
    if (ret != ESP_OK) {
        ESP_LOGE(TAG_SDCARD, "Failed to initialize bus.");
        return;
    }

    sdspi_device_config_t slot_config = SDSPI_DEVICE_CONFIG_DEFAULT(); //SD Card Module has no CS and WP
     i/o
    slot_config.gpio_cs = PIN_NUM_CS;
    slot_config.host_id = host.slot;

    ESP_LOGI(TAG_SDCARD, "Mounting to SD Card...");
    ret = esp_vfs_fat_sdspi_mount(mount_point, &host, &slot_config, &mount_config, &card);
    if (ret != ESP_OK) {
        if (ret == ESP_FAIL) {
            ESP_LOGE(TAG_SDCARD, "Failed to mount");
        } else {
            ESP_LOGE(TAG_SDCARD, "Failed to initialize the card (%s).", esp_err_to_name(ret));
        }
        return;
    }
    ESP_LOGI(TAG_SDCARD, "SD Card mounted");

    // Card has been initialized, print its properties
    //sdmmc_card_print_info(stdout, card);

    // Directory for "aq_log.txt"
    const char *aq_log_file = MOUNT_POINT"/aq_log.csv";
    char log[MAX_SIZE];

    if (first_write == 1){
     //Write Headers
     snprintf(log, MAX_SIZE, "Time, CO (ppb), CO Index, CO2 (ppm), CO2 Index, NO2 (ppb), NO2 Index, PM2.5
     (ug/m3), PM2.5 Index, PM10 (ug/m3), PM10 Index, VOC (ppb), VOC Index, Temperature (C), RH (%s),
     Latitude, Longitude, Speed (km/h)\n", "%%");
     ret = sd_card_write_file(aq_log_file, log);
        if (ret != ESP_OK) {
            return;
        }
     first_write = 0;
    }
    // Set up data to be logged
    snprintf(log, MAX_SIZE, "%s, %.2f, %.0f, %.2f, %.0f, %.2f, %.0f, %.2f, %.0f, %.2f, %.0f, %.2f, %.0f,
    %.2f, %.2f, %.7f, %.7f, %.2f\n", Current_Date_Time, co, co_score, co2, co2_score, no2, no2_score,
    pm25, pm25_score, pm10, pm10_score, voc, voc_score, temp, rh, lat_deg, long_deg, speed_kmh);
    //printf("%s\n", log);
    // Write data log to aq_log.txt
    ret = sd_card_write_file(aq_log_file, log);
    if (ret != ESP_OK) {
        return;
    }
    //Open file for reading
/*    ret = sd_card_read_file(aq_log_file);
    if (ret != ESP_OK) {
        return;
    }*/

    // All done, unmount partition and disable SPI peripheral
    esp_vfs_fat_sdcard_unmount(mount_point, card);
    ESP_LOGI(TAG_SDCARD, "Card unmounted");

    //deinitialize the bus after all devices are removed
    spi_bus_free(host.slot);
}
```

```
//Functions for LED Alert System
void led_alert_init (void){
     gpio_reset_pin(RED_LED);
     gpio_set_direction(RED_LED, GPIO_MODE_OUTPUT);
     gpio_reset_pin(ORANGE_LED);
     gpio_set_direction(ORANGE_LED, GPIO_MODE_OUTPUT);
     gpio_reset_pin(YELLOW_LED);
     gpio_set_direction(YELLOW_LED, GPIO_MODE_OUTPUT);
     gpio_reset_pin(GREEN_LED);
     gpio_set_direction(GREEN_LED, GPIO_MODE_OUTPUT);
     gpio_set_level(RED_LED, 1);
     gpio_set_level(ORANGE_LED, 1);
     gpio_set_level(YELLOW_LED, 1);
     gpio_set_level(GREEN_LED, 1);
}
void update_led_alert (float pm25, float pm10, float rh, float temp, float co, float no2, float co2, float
     voc){
     //Condition for RED Level AQI
     if (pm25 > 30 || pm10 > 100 || co > 70 || co2 > 1500 || no2 > 250 || voc > 800){
              gpio_set_level(RED_LED, 1);
              gpio_set_level(ORANGE_LED, 0);
              gpio_set_level(YELLOW_LED, 0);
              gpio_set_level(GREEN_LED, 0);

     }
     //Condition for ORANGE Level AQI
     else if (pm25 > 20 || pm10 > 75 || co > 50 || co2 > 1150 || no2 > 175 || voc > 600){
              gpio_set_level(RED_LED, 0);
              gpio_set_level(ORANGE_LED, 1);
              gpio_set_level(YELLOW_LED, 0);
              gpio_set_level(GREEN_LED, 0);

     }
     //Condition for YELLOW Level AQI
     else if (pm25 > 15 || pm10 > 50 || co > 35 || co2 > 800 || no2 > 100 || voc > 400){
              gpio_set_level(RED_LED, 0);
              gpio_set_level(ORANGE_LED, 0);
              gpio_set_level(YELLOW_LED, 1);
              gpio_set_level(GREEN_LED, 0);

     }
     //Condition for GREEN Level AQI
     else if (pm25 > 0 || pm10 > 0 || co > 0 || co2 > 0 || no2 > 0 || voc > 0){
              gpio_set_level(RED_LED, 0);
              gpio_set_level(ORANGE_LED, 0);
              gpio_set_level(YELLOW_LED, 0);
              gpio_set_level(GREEN_LED, 1);

     }
     //negative values are encountered, possible code error
     else{
              gpio_set_level(RED_LED, 0.4);
              gpio_set_level(ORANGE_LED, 0.4);
              gpio_set_level(YELLOW_LED, 0.4);
              gpio_set_level(GREEN_LED, 0.4);
     }
}
void iaq_score(float pm25, float pm10, float co, float no2, float co2, float voc){
     float index[4][2] = {{0,50}, {51,75}, {76,100}, {101,150}};
     float co_bp[4][2] = {{0,35}, {36,50}, {51,70}, {70,1000}};
     float co2_bp[4][2] = {{0,800}, {801,1150}, {1151,1500}, {1501,60000}};
     float pm25_bp[4][2] = {{0,15}, {16,20}, {21,30}, {31,1000}};
     float pm10_bp[4][2] = {{0,50}, {51,75}, {76,100}, {101,1000}};
     float no2_bp[4][2] = {{0,100}, {101,175}, {176,250}, {251,10000}};
     float voc_bp[4][2] = {{0,400}, {401,600}, {601,800}, {801,60000}};
     //PM2.5
     for (int j=3; j>=0; j--){
              if(pm25 >= pm25_bp[j][0]){
                       pm25_score =
     (((index[j][1]-index[j][0])/(pm25_bp[j][1]-pm25_bp[j][0]))*(pm25-pm25_bp[j][0]))+(index[j][0]);
                       break;
              }
     }
     //PM10
     for (int j=3; j>=0; j--){
              if(pm10 >= pm10_bp[j][0]){
                       pm10_score =
     (((index[j][1]-index[j][0])/(pm10_bp[j][1]-pm10_bp[j][0]))*(pm10-pm10_bp[j][0]))+(index[j][0]);
                       break;
              }
     }
     //CO
     for (int j=3; j>=0; j--){
              if(co >= co_bp[j][0]){
                       co_score =
     (((index[j][1]-index[j][0])/(co_bp[j][1]-co_bp[j][0]))*(co-co_bp[j][0]))+(index[j][0]);
                       break;
```

```
            }
        }
        //NO2
        for (int j=3; j>=0; j--){
                if(no2 >= no2_bp[j][0]){
                        no2_score =
        (((index[j][1]-index[j][0])/(no2_bp[j][1]-no2_bp[j][0]))*(no2-no2_bp[j][0]))+(index[j][0]);
                        break;
                }
        }
        //CO2
        for (int j=3; j>=0; j--){
                if(co2 >= co2_bp[j][0]){
                        co2_score =
        (((index[j][1]-index[j][0])/(co2_bp[j][1]-co2_bp[j][0]))*(co2-co2_bp[j][0]))+(index[j][0]);
                        break;
                }
        }
        //VOC
        for (int j=3; j>=0; j--){
                if(voc >= voc_bp[j][0]){
                        voc_score =
        (((index[j][1]-index[j][0])/(voc_bp[j][1]-voc_bp[j][0]))*(voc-voc_bp[j][0]))+(index[j][0]);
                        break;
                }
        }

}
void driver_alert_init (void){
        gpio_reset_pin(RECIRC_LED);
        gpio_set_direction(RECIRC_LED, GPIO_MODE_OUTPUT);
        gpio_reset_pin(INTAKE_LED);
        gpio_set_direction(INTAKE_LED, GPIO_MODE_OUTPUT);
        gpio_set_level(INTAKE_LED, 0);
        gpio_set_level(RECIRC_LED, 0);
}
void driver_alert(float pm25_in, float pm10_in, float co_in, float no2_in, float co2_in, float voc_in,
                                float pm25_out, float pm10_out, float co_out, float no2_out, float
        co2_out, float voc_out){
        if (co_in > co_out || no2_in > no2_out || co2_in > co2_out || voc_in > voc_out){
                gpio_set_level(INTAKE_LED, 1); // Accumulated gas concentrations (Orange level)
                gpio_set_level(RECIRC_LED, 0);
        }
        else{
                gpio_set_level(INTAKE_LED, 0);
                gpio_set_level(RECIRC_LED, 1);
        }
}

//Warm-Up for MiCS-4514 for 3 minutes
void sensor_init(int *heated_var){
        gpio_reset_pin(PREHEAT_PIN);
        gpio_set_direction(PREHEAT_PIN, GPIO_MODE_OUTPUT);
        gpio_set_level(PREHEAT_PIN, 1);

        int i2c_master_port0 = I2C_PORT_0;
       i2c_config_t conf0 = {
           .mode = I2C_MODE_MASTER,
           .sda_io_num = I2C0_SDA,
           .scl_io_num = I2C0_SCL,
           .sda_pullup_en = GPIO_PULLUP_ENABLE,
           .scl_pullup_en = GPIO_PULLUP_ENABLE,
           .master.clk_speed = I2C_FREQUENCY,
       };
       i2c_param_config(i2c_master_port0, &conf0);
       i2c_driver_install(i2c_master_port0, conf0.mode, 0, 0, 0);
       uint8_t SGP30_INIT[2] = {0x20, 0x03}; //Initialize SGP30
       i2c_master_write_to_device(i2c_master_port0, SGP30_ADDR, SGP30_INIT, 2, 1000/portTICK_PERIOD_MS);
        if(heated_var == 0){
                printf("Preheating MiCS-4514...\n");
                vTaskDelay(180000/portTICK_PERIOD_MS); // Preaheat MiCS for 3 minutes
        }
        else{
                printf("Waiting for SGP30...\n");
                vTaskDelay(1000/portTICK_PERIOD_MS);
        }
}
//For mapping MiCS-4514 raw readings
void process_mics4514(int D_co, int D_no2) {
        //Source https://myscope.net/auswertung-der-airpi-gas-sensoren/
        //For CO readings
        printf("D_OUTS: %d, %d\n", D_co, D_no2);
        double Vo_co = (D_co-2144) * (1.944)/(4081-2144);
        double Vo_no2 = (D_no2-2144) * (1.944)/(4081-2144);
        printf("VOLTAGE: %.5f  %.5f\n", Vo_co, Vo_no2);
        double Rs_co = (5-Vo_co) / (Vo_co/820);
        double Rs_no2 = (5-Vo_no2) / (Vo_no2/820);
        printf("RESISTANCE: %.2f  %.2f\n", Rs_co, Rs_no2);
```

```
    double R0_co = 3768.965; // FOR CALIBRATION (4.00 * R0 = Rs @ 0.8 ppm CO)
    co = pow(10, -1.1859 * (log(Rs_co/R0_co) / M_LN10) + 0.6201); //Curve Fitting Equation from Source
    //For NO2 readings
    double R0_no2 = 1440000; // FOR CALIBRATION (0.05*R0 = Rs @ 0.01 ppm NO2)
    no2 = pow(10, 0.9682 * (log(Rs_no2/R0_no2) / M_LN10) - 0.8108); //Curve Fitting Equation from Source
    //printf("CO: %.2f, NO2: %.2f\n", co, no2);
}
//RTOS Task for MiCS-4514 @ ADC1 Channel 1 & 2
void ADC1_Data_Task(void *params){
    //Initialize ADC1 Peripheral
    adc_oneshot_unit_handle_t adc1_handle;
    adc_oneshot_unit_init_cfg_t init_config1 = {
             .unit_id = ADC_UNIT_1,
                  .clk_src = ADC_DIGI_CLK_SRC_RC_FAST,
                  .ulp_mode = ADC_ULP_MODE_DISABLE,
    };
    printf("ADC INIT\n");
    ESP_ERROR_CHECK(adc_oneshot_new_unit(&init_config1, &adc1_handle));

    //Configure the 2 ADC Channels
    adc_oneshot_chan_cfg_t config1 = {
              .atten = ADC_ATTEN_DB_11,
    };
    adc_oneshot_chan_cfg_t config2 = {
              .atten = ADC_ATTEN_DB_11,
    };
    ESP_ERROR_CHECK(adc_oneshot_config_channel(adc1_handle, ADC1_CH1_RED, &config1)); // For RED analog
    readings
    vTaskDelay(100/portTICK_PERIOD_MS);
    ESP_ERROR_CHECK(adc_oneshot_config_channel(adc1_handle, ADC1_CH2_NOX, &config2)); // For NOX analog
    readings
    vTaskDelay(100/portTICK_PERIOD_MS);
    while(1){
        ESP_ERROR_CHECK(adc_oneshot_read(adc1_handle, ADC1_CH1_RED, &co_adc)); // Get readings from RED
        vTaskDelay(10/portTICK_PERIOD_MS);
        ESP_ERROR_CHECK(adc_oneshot_read(adc1_handle, ADC1_CH2_NOX, &no2_adc)); // Get readings from NOX
        process_mics4514(co_adc, no2_adc);
        //printf("%d", uxTaskGetStackHighWaterMark(NULL));
        //printf("CO: %.2f, NO2: %.2f\n", co, no2);
        xSemaphoreGive(semaphore);
        vTaskSuspend(mics_handle);
        vTaskDelay(1000/portTICK_PERIOD_MS);
    }
}
//Process SEN55 Data
void process_sen55(uint8_t data[24]){
    //Raw Bit Data
    //PM2.5 Bits
    uint16_t pm25_bits = data[3] << 8; //First 8 bits
                   pm25_bits |= data[4]; //Add last 8 bits
    //PM10 Bits
    uint16_t pm10_bits = data[9] << 8; //First 8 bits
                   pm10_bits |= data[10]; //Add last 8 bits
    //Humidity Bits
    int16_t rh_bits = data[12]; //First 8 bits
                  rh_bits <<= 8; //Shift by 8 bits
                  rh_bits |= data[13]; //Add last 8 bits
    //Temperature Bits
     int16_t temp_bits = data[15]; //First 8 bits
                   temp_bits <<= 8; //Shift by 8 bits
                   temp_bits |= data[16]; //Add last 8 bits


    //Divide by Scaling Factors
    float pm25_raw = pm25_bits;
    float pm10_raw = pm10_bits;
    float rh_raw = rh_bits;
    float temp_raw = temp_bits;
    pm25 = pm25_raw /10;
    pm10 = pm10_raw /10;
    rh = rh_raw / 100;
    temp = temp_raw / 200;
    //Node 2
    pm25 = 1.7992841401851236*pm25 + (-7.9867018290678296);
    pm10 = 2.157191907471857*pm10 + (-11.60142985269697);
    temp = 0.6681571289773972*temp + (8.007242383192157);
    rh = 0.8675675959276415*rh + (-6.130361505580758);
}

    /*
    Calibration from Linear Regression (AirGradient)
    N4 PM25 Coeff: 1.4859836383305598; Intercept: -8.60845666715265
    N4 PM10 Coeff: 1.5271189038734376; Intercept: -8.63324447482231
    N4 Temperature Coeff: 0.6215852043191423; Intercept: 10.288192894279575
    N4 Rel. Humidity Coeff: 0.8266878818975784; Intercept: -6.974060832750652
     */

/*  float pm25_uncalib, pm10_uncalib, temp_uncalib, rh_uncalib;
```

```
    pm25_uncalib = pm25_raw /10;
    pm10_uncalib = pm10_raw /10;
    rh_uncalib = rh_raw / 100;
    temp_uncalib = temp_raw / 200;
    pm25 = (pm25_uncalib*1.4859836383305598) - 8.60845666715265;
    pm10 = (pm10_uncalib*1.5271189038734376) - 8.63324447482231;
    temp = (temp_uncalib*0.6215852043191423) + 10.288192894279575;
    rh = (rh_uncalib*0.8266878818975784) - 6.974060832750652;*/

//SEN55 and SGP30 CRC Calculation
uint8_t CalcCrc(uint8_t data[2]) {
    uint8_t crc = 0xFF;
    for(int i = 0; i < 2; i++) {
            crc ^= data[i];
            for(uint8_t bit = 8; bit > 0; --bit) {
                    if(crc & 0x80) {
                            crc = (crc << 1) ^ 0x31u;
                    }
                    else {
                            crc = (crc << 1);
                    }
            }
     }
     return crc;
}
//RTOS Task for SEN55 @ I2C1
void I2C1_Data_Task(void *params){
    //SEN55 Initialize
    i2c_config_t conf1 = {
                    .mode = I2C_MODE_MASTER,
            .sda_io_num = I2C1_SDA,
            .scl_io_num = I2C1_SCL,
            .sda_pullup_en = GPIO_PULLUP_ENABLE,
            .scl_pullup_en = GPIO_PULLUP_ENABLE,
            .master.clk_speed = I2C_FREQUENCY,
    };
    i2c_param_config(I2C_PORT_1, &conf1);
    i2c_driver_install(I2C_PORT_1, conf1.mode, 0, 0, 0);
    uint8_t SEN55_MEAS_INIT[2] = {0x00,0x21}; //Start Measurement Mode
   uint8_t sen55_data[24];
    uint8_t SEN55_READ_DATA[2] = {0x03,0xC4}; //Get Measured Values Command
    ESP_ERROR_CHECK(i2c_master_write_to_device(I2C_PORT_1, SEN55_ADDR, SEN55_MEAS_INIT, 2,
    1000/portTICK_PERIOD_MS)); //Measurement Mode
    vTaskDelay(10/portTICK_PERIOD_MS);
   while (1) {
    //For SEN55 (Starts at Idle Mode)
    ESP_ERROR_CHECK(i2c_master_write_to_device(I2C_PORT_1, SEN55_ADDR, SEN55_READ_DATA, 2,
    1000/portTICK_PERIOD_MS)); //Send Get-Measurement Command
    vTaskDelay(20/portTICK_PERIOD_MS);
    ESP_ERROR_CHECK(i2c_master_read_from_device(I2C_PORT_1, SEN55_ADDR, sen55_data, 24,
    1000/portTICK_PERIOD_MS)); //Read Measurements
    vTaskDelay(10/portTICK_PERIOD_MS);
    process_sen55(sen55_data); //Update Global Variables
    //printf("%d", uxTaskGetStackHighWaterMark(NULL));
    //printf("PM2.5: %.2f, PM10: %.2f, Temperature: %.02f, RH: %.2f\n", pm25, pm10, temp, rh);
       xSemaphoreGive(semaphore);
       vTaskSuspend(sen_handle);
       vTaskDelay(1000/portTICK_PERIOD_MS);
    }
}
//RTOS Task for SGP30 @ I2C0
void I2C0_Data_Task(void *params){
   int i2c_master_port = I2C_PORT_0;
   uint8_t sgp30_data[6];
    uint8_t SGP30_MEAS[2] = {0x20, 0x08};
    while (true) {
            ESP_ERROR_CHECK(i2c_master_write_to_device(i2c_master_port, SGP30_ADDR, SGP30_MEAS, 2,
    1000/portTICK_PERIOD_MS)); //Send Measurement Command
    vTaskDelay(15/portTICK_PERIOD_MS);
            i2c_master_read_from_device(i2c_master_port, SGP30_ADDR, sgp30_data, 6,
    1000/portTICK_PERIOD_MS); //Read Measured Values
            //Process received bit sequence
            uint16_t co2_bits = sgp30_data[0] << 8;
            co2_bits |= sgp30_data[1];
            uint16_t voc_bits = sgp30_data[3] << 8;
            voc_bits |= sgp30_data[4];
            float co2_raw = co2_bits;
            float voc_raw = voc_bits;
            co2 = co2_raw;
            voc = voc_raw;
            vTaskDelay(1000/portTICK_PERIOD_MS);
    }
}
//RTOS Task for All_data_collected
void Data_Complete_Task (void *params){
    while(1){
            if(uxSemaphoreGetCount(semaphore) == TASK_COUNT){
                    for (int i=0; i == TASK_COUNT; i++){
```

```
                                xSemaphoreTake(semaphore, portMAX_DELAY);
                        }
                        ReadAttribute();
                        vTaskDelay(1000/portTICK_PERIOD_MS);
                        iaq_score(pm25, pm10, co, no2, co2, voc);

                        //snprintf(data1, sizeof(data1), "%.0f,%.2f,%.0f,%.2f,%.2f,%.2f,%.0f,%.2f",
    pm25_score, pm25, pm10_score, pm10, temp, rh, co2_score, co2);
                //snprintf(data2, sizeof(data2), "%.0f,%.2f,%.0f,%.2f,%.0f,%.2f", voc_score, voc,
    co_score, co, no2_score, no2);
                //snprintf(data3, sizeof(data3), "%.7f,%.7f,%.2f,%s", lat_deg, long_deg, speed_kmh,
    Current_Date_Time); // Lat, Long, Speed, Time

                        // String Send (K-Edition)
                        snprintf(data1, sizeof(data1),
    "K%.0f,%.2f,%.0f,%.2f,%.2f,%.2f,%.0f,%.2f,%.0f,%.2f,%.0f,%.2f", pm25_score, pm25, pm10_score, pm10,
    temp, rh, co2_score, co2, voc_score, voc, co_score, co);
                        snprintf(data2, sizeof(data2), "K%.0f,%.2f,%.7f,%.7f,%.2f,%s", no2_score, no2,
    lat_deg, long_deg, speed_kmh, Current_Date_Time);


                        printf("%s\n", data1);
                        printf("%s\n", data2);
                        ReportAttribute(SENSOR_DATA_ATTR_ID);
                        vTaskDelay(50/portTICK_PERIOD_MS);
                        ReportAttribute(SENSOR_DATA_ATTR_ID_2);
                        vTaskDelay(50/portTICK_PERIOD_MS);

                        sd_card_write();
                        vTaskDelay(9900/portTICK_PERIOD_MS); //THROUGHPUT
                        vTaskResume(mics_handle);
                        vTaskResume(sen_handle);
                }
                else{
                        vTaskDelay(1000/portTICK_PERIOD_MS);
                }
        }
}
 //GPS (UART), Driver Alert
void UART_Data_Task(void *params){
    //Start UART Here
    const uart_port_t uart_num = UART_NUM_1;
    uart_config_t uart_config = {
        .baud_rate = 9600, //GPS baud rate according to datasheet
        .data_bits = UART_DATA_8_BITS,
        .parity = UART_PARITY_DISABLE,
        .stop_bits = UART_STOP_BITS_1,
        .flow_ctrl = UART_HW_FLOWCTRL_DISABLE,
    };
    // Configure UART parameters
    ESP_ERROR_CHECK(uart_param_config(uart_num, &uart_config));
    // Set UART pins(TX: IO4, RX: IO5, RTS: IO18, CTS: IO19)
    ESP_ERROR_CHECK(uart_set_pin(uart_num, TX_pin, RX_pin, UART_PIN_NO_CHANGE, UART_PIN_NO_CHANGE)); //RTS
    and CTS pins will not be used
    // Setup UART buffered IO with event queue
    const int uart_buffer_size = (1024*3);
    QueueHandle_t uart_queue;
    // Install UART driver using an event queue here
    ESP_ERROR_CHECK(uart_driver_install(uart_num, uart_buffer_size, uart_buffer_size, 10, &uart_queue,
    0));
    vTaskDelay(100/portTICK_PERIOD_MS);
    const char* filter_str_loc = "$GPGGA";
    const char* filter_str_speed = "$GPVTG";
    char *loc = NULL;
    char *speed = NULL;
    char str1[100];
    char str2[45];
    int j=0;
    while(1){
            int length;

            ESP_ERROR_CHECK(uart_get_buffered_data_len(uart_num, (size_t*)&length));
            char gps_out[length];

            uart_read_bytes(uart_num, gps_out, length, 1000);

            loc = strstr(gps_out,filter_str_loc);
        if (loc == NULL) {
            printf("GPGGA not found in the sentence\n");
        }
        else{
                char *newline = strchr(loc, '\n');
                if (newline == NULL) {
                    printf("Incomplete NMEA Sentence\n");
                }
                else if(loc){
                            for (int i=0; loc[i] != '\n'; i++){
                                    str1[j] = loc[i];
```

```
                                        j++;
                                }
                                str1[j+1] = '\n';
                                j = 0;
                                printf("%s\n\n", str1);
                                char* token = strtok(str1, ",");
                                token = strtok(NULL, ",");
                                for (int i=0; i<4; i++){
                                        token = strtok(NULL,",");
                                        location[i] = token;
                                }
                                double latitude = atof(location[0]);
                                double longitude = atof(location[2]);

                                 latitude_decimal = floor(latitude/100);
                                 longitude_decimal = floor(longitude/100);

                                if(latitude_decimal == 0 && longitude_decimal == 0){
                                        printf("No Signal\n");
                                }
                                else{
                                        double latitude_decimal_minutes = modf(latitude/100, &latitude);
                                        latitude_decimal_minutes = latitude_decimal_minutes*100;
                                        double longitude_decimal_minutes = modf(longitude/100,
    &longitude);
                                        longitude_decimal_minutes = longitude_decimal_minutes*100;

                                        lat_deg = latitude_decimal + latitude_decimal_minutes/60;
                                        long_deg = longitude_decimal + longitude_decimal_minutes/60;
                                        //printf("%.8f, %.8f\n", lat_deg, long_deg);
                                }
                        }
        }
        //printf("%s\n\n", gps_out);
        speed_str = "0";
             speed = strstr(gps_out,filter_str_speed);
             //printf("%s", speed);
        if (speed == NULL) {
            printf("GPVTG not found in the sentence\n");
        }
        else{
                char *newline2 = strchr(speed, '\n');
                if (newline2 == NULL) {
                    printf("Incomplete NMEA Sentence\n");
                }
                else if (newline2 != NULL && latitude_decimal != 0 && longitude_decimal != 0){
                     for (int i=0; speed[i] != '\n'; i++){
                             str2[j] = speed[i];
                             j++;
                     }
                     str2[j+1] = '\n';
                     j = 0;
                     printf("%s\n\n", str2);
                     if(strlen(str2) > 20){
                                     char* token2 = strtok(str2, ",");
                                     token2 = strtok(NULL, ",");
                                     for (int i=0; i<6; i++){
                                             token2 = strtok(NULL,",");
                                             velocity[i] = token2;
                                             if (strcmp(token2, "N") == 0){
                                                     token2 = strtok(NULL,",");
                                                     speed_str = token2;
                                                     break;
                                             }
                                     }
                     }
                             printf("SPEED STRING: %s\n", speed_str);
                             printf("%.7f, %.7f\n", lat_deg, long_deg);
                             if(speed_str == NULL){
                                     strcpy(speed_str, "0");
                             }
                             speed_kmh = floor(atof(speed_str));
                             printf("%.2f\n\n", speed_kmh);
                             if (speed_kmh < 1){
                                     speed_kmh = 1;
                             }

                }
        }

        //xSemaphoreGive(semaphore);
             //vTaskSuspend(uart_handle);
        //sd_card_write();
        vTaskDelay(1000/portTICK_PERIOD_MS);
    }
}

static void bdb_start_top_level_commissioning_cb(uint8_t mode_mask)
```

```
{
    ESP_ERROR_CHECK(esp_zb_bdb_start_top_level_commissioning(mode_mask));
}

static void bind_cb(esp_zb_zdp_status_t zdo_status, void *user_ctx)
{
    if (zdo_status == ESP_ZB_ZDP_STATUS_SUCCESS) {
        ESP_LOGI(TAG, "Bind response from address(0x%x), endpoint(%d) with status(%d)",
    ((zdo_info_user_ctx_t *)user_ctx)->short_addr,
                 ((zdo_info_user_ctx_t *)user_ctx)->endpoint, zdo_status);
/*         configure report attribute command*/
        esp_zb_zcl_config_report_cmd_t report_cmd;
        bool report_change = 1;
        report_cmd.zcl_basic_cmd.dst_addr_u.addr_short = zb_coord_info.short_addr;
        report_cmd.zcl_basic_cmd.dst_endpoint = zb_coord_info.endpoint;
        report_cmd.zcl_basic_cmd.src_endpoint = ZB_CLIENT_ENDPOINT_4;
        report_cmd.address_mode = ESP_ZB_APS_ADDR_MODE_16_ENDP_PRESENT;
        report_cmd.clusterID = SENSOR_DATA_CLUSTER_ID;                     /*!< Cluster ID to report */
        report_cmd.attributeID = SENSOR_DATA_ATTR_ID;                    /*!< Attribute ID to report */
        report_cmd.attrType = ESP_ZB_ZCL_ATTR_TYPE_CHAR_STRING;                             /*!<
    Attribute type to report refer to zb_zcl_common.h zcl_attr_type */
        report_cmd.reportable_change = &report_change;
        esp_zb_zcl_config_report_cmd_req(&report_cmd);
    }
}

static void ieee_cb(esp_zb_zdp_status_t zdo_status, esp_zb_ieee_addr_t ieee_addr, void *user_ctx)
{
    if (zdo_status == ESP_ZB_ZDP_STATUS_SUCCESS) {
        memcpy(&(zb_coord_info.ieee_addr), ieee_addr, sizeof(esp_zb_ieee_addr_t));
        ESP_LOGI(TAG, "IEEE address: %02x:%02x:%02x:%02x:%02x:%02x:%02x:%02x",
                 ieee_addr[7], ieee_addr[6], ieee_addr[5], ieee_addr[4],
                 ieee_addr[3], ieee_addr[2], ieee_addr[1], ieee_addr[0]);


        /* BIND the on-off light to on-off switch */
        esp_zb_zdo_bind_req_param_t bind_req;
        memcpy(&(bind_req.src_address), zb_coord_info.ieee_addr, sizeof(esp_zb_ieee_addr_t));
        bind_req.src_endp = zb_coord_info.endpoint;
        bind_req.cluster_id = SENSOR_DATA_CLUSTER_ID;
        bind_req.dst_addr_mode = ESP_ZB_ZDO_BIND_DST_ADDR_MODE_64_BIT_EXTENDED;
        esp_zb_get_long_address(bind_req.dst_address_u.addr_long);
        bind_req.dst_endp = ZB_CLIENT_ENDPOINT_4;
        bind_req.req_dst_addr = zb_coord_info.short_addr;
        static zdo_info_user_ctx_t test_info_ctx;
        test_info_ctx.endpoint = ZB_CLIENT_ENDPOINT_4;
        test_info_ctx.short_addr = zb_coord_info.short_addr;
        esp_zb_zdo_device_bind_req(&bind_req, bind_cb, (void *) & (test_info_ctx));
    }
}

static void ep_cb(esp_zb_zdp_status_t zdo_status, uint8_t ep_count, uint8_t *ep_id_list, void *user_ctx)
{
    if (zdo_status == ESP_ZB_ZDP_STATUS_SUCCESS) {
        ESP_LOGI(TAG, "Active endpoint response: status(%d) and endpoint count(%d)", zdo_status,
    ep_count);
        for (int i = 0; i < ep_count; i++) {
            ESP_LOGI(TAG, "Endpoint ID List: %d", ep_id_list[i]);
        }
    }
}

static void simple_desc_cb(esp_zb_zdp_status_t zdo_status, esp_zb_af_simple_desc_1_1_t *simple_desc, void
    *user_ctx)
{
    if (zdo_status == ESP_ZB_ZDP_STATUS_SUCCESS) {
        ESP_LOGI(TAG, "Simple desc response: status(%d), device_id(%d), app_version(%d), profile_id(0x%x),
    endpoint_ID(%d)", zdo_status,
                 simple_desc->app_device_id, simple_desc->app_device_version, simple_desc->app_profile_id,
    simple_desc->endpoint);

        for (int i = 0; i < (simple_desc->app_input_cluster_count +
    simple_desc->app_output_cluster_count); i++) {
            ESP_LOGI(TAG, "Cluster ID list: 0x%x", *(simple_desc->app_cluster_list + i));
        }
    }
}

static void user_find_cb(esp_zb_zdp_status_t zdo_status, uint16_t addr, uint8_t endpoint, void *user_ctx)
{
    if (zdo_status == ESP_ZB_ZDP_STATUS_SUCCESS) {
        ESP_LOGI(TAG, "Match desc response: status(%d), address(0x%x), endpoint(%d)", zdo_status, addr,
    endpoint);
        /* save into remote device record structure for future use */
        zb_coord_info.endpoint = endpoint;
        zb_coord_info.short_addr = addr;
        /* find the active endpoint */
        esp_zb_zdo_active_ep_req_param_t active_ep_req;
        active_ep_req.addr_of_interest = zb_coord_info.short_addr;
        esp_zb_zdo_active_ep_req(&active_ep_req, ep_cb, NULL);
        /* get the node simple descriptor */
        esp_zb_zdo_simple_desc_req_param_t simple_desc_req;
        simple_desc_req.addr_of_interest = addr;
        simple_desc_req.endpoint = endpoint;
        esp_zb_zdo_simple_desc_req(&simple_desc_req, simple_desc_cb, NULL);
        /* get the light ieee address */
        esp_zb_zdo_ieee_addr_req_param_t ieee_req;
        ieee_req.addr_of_interest = zb_coord_info.short_addr;
        ieee_req.dst_nwk_addr = zb_coord_info.short_addr;
        ieee_req.request_type = 0;
        ieee_req.start_index = 0;
        esp_zb_zdo_ieee_addr_req(&ieee_req, ieee_cb, NULL);
    }
}

void esp_zb_app_signal_handler(esp_zb_app_signal_t *signal_struct)
{
    uint32_t *p_sg_p       = signal_struct->p_app_signal;
    esp_err_t err_status = signal_struct->esp_err_status;
    esp_zb_app_signal_type_t sig_type = *p_sg_p;
    esp_zb_zdo_signal_leave_params_t *leave_params = NULL;
    switch (sig_type) {
    case ESP_ZB_BDB_SIGNAL_DEVICE_FIRST_START:
    case ESP_ZB_BDB_SIGNAL_DEVICE_REBOOT:
    case ESP_ZB_BDB_SIGNAL_STEERING:
        if (err_status != ESP_OK) {
            ESP_LOGW(TAG, "Stack %s failure with %s status,
    steering",esp_zb_zdo_signal_to_string(sig_type), esp_err_to_name(err_status));
            esp_zb_scheduler_alarm((esp_zb_callback_t)bdb_start_top_level_commissioning_cb,
    ESP_ZB_BDB_MODE_NETWORK_STEERING, 1000);
        } else {
            /* device auto start successfully and on a formed network */
            esp_zb_ieee_addr_t extended_pan_id;
            esp_zb_get_extended_pan_id(extended_pan_id);
            ESP_LOGI(TAG, "Joined network successfully (Extended PAN ID:
    %02x:%02x:%02x:%02x:%02x:%02x:%02x:%02x, PAN ID: 0x%04hx, Channel:%d, Short Address: 0x%04hx)",
                     extended_pan_id[7], extended_pan_id[6], extended_pan_id[5], extended_pan_id[4],
                     extended_pan_id[3], extended_pan_id[2], extended_pan_id[1], extended_pan_id[0],
                     esp_zb_get_pan_id(), esp_zb_get_current_channel(), esp_zb_get_short_address());
            esp_zb_zdo_match_desc_req_param_t  find_req;
            find_req.addr_of_interest = 0x0000;
            find_req.dst_nwk_addr = 0x0000;
            /* find the match on-off light device */
            esp_zb_zdo_find_on_off_light(&find_req, user_find_cb, NULL);

        }
        break;
    case ESP_ZB_ZDO_SIGNAL_LEAVE:
        leave_params = (esp_zb_zdo_signal_leave_params_t *)esp_zb_app_signal_get_params(p_sg_p);
        if (leave_params->leave_type == ESP_ZB_NWK_LEAVE_TYPE_RESET) {
            ESP_LOGI(TAG, "Reset device");
        }
        break;
    default:
        ESP_LOGI(TAG, "ZDO signal: %s (0x%x), status: %s", esp_zb_zdo_signal_to_string(sig_type),
    sig_type,
                 esp_err_to_name(err_status));
        break;
    }
}

static esp_err_t zb_attribute_reporting_handler(const esp_zb_zcl_report_attr_message_t *message)
{
    ESP_RETURN_ON_FALSE(message, ESP_FAIL, TAG, "Empty message");
    ESP_RETURN_ON_FALSE(message->status == ESP_ZB_ZCL_STATUS_SUCCESS, ESP_ERR_INVALID_ARG, TAG, "Received
    message: error status(%d)",
                        message->status);
    ESP_LOGI(TAG, "Received report from address(0x%x) src endpoint(%d) to dst endpoint(%d) cluster(0x%x)",
    message->src_address.u.short_addr,
             message->src_endpoint, message->dst_endpoint, message->cluster);
    ESP_LOGI(TAG, "Received report information: attribute(0x%x), type(0x%x), value(%d)\n",
    message->attribute.id, message->attribute.data.type,
             message->attribute.data.value ? *(uint8_t *)message->attribute.data.value : 0);
    return ESP_OK;
}

static esp_err_t zb_configure_report_resp_handler(const esp_zb_zcl_cmd_config_report_resp_message_t
    *message)
{
    ESP_RETURN_ON_FALSE(message, ESP_FAIL, TAG, "Empty message");
    ESP_RETURN_ON_FALSE(message->info.status == ESP_ZB_ZCL_STATUS_SUCCESS, ESP_ERR_INVALID_ARG, TAG,
    "Received message: error status(%d)",
                        message->info.status);
    return ESP_OK;
}
```

```
static esp_err_t zb_read_attr_resp_handler(const esp_zb_zcl_cmd_read_attr_resp_message_t *message){
    ESP_RETURN_ON_FALSE(message, ESP_FAIL, TAG, "Empty message");
    ESP_RETURN_ON_FALSE(message->info.status == ESP_ZB_ZCL_STATUS_SUCCESS, ESP_ERR_INVALID_ARG, TAG,
     "Received message: error status(%d)",
                        message->info.status);
    ESP_LOGI(TAG, "Received report information: attribute(0x%x), type(0x%x), value(%s)\n",
     message->attribute.id, message->attribute.data.type,
             (char *)message->attribute.data.value);
    char buffer[20];
    snprintf(buffer, 20, "%s", (char* )message->attribute.data.value);

    strcpy(Current_Date_Time, buffer);
    return ESP_OK;
}

static esp_err_t zb_action_handler(esp_zb_core_action_callback_id_t callback_id, const void *message)
{
     //THESE ARE EXECUTED WHEN A CALLBACK ID IS RECEVIED BY THIS DEVICE
    esp_err_t ret = ESP_OK;
    switch (callback_id) {
    //IMPORTANT CASE FOR RECEIVING ATTRIBUTE REPORTS
    case ESP_ZB_CORE_REPORT_ATTR_CB_ID:
        ret = zb_attribute_reporting_handler((esp_zb_zcl_report_attr_message_t *)message);
        break;
    //IMPORTANT CASE FOR CONFIGURING A REPORT (ATTRIBUTE REPORT) TO THE PROPER FORMAT
    case ESP_ZB_CORE_CMD_REPORT_CONFIG_RESP_CB_ID:
        ret = zb_configure_report_resp_handler((esp_zb_zcl_cmd_config_report_resp_message_t *)message);
        break;
    //IMPORTANT CASE WHEN SETTING ATTRIBUTE TO A SPECIFIC VALUE
    //case ESP_ZB_CORE_SET_ATTR_VALUE_CB_ID:
        //ret = zb_attribute_handler((esp_zb_zcl_set_attr_value_message_t *)message);
        //break;
    case ESP_ZB_CORE_CMD_READ_ATTR_RESP_CB_ID:
     ret = zb_read_attr_resp_handler((esp_zb_zcl_cmd_read_attr_resp_message_t *)message);
     break;
    default:
        ESP_LOGW(TAG, "Receive Zigbee action(0x%x) callback", callback_id);
        break;
    }
    return ret;
}

static void esp_zb_task(void *pvParameters)
{
    /* initialize Zigbee stack */
     xSemaphoreGive(semaphore);
    esp_zb_cfg_t zb_nwk_cfg = ESP_ZB_ZED_CONFIG();
    esp_zb_init(&zb_nwk_cfg);

    const uint8_t attr_access = ESP_ZB_ZCL_ATTR_ACCESS_REPORTING | ESP_ZB_ZCL_ATTR_ACCESS_READ_WRITE;

    esp_zb_attribute_list_t *esp_zb_sensordata_cluster =
     esp_zb_zcl_attr_list_create(SENSOR_DATA_CLUSTER_ID);
    ESP_ERROR_CHECK(esp_zb_custom_cluster_add_custom_attr(esp_zb_sensordata_cluster, SENSOR_DATA_ATTR_ID,
     ESP_ZB_ZCL_ATTR_TYPE_CHAR_STRING, attr_access, &data1));
    ESP_ERROR_CHECK(esp_zb_custom_cluster_add_custom_attr(esp_zb_sensordata_cluster,
     SENSOR_DATA_ATTR_ID_2, ESP_ZB_ZCL_ATTR_TYPE_CHAR_STRING, attr_access, &data2));
    ESP_ERROR_CHECK(esp_zb_custom_cluster_add_custom_attr(esp_zb_sensordata_cluster,
     SENSOR_DATA_ATTR_ID_3, ESP_ZB_ZCL_ATTR_TYPE_CHAR_STRING, attr_access, &data3));
    esp_zb_cluster_list_t *esp_zb_cluster_list = esp_zb_zcl_cluster_list_create();
    esp_zb_cluster_list_add_custom_cluster(esp_zb_cluster_list, esp_zb_sensordata_cluster,
     ESP_ZB_ZCL_CLUSTER_SERVER_ROLE);

    esp_zb_ep_list_t *esp_zb_ep_list = esp_zb_ep_list_create();

    //Set Endpoint HERE
    esp_zb_ep_list_add_ep(esp_zb_ep_list, esp_zb_cluster_list, ZB_CLIENT_ENDPOINT_4,
     ESP_ZB_AF_HA_PROFILE_ID, ESP_ZB_HA_ON_OFF_SWITCH_DEVICE_ID);


    esp_zb_device_register(esp_zb_ep_list);
    esp_zb_core_action_handler_register(zb_action_handler);
    esp_zb_set_primary_network_channel_set(ESP_ZB_PRIMARY_CHANNEL_MASK);
    esp_zb_set_secondary_network_channel_set(ESP_ZB_SECONDARY_CHANNEL_MASK);
    ESP_ERROR_CHECK(esp_zb_start(true));
    esp_zb_main_loop_iteration();
}

void app_main(void)
{
     //OUTDOOR MODE SETUP -- LAST EDIT
    esp_zb_platform_config_t config = {
        .radio_config = ESP_ZB_DEFAULT_RADIO_CONFIG(),
        .host_config = ESP_ZB_DEFAULT_HOST_CONFIG(),
    };
    ESP_ERROR_CHECK(nvs_flash_init());
    ESP_ERROR_CHECK(esp_zb_platform_config(&config));
    semaphore = xSemaphoreCreateCounting(TASK_COUNT, 0);
    heated = 1;
    sensor_init(&heated);

    //configure_led();
    //led_alert_init();


   xTaskCreate(esp_zb_task, "Zigbee_main", 4096, NULL, 5, &zb_handle); //START ZIGBEE
   vTaskDelay(15000/portTICK_PERIOD_MS);
    xTaskCreate(ADC1_Data_Task, "ADC1 Data Task", 1024*3, NULL, 5, &mics_handle);
    xTaskCreate(I2C0_Data_Task, "I2C0 Data Task", 1024*3, NULL, 5, &sgp_handle);
    xTaskCreate(I2C1_Data_Task, "I2C1 Data Task", 1024*2, NULL, 5, &sen_handle);
    xTaskCreate(UART_Data_Task, "UART Data Task", 1024*8, NULL, 5, &uart_handle);
   xTaskCreate(Data_Complete_Task, "Data Upload and Save", 4096, NULL, 5, &finish_handle);


}
```

Appendix G
Code for Zigbee System - Gateway
Template Source Code: https://github.com/espressif/esp-zigbee-sdk/tree/main/examples/esp_zigbee_gateway

```
#include <fcntl.h>
#include <string.h>
#include "freertos/FreeRTOS.h"
#include "freertos/task.h"
#include "driver/usb_serial_jtag.h"
#include "esp_coexist_internal.h"
#include "esp_log.h"
#include "esp_netif.h"
#include "esp_spiffs.h"
#include "esp_vfs_eventfd.h"
#include "esp_vfs_dev.h"
#include "esp_vfs_usb_serial_jtag.h"
#include "esp_wifi.h"
#include "nvs_flash.h"
#include "esp_zigbee_gateway.h"
#include "esp_check.h"
#include "esp_zigbee_core.h"
#include "esp_partition.h"
#include "protocol_examples_common.h"
#include "esp_tls.h"
#include "esp_ota_ops.h"
#include <sys/param.h>
#include <stdio.h>
#include <stdint.h>
#include <stddef.h>
#include <string.h>
#include "esp_wifi.h"
#include "esp_system.h"
#include "nvs_flash.h"
#include "esp_event.h"
#include "esp_netif.h"
#include "freertos/FreeRTOS.h"
#include "freertos/task.h"
#include "freertos/semphr.h"
#include "freertos/queue.h"
#include "lwip/sockets.h"
#include "lwip/dns.h"
#include "lwip/netdb.h"
#include "esp_log.h"
#include "mqtt_client.h"
#include <stdlib.h>
#include <time.h>
#include <esp_wifi_types.h>
#include "driver/uart.h"
#include <time.h>
#include <sys/time.h>
#include "esp_attr.h"
#include "esp_sleep.h"
#include "esp_sntp.h"
#include "driver/gpio.h"

#if (!defined ZB_MACSPLIT_HOST && defined ZB_MACSPLIT_DEVICE)
#error Only Zigbee gateway host device should be defined
#endif

SemaphoreHandle_t node3_semaphore, node1_semaphore, node4_semaphore, mqtt_semaphore, semaphore;
#define TASK_COUNT 9
TaskHandle_t zigbee_handle, upload_handle, timer_handle, node1_upload_handle, node4_upload_handle;


//global variables
int test_count=0;
char payload[50]="Payload sample text"; //50 characters max
esp_mqtt_client_handle_t MQTtest;
char sample_char[50];
uint16_t SENSOR_DATA_CLUSTER_ID = 0xFFF1;
uint16_t TIME_CLUSTER_ID = 0xFFF2;
uint8_t SENSOR_DATA_ATTR_ID = 0;
uint8_t SENSOR_DATA_ATTR_ID_2 = 1;
uint8_t TIME_ATTR_ID = 3;

const uart_port_t uart_num = UART_NUM_2;
```

```
int node_id;

int node1_data1_flag,node1_data2_flag,node1_data3_flag,node2_data1_flag,node2_data2_flag;
int node3_data1_flag,node3_data2_flag,node3_data3_flag,node4_data1_flag,node4_data2_flag,node4_data3_flag;

float pm25_in, pm10_in, co_in, no2_in, co2_in, voc_in, co2_in3, voc_in3;
float pm25_out, pm10_out, co_out, no2_out, co2_out, voc_out;
float speed;

float lat_decimal = 0;
float long_decimal = 0;


char Current_Date_Time[100];
char Upload_Time[20];
char json_payload[701];
char json_payload_1[701];
char json_payload_3[701];
char json_payload_4[701];

char n11_buffer[76], n12_buffer[76], n13_buffer[76], n41_buffer[76], n42_buffer[76], n43_buffer[76],
    n21_buffer[76], n22_buffer[76], n31_buffer[76], n32_buffer[76], n33_buffer[76];

double lat_deg, long_deg;
char* parsed_data1[16];
char* parsed_data3[16];
char* parsed_data4[18];
char* data[25];
char count[2];

//UART2 for Occupancy Sensor
#define TX_pin 1
#define RX_pin 2

//Drive Alert LEDs
#define RECIRC_LED 15 //WHITE LED
#define INTAKE_LED 16 //BLUE LED

#if CONFIG_BROKER_CERTIFICATE_OVERRIDDEN == 1
static const uint8_t mqtt_eclipseprojects_io_pem_start[]  = "-----BEGIN CERTIFICATE-----\n"
    CONFIG_BROKER_CERTIFICATE_OVERRIDE "\n-----END CERTIFICATE-----";
#else
extern const uint8_t mqtt_eclipseprojects_io_pem_start[]
    asm("_binary_mqtt_eclipseprojects_io_pem_start");
#endif
extern const uint8_t mqtt_eclipseprojects_io_pem_end[]   asm("_binary_mqtt_eclipseprojects_io_pem_end");

static const char *TAG = "ESP_ZB_GATEWAY";
static const char *TAG_MQTT = "MQTTS_EXAMPLE";

/* Note: Please select the correct console output port based on the development board in menuconfig */
#if CONFIG_ESP_CONSOLE_USB_SERIAL_JTAG
esp_err_t esp_zb_gateway_console_init(void)
{
    esp_err_t ret = ESP_OK;
    /* Disable buffering on stdin */
    setvbuf(stdin, NULL, _IONBF, 0);

    /* Minicom, screen, idf_monitor send CR when ENTER key is pressed */
    esp_vfs_dev_usb_serial_jtag_set_rx_line_endings(ESP_LINE_ENDINGS_CR);
    /* Move the caret to the beginning of the next line on '\n' */
    esp_vfs_dev_usb_serial_jtag_set_tx_line_endings(ESP_LINE_ENDINGS_CRLF);

    /* Enable non-blocking mode on stdin and stdout */
    fcntl(fileno(stdout), F_SETFL, O_NONBLOCK);
    fcntl(fileno(stdin), F_SETFL, O_NONBLOCK);

    usb_serial_jtag_driver_config_t usb_serial_jtag_config = USB_SERIAL_JTAG_DRIVER_CONFIG_DEFAULT();
    ret = usb_serial_jtag_driver_install(&usb_serial_jtag_config);
    esp_vfs_usb_serial_jtag_use_driver();
    esp_vfs_dev_uart_register();
    return ret;
}
#endif

void time_sync_notification_cb(struct timeval *tv)
{
    ESP_LOGI(TAG, "Notification of a time synchronization event");
}

//RETURNS the final time in string form
void Get_current_date_time(char *date_time){
    char strftime_buf[100];
    time_t now;
        struct tm timeinfo;
        time(&now);
        localtime_r(&now, &timeinfo);
```

```
            // Set timezone to Indian Standard Time
                              setenv("TZ", "UTC-08:00", 1);
                tzset();
                localtime_r(&now, &timeinfo);

                strftime(strftime_buf, sizeof(strftime_buf), "%c", &timeinfo);
                //strftime(date_time, 100, "%Y-%m-%dT%H:%M:%S", &timeinfo);
                //ESP_LOGI(TAG, "The current date/time in Philippines is: %s", strftime_buf);

                // Extract the time values manually
                int year = timeinfo.tm_year + 1900;  // Years since 1900, so add 1900
                int month = timeinfo.tm_mon + 1;     // Months are 0-based, so add 1
                int day = timeinfo.tm_mday;
                int hour = timeinfo.tm_hour;
                int minute = timeinfo.tm_min;
                int second = timeinfo.tm_sec;

                //concatenate the string manually
                sprintf(strftime_buf, "%d-%02d-%02dT%02d:%02d:%02d", year, month, day, hour, minute,
    second);
               strcpy(date_time,strftime_buf);
}

//initializes SNTP server settings
static void initialize_sntp(void)
{
    ESP_LOGI(TAG, "Initializing SNTP");
    esp_sntp_setoperatingmode(SNTP_OPMODE_POLL);
    esp_sntp_setservername(0, "time.google.com");
    sntp_set_time_sync_notification_cb(time_sync_notification_cb);
#ifdef CONFIG_SNTP_TIME_SYNC_METHOD_SMOOTH
    sntp_set_sync_mode(SNTP_SYNC_MODE_SMOOTH);
#endif
    esp_sntp_init();
}

// helper function that obtains timezone
static void obtain_time(void)
{
    initialize_sntp();
    // wait for time to be set
    time_t now = 0;
    struct tm timeinfo = { 0 };
    int retry = 0;
    const int retry_count = 10;
    while (sntp_get_sync_status() == SNTP_SYNC_STATUS_RESET && ++retry < retry_count) {
        ESP_LOGI(TAG, "Waiting for system time to be set... (%d/%d)", retry, retry_count);
        vTaskDelay(2000 / portTICK_PERIOD_MS);
    }
    time(&now);
    localtime_r(&now, &timeinfo);
}

//function that actually syncs with NTP server
 void Set_SystemTime_SNTP()  {

     time_t now;
        struct tm timeinfo;
        time(&now);
        localtime_r(&now, &timeinfo);
        // Is time set? If not, tm_year will be (1970 - 1900).
        if (timeinfo.tm_year < (2016 - 1900)) {
            ESP_LOGI(TAG, "Time is not set yet. Connecting to WiFi and getting time over NTP.");
            obtain_time();
            // update 'now' variable with current time
            time(&now);
        }
}

void init_uart(void){
    uart_driver_delete(UART_NUM_0);
    uart_config_t uart_config = {
        .baud_rate = 9600, //GPS baud rate according to datasheet
        .data_bits = UART_DATA_8_BITS,
        .parity = UART_PARITY_DISABLE,
        .stop_bits = UART_STOP_BITS_1,
        .flow_ctrl = UART_HW_FLOWCTRL_DISABLE,
    };
    // Configure UART parameters
    ESP_ERROR_CHECK(uart_param_config(uart_num, &uart_config));
    // Set UART pins(TX: IO4, RX: IO5, RTS: IO18, CTS: IO19)
    ESP_ERROR_CHECK(uart_set_pin(uart_num, TX_pin, RX_pin, UART_PIN_NO_CHANGE, UART_PIN_NO_CHANGE)); //RTS
    and CTS pins will not be used
    // Setup UART buffered IO with event queue
    const int uart_buffer_size = (1024*4);
    QueueHandle_t uart_queue;
    // Install UART driver using an event queue here
    ESP_ERROR_CHECK(uart_driver_install(uart_num, uart_buffer_size, uart_buffer_size, 10, &uart_queue,
    0));
}

//global float
float speed = 1.00;
char occu_cleaned[8];
char occu_count[8] = {};

void cleanData(char *raw_data, char *cleaned_data) {
    int i;
    for (i = 0; i < strlen(raw_data); i++) {
        if (raw_data[i] == '\n') {
            break;  // Stop iterating if newline is detected
        }
        cleaned_data[i] = raw_data[i]; // Copy character to cleaned_data
    }
    cleaned_data[i] = '\0'; // Null-terminate the cleaned string
}


void occu_task(void){
    //Occupancy Exchange
    int length;
    char buffer[10]; // Adjust size as needed
    snprintf(buffer, sizeof(buffer), "%.0f", speed);
    //Send UART Command
    uart_write_bytes(uart_num, buffer, strlen(buffer));
    //uart_flush(uart_num);
    //Receive Passenger Count
    ESP_ERROR_CHECK(uart_get_buffered_data_len(uart_num, (size_t*)&length));
    while(length == 0){
            //wait here
            printf("WAITING...\n");
            vTaskDelay(1000/portTICK_PERIOD_MS);
            ESP_ERROR_CHECK(uart_get_buffered_data_len(uart_num, (size_t*)&length)); //check every1s for
    update
    }
    char occu_in[length];
    uart_read_bytes(uart_num, occu_in, length, 1000);

    //call data cleaner
    cleanData(occu_in,occu_cleaned);
    //snprintf(occu_count, 8, "%s.00", occu_cleaned);
    printf("COUNT: %s\n", occu_cleaned);
    vTaskDelay(1000/portTICK_PERIOD_MS);
}

static void wifi_event_handler(void *event_handler_arg, esp_event_base_t event_base, int32_t event_id,
    void *event_data)
{
    switch (event_id)
    {
    case WIFI_EVENT_STA_START:
        printf("WiFi connecting ... \n");
        break;
    case WIFI_EVENT_STA_CONNECTED:
        printf("WiFi connected ... \n");
        break;
    case WIFI_EVENT_STA_DISCONNECTED:
        printf("WiFi lost connection ... \n");
        esp_wifi_connect();
        break;
    case IP_EVENT_STA_GOT_IP:
        printf("WiFi got IP...\n\n");
        break;
    default:
        break;
    }
}

void wifi_connection()
{
    // 1 - Wi-Fi/LwIP Init Phase
    esp_netif_init();                    // TCP/IP initiation
    esp_netif_create_default_wifi_sta(); // WiFi station
    wifi_init_config_t wifi_initiation = WIFI_INIT_CONFIG_DEFAULT();
    esp_wifi_init(&wifi_initiation); // s1.4
    esp_event_handler_register(WIFI_EVENT, ESP_EVENT_ANY_ID, wifi_event_handler, NULL);
    esp_event_handler_register(IP_EVENT, IP_EVENT_STA_GOT_IP, wifi_event_handler, NULL);
    wifi_config_t wifi_configuration = {
        .sta = {
                    .ssid = "ssid",
                    .password = "password"}};
    esp_wifi_set_mode(WIFI_MODE_STA);


    esp_wifi_set_config(WIFI_IF_STA, &wifi_configuration);
    // 3 - Wi-Fi Start Phase
    printf("WiFi Starting...\n");
    esp_wifi_start();
    // 4- Wi-Fi Connect Phase
    esp_wifi_connect();
}

void mqtt_publish(esp_mqtt_client_handle_t client){
     const char *HiveMQ = "UPCARE/UNDERGRAD/ECE199_PUB2324";
     esp_mqtt_client_publish(client, HiveMQ, json_payload, 0, 2, 0);
}

void mqtt_publish_node_specific(esp_mqtt_client_handle_t client, int node_num){
     const char *Topic_Name = "UPCARE/UNDERGRAD/ECE199_PUB2324";
     if(node_num == 1){
              esp_mqtt_client_publish(client, Topic_Name, json_payload_1, 0, 2, 0);
     }
     else if (node_num == 3){
              esp_mqtt_client_publish(client, Topic_Name, json_payload_3, 0, 2, 0);
     }
     else if (node_num == 4){
              esp_mqtt_client_publish(client, Topic_Name, json_payload_4, 0, 2, 0);
     }

}

void driver_alert_init (void){
     gpio_reset_pin(RECIRC_LED);
     gpio_set_direction(RECIRC_LED, GPIO_MODE_OUTPUT);
     gpio_reset_pin(INTAKE_LED);
     gpio_set_direction(INTAKE_LED, GPIO_MODE_OUTPUT);
     gpio_set_level(INTAKE_LED, 0);
     gpio_set_level(RECIRC_LED, 1);
}
void driver_alert(void){ // (Switch to INTAKE when Above ORANGE Indoor) AND (In_Score > Out_Score)
     if((co_in > 76 && co_in > co_out) || (no2_in > 76 && no2_in > no2_out) || (co2_in > 76 && no2_in >
     co2_out) || (voc_in > 76 && voc_in > voc_out)){
                       gpio_set_level(INTAKE_LED, 1);
                       gpio_set_level(RECIRC_LED, 0);
     }
     else{
              gpio_set_level(INTAKE_LED, 0);
              gpio_set_level(RECIRC_LED, 1);
     }
}

void upload_node1 (void* params){
     while(1){
              if(strcmp(n11_buffer, "") && strcmp(n12_buffer, "")){

                       int node_num = 1;
                       //UPLOAD NODE 1 DATA
                       char* token11 = strtok(n11_buffer, "K");
                       token11 = strtok(token11, ",");
                       for (int i=0; i<12; i++){
                                parsed_data1[i] = token11;
                                token11 = strtok(NULL, ",");
                       }

                       char* token12 = strtok(n12_buffer, "K");
                       token12 = strtok(token12, ",");
                       for (int i=0; i<3; i++){
                                parsed_data1[i+12] = token12;
                                token12 = strtok(NULL, ",");
                       }
//                     co_in = atof(data[10]);
//                     co2_in = atof(data[6]);
//                     no2_in = atof(data[12]);
//                     pm25_in = atof(data[0]);
//                     pm10_in = atof(data[2]);
//                     voc_in = atof(data[8]);
                       //driver_alert(); //UPDATE DRIVER ALERT
                       //snprintf(json_payload_1, 601,
     "{\"COraw\":\"%s\",\"COindex\":\"%s\",\"CO2raw\":\"%s\",\"CO2index\":\"%s\",\"NO2raw\":\"%s\",\"NO2ind
     ex\":\"%s\",\"PM25raw\":\"%s\",\"PM25index\":\"%s\",\"PM10raw\":\"%s\",\"PM10index\":\"%s\",\"VOCraw\"
     :\"%s\",\"VOCindex\":\"%s\",\"T\":\"%s\",\"RH\":\"%s\",\"source\":\"Node
     1\",\"local_time\":\"%s\",\"type\":\"data\"}",data[11],data[10],data[7],data[6],data[13],data[12],data
     [1],data[0],data[3],data[2],data[9],data[8],data[4],data[5],n13_buffer);

                       printf("SENDING NODE 1\n");
                       snprintf(json_payload_1, 650,
     "{\"COraw\":\"%s\",\"COindex\":\"%s\",\"CO2raw\":\"%s\",\"CO2index\":\"%s\",\"NO2raw\":\"%s\",\"NO2ind
     ex\":\"%s\",\"PM25raw\":\"%s\",\"PM25index\":\"%s\",\"PM10raw\":\"%s\",\"PM10index\":\"%s\",\"VOCraw\"
     :\"%s\",\"VOCindex\":\"%s\",\"T\":\"%s\",\"RH\":\"%s\",\"source\":\"Node
     1\",\"local_time\":\"%s\",\"type\":\"data\"}","0.01","1",parsed_data1[7],parsed_data1[6],"0.01","1",pa
     rsed_data1[1],parsed_data1[0],parsed_data1[3],parsed_data1[2],parsed_data1[9],parsed_data1[8],parsed_d
     ata1[4],parsed_data1[5],parsed_data1[14]);
                       mqtt_publish_node_specific(MQTtest, node_num);
```

```
                        strcpy(n11_buffer, "");
                        strcpy(n12_buffer, "");
                }
                vTaskDelay(500/portTICK_PERIOD_MS);

        }
}

void upload_node3 (void* params){
        while(1){
                if(strcmp(n31_buffer, "") && strcmp(n32_buffer, "")){

                        int node_num = 3;
                        //UPLOAD NODE 3 DATA
                        char* token31 = strtok(n31_buffer, "K");
                        token31 = strtok(token31, ",");
                        for (int i=0; i<12; i++){
                                parsed_data3[i] = token31;
                                token31 = strtok(NULL, ",");
                        }

                        char* token32 = strtok(n32_buffer, "K");
                        token32 = strtok(token32, ",");
                        for (int i=0; i<3; i++){
                                parsed_data3[i+12] = token32;
                                token32 = strtok(NULL, ",");
                        }
//                      co_in = atof(data[10]);
//                      co2_in = atof(data[6]);
//                      no2_in = atof(data[12]);
//                      pm25_in = atof(data[0]);
//                      pm10_in = atof(data[2]);
//                      voc_in = atof(data[8]);
                        //driver_alert(); //UPDATE DRIVER ALERT
                        //snprintf(json_payload_1, 601,
        "{\"COraw\":\"%s\",\"COindex\":\"%s\",\"CO2raw\":\"%s\",\"CO2index\":\"%s\",\"NO2raw\":\"%s\",\"NO2ind
        ex\":\"%s\",\"PM25raw\":\"%s\",\"PM25index\":\"%s\",\"PM10raw\":\"%s\",\"PM10index\":\"%s\",\"VOCraw\"
        :\"%s\",\"VOCindex\":\"%s\",\"T\":\"%s\",\"RH\":\"%s\",\"source\":\"Node
        1\",\"local_time\":\"%s\",\"type\":\"data\"}",data[11],data[10],data[7],data[6],data[13],data[12],data
        [1],data[0],data[3],data[2],data[9],data[8],data[4],data[5],n13_buffer);

                        printf("SENDING NODE 3\n");
                        snprintf(json_payload_3, 650,
        "{\"COraw\":\"%s\",\"COindex\":\"%s\",\"CO2raw\":\"%s\",\"CO2index\":\"%s\",\"NO2raw\":\"%s\",\"NO2ind
        ex\":\"%s\",\"PM25raw\":\"%s\",\"PM25index\":\"%s\",\"PM10raw\":\"%s\",\"PM10index\":\"%s\",\"VOCraw\"
        :\"%s\",\"VOCindex\":\"%s\",\"T\":\"%s\",\"RH\":\"%s\",\"source\":\"Node
        3\",\"local_time\":\"%s\",\"type\":\"data\"}","0.01","1",parsed_data3[7],parsed_data3[6],"0.01","1",pa
        rsed_data3[1],parsed_data3[0],parsed_data3[3],parsed_data3[2],parsed_data3[9],parsed_data3[8],parsed_d
        ata3[4],parsed_data3[5],parsed_data3[14]);
                        mqtt_publish_node_specific(MQTtest, node_num);
                        strcpy(n31_buffer, "");
                        strcpy(n32_buffer, "");
                }
                vTaskDelay(500/portTICK_PERIOD_MS);
        }
}

void upload_node4 (void* params){
        while(1){
                if(strcmp(n41_buffer, "") && strcmp(n42_buffer, "")){

                        //UPLOAD NODE 4 DATA
                        int node_num = 4;
                        char* token41 = strtok(n41_buffer, "K");
                        token41 = strtok(token41, ",");
                        for (int i=0; i<12; i++){
                                parsed_data4[i] = token41;
                                token41 = strtok(NULL, ",");
                        }

                        char* token42 = strtok(n42_buffer, "K");
                        token42 = strtok(token42, ",");
                        for (int i=0; i<6; i++){
                                parsed_data4[i+12] = token42;
                                token42 = strtok(NULL, ",");
                        }
                        //data4[16] is SPEED - REMOVED for now
/*                      co_out = atof(data4[10]);
                        co2_out = atof(data4[6]);
                        no2_out = atof(data4[12]);
                        pm25_out = atof(data4[0]);
                        pm10_out = atof(data4[2]);
                        voc_out = atof(data4[8]);*/
                        //speed = atof(data4[16]);
                        //lat_decimal = atof(data4[14]);
                        //long_decimal = atof(data4[15]);
                        //occu_task();
```

```
                    snprintf(json_payload_4, 650,
    "{\"COraw\":\"%s\",\"COindex\":\"%s\",\"CO2raw\":\"%s\",\"CO2index\":\"%s\",\"NO2raw\":\"%s\",\"NO2ind
    ex\":\"%s\",\"PM25raw\":\"%s\",\"PM25index\":\"%s\",\"PM10raw\":\"%s\",\"PM10index\":\"%s\",\"VOCraw\"
    :\"%s\",\"VOCindex\":\"%s\",\"T\":\"%s\",\"RH\":\"%s\",\"Lat\":\"%s\",\"Long\":\"%s\",\"speed\":\"%s\"
    ,\"Occu\":\"%s\",\"source\":\"Node
    4\",\"local_time\":\"%s\",\"type\":\"data\"}",parsed_data4[11],parsed_data4[10],parsed_data4[7],parsed
    _data4[6],parsed_data4[13],parsed_data4[12],parsed_data4[1],parsed_data4[0],parsed_data4[3],parsed_dat
    a4[2],parsed_data4[9],parsed_data4[8],parsed_data4[4],parsed_data4[5],parsed_data4[14],parsed_data4[15
    ],parsed_data4[16],"10",parsed_data4[17]);

                    printf("SENDING NODE 4\n");
                    mqtt_publish_node_specific(MQTtest, node_num);
                    strcpy(n41_buffer, "");
                    strcpy(n42_buffer, "");
            }
            vTaskDelay(500/portTICK_PERIOD_MS);
    }
}

void upload_task(void *params){
    while(1){
            if(uxSemaphoreGetCount(semaphore) == TASK_COUNT){ //TASK_COUNT
                    xSemaphoreTake(semaphore, portMAX_DELAY);
                    xSemaphoreTake(semaphore, portMAX_DELAY);
                    xSemaphoreTake(semaphore, portMAX_DELAY);

                    xSemaphoreTake(semaphore, portMAX_DELAY);
                    xSemaphoreTake(semaphore, portMAX_DELAY);
                    xSemaphoreTake(semaphore, portMAX_DELAY);

                    xSemaphoreTake(semaphore, portMAX_DELAY);
                    xSemaphoreTake(semaphore, portMAX_DELAY);
                    xSemaphoreTake(semaphore, portMAX_DELAY);

                    printf("UPLOAD TASK\n");
                    //UPLOAD NODE 1 DATA
                    printf("DECODING NODE 1.1\n");
                    int node_num = 1;
                    char* token11 = strtok(n11_buffer, ",");
                    for (int i=0; i<8; i++){
                            data[i] = token11;
                            token11 = strtok(NULL, ",");
                    }
                    printf("DECODING NODE 1.2\n");
                    char* token12 = strtok(n12_buffer, ",");
                    for (int i=0; i<6; i++){
                            data[i+8] = token12;
                            token12 = strtok(NULL, ",");
                    }


//                  co_in = atof(data[10]);
//                  co2_in = atof(data[6]);
//                  no2_in = atof(data[12]);
//                  pm25_in = atof(data[0]);
//                  pm10_in = atof(data[2]);
//                  voc_in = atof(data[8]);
                    //driver_alert(); //UPDATE DRIVER ALERT
                    //snprintf(json_payload_1, 601,
    "{\"COraw\":\"%s\",\"COindex\":\"%s\",\"CO2raw\":\"%s\",\"CO2index\":\"%s\",\"NO2raw\":\"%s\",\"NO2ind
    ex\":\"%s\",\"PM25raw\":\"%s\",\"PM25index\":\"%s\",\"PM10raw\":\"%s\",\"PM10index\":\"%s\",\"VOCraw\"
    :\"%s\",\"VOCindex\":\"%s\",\"T\":\"%s\",\"RH\":\"%s\",\"source\":\"Node
    1\",\"local_time\":\"%s\",\"type\":\"data\"}",data[11],data[10],data[7],data[6],data[13],data[12],data
    [1],data[0],data[3],data[2],data[9],data[8],data[4],data[5],n13_buffer);

                    printf("SENDING NODE 1\n");
                    snprintf(json_payload_1, 650,
    "{\"COraw\":\"%s\",\"COindex\":\"%s\",\"CO2raw\":\"%s\",\"CO2index\":\"%s\",\"NO2raw\":\"%s\",\"NO2ind
    ex\":\"%s\",\"PM25raw\":\"%s\",\"PM25index\":\"%s\",\"PM10raw\":\"%s\",\"PM10index\":\"%s\",\"VOCraw\"
    :\"%s\",\"VOCindex\":\"%s\",\"T\":\"%s\",\"RH\":\"%s\",\"source\":\"Node
    1\",\"local_time\":\"%s\",\"type\":\"data\"}","0.01","1",data[7],data[6],"0.01","1",data[1],data[0],da
    ta[3],data[2],data[9],data[8],data[4],data[5],n13_buffer);
                    printf("MQTT\n");
                    mqtt_publish_node_specific(MQTtest, node_num);
                    printf("MQTT-P\n");
                    vTaskDelay(500/portTICK_PERIOD_MS);


                    //UPLOAD NODE 3 DATA
                    printf("DECODING NODE 3.1\n");
                    char* token31 = strtok(n31_buffer, ",");
                    node_num = 3;
                    for (int i=0; i<8; i++){
                            parsed_data3[i] = token31;
                            token31 = strtok(NULL, ",");
                    }
                    printf("DECODING NODE 3.2\n");
                    char* token32 = strtok(n32_buffer, ",");
```

```
                      for (int i=0; i<6; i++){
                               parsed_data3[i+8] = token32;
                               token32 = strtok(NULL, ",");
                      }
/*                    co_in_3 = atof(data3[10]);
                      co2_in = atof(data3[6]);
                      no2_in = atof(data3[12]);
                      pm25_in = atof(data3[0]);
                      pm10_in = atof(data3[2]);
                      voc_in = atof(data3[8]);*/
                      printf("SENDING NODE 3\n");
                      //snprintf(json_payload_3, 700,
    "{\"COraw\":\"%s\",\"COindex\":\"%s\",\"CO2raw\":\"%s\",\"CO2index\":\"%s\",\"NO2raw\":\"%s\",\"NO2ind
    ex\":\"%s\",\"PM25raw\":\"%s\",\"PM25index\":\"%s\",\"PM10raw\":\"%s\",\"PM10index\":\"%s\",\"VOCraw\"
    :\"%s\",\"VOCindex\":\"%s\",\"T\":\"%s\",\"RH\":\"%s\",\"source\":\"Node
    3\",\"local_time\":\"%s\",\"type\":\"data\"}",data3[11],data3[10],data3[7],data3[6],data3[13],data3[12
    ],data3[1],data3[0],data3[3],data3[2],data3[9],data3[8],data3[4],data3[5],n33_buffer);
                      snprintf(json_payload_3, 650,
    "{\"COraw\":\"%s\",\"COindex\":\"%s\",\"CO2raw\":\"%s\",\"CO2index\":\"%s\",\"NO2raw\":\"%s\",\"NO2ind
    ex\":\"%s\",\"PM25raw\":\"%s\",\"PM25index\":\"%s\",\"PM10raw\":\"%s\",\"PM10index\":\"%s\",\"VOCraw\"
    :\"%s\",\"VOCindex\":\"%s\",\"T\":\"%s\",\"RH\":\"%s\",\"source\":\"Node
    3\",\"local_time\":\"%s\",\"type\":\"data\"}","0.01","1",parsed_data3[7],parsed_data3[6],"0.01","1",pa
    rsed_data3[1],parsed_data3[0],parsed_data3[3],parsed_data3[2],parsed_data3[9],parsed_data3[8],parsed_d
    ata3[4],parsed_data3[5],n33_buffer);
                      printf("MQTT\n");
                      mqtt_publish_node_specific(MQTtest, node_num);
                      printf("MQTT-P\n");
                      vTaskDelay(500/portTICK_PERIOD_MS);


                      //UPLOAD NODE 4 DATA
                      node_num = 4;
                      printf("DECODING NODE 4.1\n");
                      char* token41 = strtok(n41_buffer, ",");
                      char* data4[20];
                      for (int i=0; i<8; i++){
                               data4[i] = token41;
                               token41 = strtok(NULL, ",");
                      }
                      printf("DECODING NODE 4.2\n");
                      char* token42 = strtok(n42_buffer, ",");
                      for (int i=0; i<6; i++){
                               data4[i+8] = token42;
                               token42 = strtok(NULL, ",");
                      }
                      printf("DECODING NODE 4.3\n");
                      char* token43 = strtok(n43_buffer, ",");
                      for (int i=0; i<4 ; i++){
                               data4[i+14] = token43;
                               token43 = strtok(NULL, ",");
                      }
                      //data4[16] is SPEED - REMOVED for now
/*                    co_out = atof(data4[10]);
                      co2_out = atof(data4[6]);
                      no2_out = atof(data4[12]);
                      pm25_out = atof(data4[0]);
                      pm10_out = atof(data4[2]);
                      voc_out = atof(data4[8]);*/
                      //speed = atof(data4[16]);
                      //lat_decimal = atof(data4[14]);
                      //long_decimal = atof(data4[15]);
                      //occu_task();

                      printf("With GPS");
                      snprintf(json_payload_4, 650,
    "{\"COraw\":\"%s\",\"COindex\":\"%s\",\"CO2raw\":\"%s\",\"CO2index\":\"%s\",\"NO2raw\":\"%s\",\"NO2ind
    ex\":\"%s\",\"PM25raw\":\"%s\",\"PM25index\":\"%s\",\"PM10raw\":\"%s\",\"PM10index\":\"%s\",\"VOCraw\"
    :\"%s\",\"VOCindex\":\"%s\",\"T\":\"%s\",\"RH\":\"%s\",\"Lat\":\"%s\",\"Long\":\"%s\",\"speed\":\"%s\"
    ,\"Occu\":\"%s\",\"source\":\"Node
    4\",\"local_time\":\"%s\",\"type\":\"data\"}",data4[11],data4[10],data4[7],data4[6],data4[13],data4[12
    ],data4[1],data4[0],data4[3],data4[2],data4[9],data4[8],data4[4],data4[5],data4[14],data4[15],data4[16
    ],occu_cleaned,data4[17]);

                      printf("MQTT\n");
                      mqtt_publish_node_specific(MQTtest, node_num);
                      printf("MQTT-P\n");
                      vTaskDelay(500/portTICK_PERIOD_MS);

                      //RESET FLAGS (CONSIDER CHANGING TO BOOL)
                      node1_data1_flag = 0;
                      node1_data2_flag = 0;
                      node1_data3_flag = 0;
                      node3_data1_flag = 0;
                      node3_data2_flag = 0;
                      node3_data3_flag = 0;
                      node4_data1_flag = 0;
                      node4_data2_flag = 0;
                      node4_data3_flag = 0;
```

```
                    vTaskDelay(1000/portTICK_PERIOD_MS);
            }
            printf("%d %d %d %d %d %d %d %d %d %d\n", uxSemaphoreGetCount(semaphore), node1_data1_flag,
    node1_data2_flag, node1_data3_flag, node3_data1_flag, node3_data2_flag, node3_data3_flag,
    node4_data1_flag, node4_data2_flag, node4_data3_flag);
            vTaskDelay(1000/portTICK_PERIOD_MS);
    }
}
static void mqtt_event_handler(void *handler_args, esp_event_base_t base, int32_t event_id, void
    *event_data)
{
    ESP_LOGD(TAG_MQTT, "Event dispatched from event loop base=%s, event_id=%" PRIi32, base, event_id);
    esp_mqtt_event_handle_t event = event_data;
    esp_mqtt_client_handle_t client = event->client;
    int msg_id;
    switch ((esp_mqtt_event_id_t)event_id) {
    case MQTT_EVENT_CONNECTED:
        ESP_LOGI(TAG_MQTT, "MQTT_EVENT_CONNECTED");
        msg_id = esp_mqtt_client_subscribe(client, "/topic/qos0", 0);
        ESP_LOGI(TAG_MQTT, "sent subscribe successful, msg_id=%d", msg_id);
        break;
    case MQTT_EVENT_DISCONNECTED:
        ESP_LOGI(TAG_MQTT, "MQTT_EVENT_DISCONNECTED");
        break;
    case MQTT_EVENT_SUBSCRIBED:
    ESP_LOGI(TAG_MQTT, "MQTT_EVENT_SUBSCRIBED, msg_id=%d", event->msg_id);
        break;
    case MQTT_EVENT_UNSUBSCRIBED:
        ESP_LOGI(TAG_MQTT, "MQTT_EVENT_UNSUBSCRIBED, msg_id=%d", event->msg_id);
        break;
    case MQTT_EVENT_PUBLISHED:
        ESP_LOGI(TAG_MQTT, "MQTT_EVENT_PUBLISHED, msg_id=%d", event->msg_id);
        break;
    case MQTT_EVENT_DATA:
        ESP_LOGI(TAG_MQTT, "MQTT_EVENT_DATA");
        break;
    case MQTT_EVENT_ERROR:
        ESP_LOGI(TAG_MQTT, "MQTT_EVENT_ERROR");
        if (event->error_handle->error_type == MQTT_ERROR_TYPE_TCP_TRANSPORT) {
            ESP_LOGI(TAG_MQTT, "Last error code reported from esp-tls: 0x%x",
    event->error_handle->esp_tls_last_esp_err);
            ESP_LOGI(TAG_MQTT, "Last tls stack error number: 0x%x",
    event->error_handle->esp_tls_stack_err);
            ESP_LOGI(TAG_MQTT, "Last captured errno : %d (%s)",
    event->error_handle->esp_transport_sock_errno,
                    strerror(event->error_handle->esp_transport_sock_errno));
        } else if (event->error_handle->error_type == MQTT_ERROR_TYPE_CONNECTION_REFUSED) {
            ESP_LOGI(TAG_MQTT, "Connection refused error: 0x%x",
    event->error_handle->connect_return_code);
        } else {
            ESP_LOGW(TAG_MQTT, "Unknown error type: 0x%x", event->error_handle->error_type);
        }
        break;
    default:
        ESP_LOGI(TAG_MQTT, "Other event id:%d", event->event_id);
        break;
    }
}

static void mqtt_app_start(esp_mqtt_client_handle_t client)
{
    esp_mqtt_client_register_event(client, ESP_EVENT_ANY_ID, mqtt_event_handler, NULL);
    esp_mqtt_client_start(client);
}

void update_time_attr (void *params){
    while(1){
            Get_current_date_time(Current_Date_Time);
            //UPDATE TIME ATTRIBUTE
            esp_zb_zcl_attr_t *value_r = esp_zb_zcl_get_attribute(COORDINATOR_ENDPOINT, TIME_CLUSTER_ID,
    ESP_ZB_ZCL_CLUSTER_SERVER_ROLE, TIME_ATTR_ID);
            memcpy(value_r->data_p, &Current_Date_Time, sizeof(Current_Date_Time));
            vTaskDelay(1000/portTICK_PERIOD_MS);
    }
}

static void bdb_start_top_level_commissioning_cb(uint8_t mode_mask)
{
    ESP_ERROR_CHECK(esp_zb_bdb_start_top_level_commissioning(mode_mask));
}

void esp_zb_app_signal_handler(esp_zb_app_signal_t *signal_struct)
{
    uint32_t *p_sg_p       = signal_struct->p_app_signal;
    esp_err_t err_status = signal_struct->esp_err_status;
    esp_zb_app_signal_type_t sig_type = *p_sg_p;
    esp_zb_zdo_signal_device_annce_params_t *dev_annce_params = NULL;
    esp_zb_zdo_signal_macsplit_dev_boot_params_t *rcp_version = NULL;
    switch (sig_type) {
    case ESP_ZB_ZDO_SIGNAL_SKIP_STARTUP:
        ESP_LOGI(TAG, "Zigbee stack initialized");
        esp_zb_bdb_start_top_level_commissioning(ESP_ZB_BDB_MODE_INITIALIZATION);
        break;
    case ESP_ZB_MACSPLIT_DEVICE_BOOT:
        ESP_LOGI(TAG, "Zigbee rcp device booted");
        rcp_version = (esp_zb_zdo_signal_macsplit_dev_boot_params_t
    *)esp_zb_app_signal_get_params(p_sg_p);
        ESP_LOGI(TAG, "Running RCP Version: %s", rcp_version->version_str);
        break;
    case ESP_ZB_BDB_SIGNAL_DEVICE_FIRST_START:
    case ESP_ZB_BDB_SIGNAL_DEVICE_REBOOT:
        if (err_status == ESP_OK) {
            ESP_LOGI(TAG, "Start network formation");
            esp_zb_bdb_start_top_level_commissioning(ESP_ZB_BDB_MODE_NETWORK_FORMATION);
        } else {
            ESP_LOGE(TAG, "Failed to initialize Zigbee stack (status: %s)", esp_err_to_name(err_status));
        }
        break;
    case ESP_ZB_BDB_SIGNAL_FORMATION:
        if (err_status == ESP_OK) {
            esp_zb_ieee_addr_t ieee_address;
            esp_zb_get_long_address(ieee_address);
            ESP_LOGI(TAG, "Formed network successfully (ieee_address:
    %02x:%02x:%02x:%02x:%02x:%02x:%02x:%02x, PAN ID: 0x%04hx, Channel:%d, Short Address: 0x%04hx)",
                     ieee_address[7], ieee_address[6], ieee_address[5], ieee_address[4],
                     ieee_address[3], ieee_address[2], ieee_address[1], ieee_address[0],
                     esp_zb_get_pan_id(), esp_zb_get_current_channel(), esp_zb_get_short_address());
            ESP_LOGI(TAG, "Starting time cluster");
            xTaskCreate(update_time_attr, "Time Cluster", 4096, NULL, 5, &timer_handle);
            esp_zb_bdb_start_top_level_commissioning(ESP_ZB_BDB_MODE_NETWORK_STEERING);
        } else {
            ESP_LOGI(TAG, "Restart network formation (status: %s)", esp_err_to_name(err_status));
            esp_zb_scheduler_alarm((esp_zb_callback_t)bdb_start_top_level_commissioning_cb,
    ESP_ZB_BDB_MODE_NETWORK_FORMATION, 1000);
        }
        break;
    case ESP_ZB_BDB_SIGNAL_STEERING:
        if (err_status == ESP_OK) {
            ESP_LOGI(TAG, "Network steering started");
            ESP_LOGI(TAG, "Uploading to CARE Database started");
            //xTaskCreate(upload_task, "uploading", 1024*9, NULL, 5, &upload_handle);
        }
        break;
    case ESP_ZB_ZDO_SIGNAL_DEVICE_ANNCE:
        dev_annce_params = (esp_zb_zdo_signal_device_annce_params_t
    *)esp_zb_app_signal_get_params(p_sg_p);
        ESP_LOGI(TAG, "New device commissioned or rejoined (short: 0x%04hx)",
    dev_annce_params->device_short_addr);
        break;
    default:
        ESP_LOGI(TAG, "ZDO signal: %s (0x%x), status: %s", esp_zb_zdo_signal_to_string(sig_type),
    sig_type,
                 esp_err_to_name(err_status));
        break;
    }
}

static esp_err_t zb_attribute_reporting_handler(const esp_zb_zcl_report_attr_message_t *message)
{
    ESP_RETURN_ON_FALSE(message, ESP_FAIL, TAG, "Empty message");
    ESP_RETURN_ON_FALSE(message->status == ESP_ZB_ZCL_STATUS_SUCCESS, ESP_ERR_INVALID_ARG, TAG, "Received
    message: error status(%d)",
                        message->status);
    ESP_LOGI(TAG, "Report from address(0x%x) src_ep(%d) to dst_ep(%d) cluster(0x%x)",
    message->src_address.u.short_addr,
             message->src_endpoint, message->dst_endpoint, message->cluster);
    ESP_LOGI(TAG, "Information: attribute(0x%x), type(0x%x), value(%s)\n", message->attribute.id,
    message->attribute.data.type,
             (char *)message->attribute.data.value);
    if (message->src_endpoint == 1) { //INDOOR SOURCE NODE 1
             if (message->attribute.id == 0){ //PM2.5, PM10, TEMPERATURE, REL. HUMIDITY, CO2, VOC, CO
                      snprintf(n11_buffer, sizeof(n11_buffer), "%s", (char*
    )message->attribute.data.value);
             }
             else if (message->attribute.id == 1){ //NO2, DateTime
                      snprintf(n12_buffer, sizeof(n12_buffer), "%s", (char*
    )message->attribute.data.value);
             }
    }
    else if (message->src_endpoint == 3) { //"OUTDOOR" SOURCE NODE 4
             if (message->attribute.id == 0){ //PM2.5, PM10, TEMPERATURE, REL. HUMIDITY, CO2, VOC, CO
                      snprintf(n31_buffer, sizeof(n31_buffer), "%s", (char*
    )message->attribute.data.value);
             }
             else if (message->attribute.id == 1){ //NO2, DateTime
```

```
                    snprintf(n32_buffer, sizeof(n32_buffer), "%s", (char*
    )message->attribute.data.value);
            }
    }
    else if (message->src_endpoint == 4) { //"OUTDOOR" SOURCE NODE 4
            if (message->attribute.id == 0){ //PM2.5, PM10, TEMPERATURE, REL. HUMIDITY, CO2, VOC, CO
                    snprintf(n41_buffer, sizeof(n41_buffer), "%s", (char*
    )message->attribute.data.value);
            }
            else if (message->attribute.id == 1){ //NO2, LAT, LONG, SPEED, DateTime
                    snprintf(n42_buffer, sizeof(n42_buffer), "%s", (char*
    )message->attribute.data.value);
            }
    }
//    if (message->src_endpoint == 1) { //INDOOR SOURCE NODE 1
//          if (node1_data1_flag == 1 && node1_data2_flag == 1 && node1_data3_flag == 1){
//                  return ESP_OK; //SKIP
//          }
//          else{
//                  if (message->attribute.id == 0){ //PM2.5, PM10, TEMPERATURE, REL. HUMIDITY, CO2
//                          if (node1_data1_flag == 1){
//                                  return ESP_OK; //SKIP
//                          }
//                          else{
//                                  snprintf(n11_buffer, 50, "%s", (char*
    )message->attribute.data.value);
//                                  xSemaphoreGive(semaphore);
//                                  //xSemaphoreGive(node1_semaphore);
//                                  node1_data1_flag = 1;
//                          }
//                  }
//                  else if (message->attribute.id == 1){ //VOC, CO, NO2
//                          if (node1_data2_flag == 1){
//                                  return ESP_OK; //SKIP
//                          }
//                          else{
//                                  snprintf(n12_buffer, 50, "%s", (char*
    )message->attribute.data.value);
//                                  xSemaphoreGive(semaphore);
//                                  //xSemaphoreGive(node1_semaphore);
//                                  node1_data2_flag = 1;
//                          }
//                  }
//                  else if(message->attribute.id == 2){ //TIME
//                          if (node1_data3_flag == 1){
//                                  return ESP_OK; //SKIP
//                          }
//                          else{
//                                  snprintf(n13_buffer, 50, "%s", (char*
    )message->attribute.data.value);
//                                  xSemaphoreGive(semaphore);
//                                  //xSemaphoreGive(node1_semaphore);
//                                  node1_data3_flag = 1;
//                          }
//                  }
//                  else{
//                          ESP_LOGI(TAG, "Unrecognized Attribute ID: %d", message->attribute.id);
//                  }
//          }
//    }    else if (message->src_endpoint == 2){
//          if (node2_data1_flag == 1 && node2_data2_flag == 1){
//                  return ESP_OK; //SKIP
//          }
//          else{
//                  if (message->attribute.id == 0){ //PM2.5, PM10, TEMPERATURE, REL. HUMIDITY, CO2
//                          if (node2_data1_flag == 1){
//                                  return ESP_OK; //SKIP
//                          }
//                          else{
//                                  snprintf(n21_buffer, 50, "%s", (char*
    )message->attribute.data.value);
//                                  xSemaphoreGive(semaphore);
//                                  node2_data1_flag = 1;
//
//                          }
//                  }
//                  else if (message->attribute.id == 1){ //VOC, CO, NO2, LAT, LONG
//                          if (node2_data2_flag == 1){
//                                  return ESP_OK; //SKIP
//                          }
//                          else{
//                                  snprintf(n22_buffer, 50, "%s", (char*
    )message->attribute.data.value);
//                                  xSemaphoreGive(semaphore);
//                                  node2_data2_flag = 1;
//                          }
//                  }
//                  else{
```

```
//                              ESP_LOGI(TAG, "Unrecognized Attribute ID: %d", message->attribute.id);
//                      }
//              }
//      }
//      else if (message->src_endpoint == 3){
//              if (node3_data1_flag == 1 && node3_data2_flag == 1 && node3_data3_flag == 1){
//                      return ESP_OK; //SKIP
//              }
//              else{
//                      if (message->attribute.id == 0){ //PM2.5, PM10, TEMPERATURE, REL. HUMIDITY, CO2
//                              if (node3_data1_flag == 1){
//                                      return ESP_OK; //SKIP
//                              }
//                              else{
//                                      snprintf(n31_buffer, 50, "%s", (char*
    )message->attribute.data.value);
//                                      xSemaphoreGive(semaphore);
//                                      //xSemaphoreGive(node3_semaphore);
//                                      node3_data1_flag = 1;
//
//                              }
//                      }
//                      else if (message->attribute.id == 1){ //VOC, CO, NO2
//                              if (node3_data2_flag == 1){
//                                      return ESP_OK; //SKIP
//                              }
//                              else{
//                                      snprintf(n32_buffer, 50, "%s", (char*
    )message->attribute.data.value);
//                                      xSemaphoreGive(semaphore);
//                                      //xSemaphoreGive(node3_semaphore);
//                                      node3_data2_flag = 1;
//                              }
//                      }
//                      else if(message->attribute.id == 2){ //TIME
//                              if (node3_data3_flag == 1){
//                                      return ESP_OK; //SKIP
//                              }
//                              else{
//                                      snprintf(n33_buffer, 50, "%s", (char*
    )message->attribute.data.value);
//                                      xSemaphoreGive(semaphore);
//                                      //xSemaphoreGive(node1_semaphore);
//                                      node3_data3_flag = 1;
//                              }
//                      }
//                      else{
//                              ESP_LOGI(TAG, "Unrecognized Attribute ID: %d", message->attribute.id);
//                      }
//              }
//      }
//      else if (message->src_endpoint == 4){
//              if (node4_data1_flag == 1 && node4_data2_flag == 1 && node4_data3_flag == 1){
//                      return ESP_OK; //SKIP
//              }
//              else{
//                      if (message->attribute.id == 0){ //PM2.5, PM10, TEMPERATURE, REL. HUMIDITY, CO2
//                              if (node4_data1_flag == 1){
//                                      return ESP_OK; //SKIP
//                              }
//                              else{
//                                      snprintf(n41_buffer, 50, "%s", (char*
    )message->attribute.data.value);
//                                      xSemaphoreGive(semaphore);
//                                      //xSemaphoreGive(node4_semaphore);
//                                      node4_data1_flag = 1;
//
//                              }
//                      }
//                      else if (message->attribute.id == 1){ //VOC, CO, NO2
//                              if (node4_data2_flag == 1){
//                                      return ESP_OK; //SKIP
//                              }
//                              else{
//                                      snprintf(n42_buffer, 50, "%s", (char*
    )message->attribute.data.value);
//                                      xSemaphoreGive(semaphore);
//                                      //xSemaphoreGive(node4_semaphore);
//                                      node4_data2_flag = 1;
//                              }
//                      }
//                      else if (message->attribute.id == 2){ //LAT, LONG, SPEED
//                              if (node4_data3_flag == 1){
//                                      return ESP_OK; //SKIP
//                              }
//                              else{
//                                      snprintf(n43_buffer, 50, "%s", (char*
    )message->attribute.data.value);
//                                      xSemaphoreGive(semaphore);
//                                      //xSemaphoreGive(node4_semaphore);
//                                      node4_data3_flag = 1;
//                             }
//                      }
//                      else{
//                               ESP_LOGI(TAG, "Unrecognized Attribute ID: %d", message->attribute.id);
//                      }
//          }
//    }
    return ESP_OK;
}

static esp_err_t zb_attribute_handler(const esp_zb_zcl_set_attr_value_message_t *message)
{
    esp_err_t ret = ESP_OK;
    float light_state = 0;
    ESP_RETURN_ON_FALSE(message, ESP_FAIL, TAG, "Empty message");
    ESP_RETURN_ON_FALSE(message->info.status == ESP_ZB_ZCL_STATUS_SUCCESS, ESP_ERR_INVALID_ARG, TAG,
     "Received message: error status(%d)",
                        message->info.status);
    ESP_LOGI(TAG, "Received message: endpoint(%d), cluster(0x%x), attribute(0x%x), data size(%d)",
     message->info.dst_endpoint, message->info.cluster,
             message->attribute.id, message->attribute.data.size);
    if (message->info.dst_endpoint == COORDINATOR_ENDPOINT) {
        if (message->info.cluster == SENSOR_DATA_CLUSTER_ID) {
            if (message->attribute.id == SENSOR_DATA_ATTR_ID && message->attribute.data.type ==
     ESP_ZB_ZCL_ATTR_TYPE_CHAR_STRING) {
                light_state = message->attribute.data.value ? *(char *)message->attribute.data.value :
     light_state;
                ESP_LOGI(TAG, "Data received: %.2f", light_state);
                ESP_LOGI(TAG, "Light sets to %s", light_state ? "On" : "Off");
                //light_driver_set_power(light_state);
            }
        }
    }
    return ret;
}

static esp_err_t zb_action_handler(esp_zb_core_action_callback_id_t callback_id, const void *message)
{
    esp_err_t ret = ESP_OK;
    switch (callback_id) {
    case ESP_ZB_CORE_SET_ATTR_VALUE_CB_ID:
        ret = zb_attribute_handler((esp_zb_zcl_set_attr_value_message_t *)message);
        break;
    case ESP_ZB_CORE_REPORT_ATTR_CB_ID:
        ret = zb_attribute_reporting_handler((esp_zb_zcl_report_attr_message_t *)message);
        break;
    case ESP_ZB_CORE_CMD_REPORT_CONFIG_RESP_CB_ID:
        ret = zb_attribute_reporting_handler((esp_zb_zcl_cmd_config_report_resp_message_t *)message);
        break;
    default:
        ESP_LOGW(TAG, "Receive Zigbee action(0x%x) callback", callback_id);
        break;
    }
    return ret;
}

static void esp_zb_task(void *pvParameters)
{
    /* initialize Zigbee stack with Zigbee coordinator config */
    esp_zb_cfg_t zb_nwk_cfg = ESP_ZB_ZC_CONFIG();
    esp_zb_init(&zb_nwk_cfg);

    const uint8_t attr_access = ESP_ZB_ZCL_ATTR_ACCESS_REPORTING | ESP_ZB_ZCL_ATTR_ACCESS_READ_WRITE;

     esp_zb_attribute_list_t *esp_zb_sensordata_cluster =
     esp_zb_zcl_attr_list_create(SENSOR_DATA_CLUSTER_ID);
     ESP_ERROR_CHECK(esp_zb_custom_cluster_add_custom_attr(esp_zb_sensordata_cluster, SENSOR_DATA_ATTR_ID,
     ESP_ZB_ZCL_ATTR_TYPE_CHAR_STRING, attr_access, &sample_char));
     esp_zb_custom_cluster_add_custom_attr(esp_zb_sensordata_cluster, SENSOR_DATA_ATTR_ID_2,
     ESP_ZB_ZCL_ATTR_TYPE_CHAR_STRING, attr_access, &sample_char);

     esp_zb_attribute_list_t *esp_zb_time_cluster = esp_zb_zcl_attr_list_create(TIME_CLUSTER_ID);
     esp_zb_custom_cluster_add_custom_attr(esp_zb_time_cluster, TIME_ATTR_ID,
     ESP_ZB_ZCL_ATTR_TYPE_CHAR_STRING, attr_access, &sample_char);

     esp_zb_cluster_list_t *esp_zb_cluster_list = esp_zb_zcl_cluster_list_create();
     esp_zb_cluster_list_add_custom_cluster(esp_zb_cluster_list, esp_zb_sensordata_cluster,
     ESP_ZB_ZCL_CLUSTER_CLIENT_ROLE);
     esp_zb_cluster_list_add_custom_cluster(esp_zb_cluster_list, esp_zb_time_cluster,
     ESP_ZB_ZCL_CLUSTER_SERVER_ROLE);


     esp_zb_ep_list_t *esp_zb_ep_list = esp_zb_ep_list_create();
     esp_zb_ep_list_add_ep(esp_zb_ep_list, esp_zb_cluster_list, COORDINATOR_ENDPOINT,
     ESP_ZB_AF_HA_PROFILE_ID, ESP_ZB_HA_ON_OFF_LIGHT_DEVICE_ID);
     esp_zb_device_register(esp_zb_ep_list);

     //esp_zb_device_register(esp_zb_ep_node2_list);
    esp_zb_core_action_handler_register(zb_action_handler);
    /* initiate Zigbee Stack start without zb_send_no_autostart_signal auto-start */
    esp_zb_set_primary_network_channel_set(ESP_ZB_PRIMARY_CHANNEL_MASK);
    ESP_ERROR_CHECK(esp_zb_start(false));
    esp_zb_main_loop_iteration();
}

void app_main(void)
{
     init_uart();
    semaphore = xSemaphoreCreateCounting(TASK_COUNT, 0);
    node1_semaphore = xSemaphoreCreateCounting(4, 0);
    node3_semaphore = xSemaphoreCreateCounting(4, 0);
    node4_semaphore = xSemaphoreCreateCounting(4, 0);

    esp_log_level_set("*", ESP_LOG_INFO);
    esp_log_level_set("esp-tls", ESP_LOG_VERBOSE);
    esp_log_level_set("MQTT_CLIENT", ESP_LOG_VERBOSE);
    esp_log_level_set("MQTT_EXAMPLE", ESP_LOG_VERBOSE);
    esp_log_level_set("TRANSPORT_BASE", ESP_LOG_VERBOSE);
    esp_log_level_set("TRANSPORT", ESP_LOG_VERBOSE);
    esp_log_level_set("OUTBOX", ESP_LOG_VERBOSE);

     esp_zb_platform_config_t config = {
        .radio_config = ESP_ZB_DEFAULT_RADIO_CONFIG(),
        .host_config = ESP_ZB_DEFAULT_HOST_CONFIG(),
    };
    ESP_ERROR_CHECK(esp_zb_platform_config(&config));
    ESP_ERROR_CHECK(nvs_flash_init());
    ESP_ERROR_CHECK(esp_netif_init());
    ESP_ERROR_CHECK(esp_event_loop_create_default());
#if CONFIG_ESP_CONSOLE_USB_SERIAL_JTAG
    ESP_ERROR_CHECK(esp_zb_gateway_console_init());
#endif
#if CONFIG_EXAMPLE_CONNECT_WIFI
    wifi_connection();
    vTaskDelay(8000 / portTICK_PERIOD_MS);
    Set_SystemTime_SNTP();
    const esp_mqtt_client_config_t mqtt_cfg = {
     //broker URL or URI
        .broker = {
            .address.uri = "URI HERE",
            .verification.certificate = (const char *)mqtt_eclipseprojects_io_pem_start
        },
        //this is where I put the credentials
     //I added this
     .credentials = {
            .username="USERNAME",
             .authentication = {
                .password="PASSWORD"
            }
        },

    };
    MQTtest = esp_mqtt_client_init(&mqtt_cfg);
    mqtt_app_start(MQTtest);
#if CONFIG_ESP_COEX_SW_COEXIST_ENABLE
    ESP_ERROR_CHECK(esp_wifi_set_ps(WIFI_PS_MIN_MODEM));
    coex_enable();
    coex_schm_status_bit_set(1, 1);
#else
    ESP_ERROR_CHECK(esp_wifi_set_ps(WIFI_PS_NONE));
#endif
#endif
    //xTaskCreate(occu_task, "uart test", 4096, NULL, 5, NULL);
    driver_alert_init();
    //xTaskCreate(upload_task, "uploading", 1024*9, NULL, 5, &upload_handle);
    xTaskCreate(esp_zb_task, "Zigbee_main", 4096*2, NULL, 5, &zigbee_handle);
    xTaskCreate(upload_node1, "Node 1 Upload", 4096, NULL, 5, NULL);
    xTaskCreate(upload_node3, "Node 3 Upload", 4096, NULL, 5, NULL);
    xTaskCreate(upload_node4, "Node 4 Upload", 4096, NULL, 5, NULL);

}
```

Appendix H
Code for Thread System - Node

Template Source Code:
https://github.com/espressif/esp-idf/tree/master/examples/openthread/ot_cli

```c
#include <stdio.h>
#include <unistd.h>
#include <string.h>

#include "esp_err.h"
#include "esp_event.h"
#include "esp_log.h"
#include "esp_netif.h"
#include "esp_netif_types.h"
#include "esp_openthread.h"
#include "esp_openthread_cli.h"
#include "esp_openthread_lock.h"
#include "esp_openthread_netif_glue.h"
#include "esp_openthread_types.h"
#include "esp_ot_config.h"
#include "esp_vfs_eventfd.h"
#include "driver/uart.h"
#include "freertos/FreeRTOS.h"
#include "freertos/task.h"
#include "hal/uart_types.h"
#include "nvs_flash.h"
#include "openthread/cli.h"
#include "openthread/instance.h"
#include "openthread/logging.h"
#include "openthread/tasklet.h"

// add necessary libraries

#include "openthread/udp.h"
#include "driver/i2c.h"
#include "driver/gpio.h"
#include <sys/unistd.h>
#include <sys/stat.h>
#include "esp_vfs_fat.h"
#include "sdmmc_cmd.h"
#include "driver/sdspi_host.h"
#include <stdlib.h>
#include "soc/soc_caps.h"
#include "esp_adc/adc_oneshot.h"
#include "esp_adc/adc_cali.h"
#include "esp_adc/adc_cali_scheme.h"
#include <math.h>
#include "driver/uart.h"

#if CONFIG_OPENTHREAD_CLI_ESP_EXTENSION
#include "esp_ot_cli_extension.h"
#endif // CONFIG_OPENTHREAD_CLI_ESP_EXTENSION

//OpenThread
#define TAG "OpenThread Indoor Node 1"
#define UDP_PORT 2222
// #define UDP_PORT 1111 // designated UDP Port in BR for node 1
#define UDP_PORT_nodes 1234
otUdpSocket udpSocket;


char Rx_Date_Time[20];
char Current_Date_Time[20];
int latitude_decimal;
int longitude_decimal;
SemaphoreHandle_t semaphore;
TaskHandle_t sen_handle, mics_handle, dummy, sgp_handle, finish_handle, uart_handle, ot_handle;

static const char *TAG_SDCARD = "SD_CARD";
int heated = 0;
#define TASK_COUNT 3

//GPIO Pins for LED
#define RED_LED 22
#define ORANGE_LED 23
#define YELLOW_LED 27
#define GREEN_LED 26
#define RECIRC_LED 30 //CHANGE ACCORDINGLY
#define INTAKE_LED 30

//I2C0 for SGP30
#define I2C_PORT_0 0
```

```
#define I2C0_SCL 10
#define I2C0_SDA 11

//I2C1 for SEN55 and MiCS-4514
#define I2C_PORT_1 1
#define I2C1_SCL 12
#define I2C1_SDA 8

//For Sensor I2C Addresses
#define SEN55_ADDR 0x69
#define SGP30_ADDR 0x58

//General I2C parameters
#define I2C_FREQUENCY 100000

//Global Variables for Sensor Readings
static float pm25, pm10, rh, temp, co, no2, co2, voc;
int no2_adc, co_adc;
float pm25_score, pm10_score, co_score, no2_score, co2_score, voc_score, iaq, speed_kmh;
float speed_kmh = 0;
char* location[4];
char* velocity[10];
double lat_deg = 0;
double long_deg = 0;
char* speed_str = "0";

//SPI for SD Card Module
#define PIN_NUM_MISO  0
#define PIN_NUM_MOSI  5
#define PIN_NUM_CLK   4
#define PIN_NUM_CS    1
#define MAX_SIZE 220
#define MOUNT_POINT "/sdcard"
int first_write = 1;

//ADC1 for MiCS-4514
adc_channel_t ADC1_CH1_RED = ADC_CHANNEL_1; // For RED ADC Readings
adc_channel_t ADC1_CH2_NOX = ADC_CHANNEL_2; // For NOX ADC Readings
#define PREHEAT_PIN 13 // Pin to pre-heat the module

//UART1 for Ublox Neo 6M V2
#define TX_pin 26
#define RX_pin 27

//static const char *TAG = "ESP_ZB_CLIENT_4";
//char data1[50] = "150,1000.00,150,1000.00,85.00,100.00,150,60000.00";
//char data2[50] = "150,10000.00,150,1000.00,150,10.00,90.000,121.000";
//char data3[50] = "00.000000,000.000000,00.00";

// char data_payload[150] =
    "1,150,1000.00,150,1000.00,85.00,100.00,150,60000.00,150,10000.00,150,1000.00,150,10.00,90.000,121.000
    ,00.000000,000.000000,00.00";
char data_payload[500] =
    "{\"COraw\":\"1000.00\",\"COindex\":\"150\",\"CO2raw\":\"60000.00\",\"CO2index\":\"150\",\"NO2raw\":\"
    1000.00\",\"NO2index\":\"150\",\"PM25raw\":\"1000.00\",\"PM25index\":\"150\",\"PM10raw\":\"1000.00\",\
    "PM10index\":\"150\",\"VOCraw\":\"10000.00\",\"VOCindex\":\"150\",\"T\":\"100.00\",\"RH\":\"100.00\",\
    "source\":\"Node 1\",\"local_time\":\"2024-05-05T12:12:12\",\"type\":\"data\"}";

//OpenThread
static esp_netif_t *init_openthread_netif(const esp_openthread_platform_config_t *config)
{
    esp_netif_config_t cfg = ESP_NETIF_DEFAULT_OPENTHREAD();
    esp_netif_t *netif = esp_netif_new(&cfg);
    assert(netif != NULL);
    ESP_ERROR_CHECK(esp_netif_attach(netif, esp_openthread_netif_glue_init(config)));

    return netif;
}

void udp_rx_cb(void *aContext, otMessage *aMessage, const otMessageInfo *aMessageInfo)
{
    uint16_t payloadLength = otMessageGetLength(aMessage) - otMessageGetOffset(aMessage);
    char buf[payloadLength+1];
    otMessageRead(aMessage, otMessageGetOffset(aMessage),buf, payloadLength);
    buf[payloadLength]='\0';
    // ESP_LOGI(TAG, "UDP received successfully");
    // ESP_LOGI(TAG, "UDP message: %s", buf);
    // printf("%s\n",buf);
    snprintf(Rx_Date_Time, sizeof(Rx_Date_Time), "%s", buf);
    // Rx_Date_Time = buf;
}

static void udp_init(void)
{
    otInstance * thread_instance = esp_openthread_get_instance();
    otSockAddr bind_info;
//    otUdpSocket udpSocket; // Declare this globally
    otNetifIdentifier netif = OT_NETIF_THREAD;
    memset(&bind_info, 0, sizeof(otSockAddr));
    otIp6AddressFromString("::", &bind_info.mAddress);
    bind_info.mPort = UDP_PORT_nodes;

    otError error = otUdpOpen(thread_instance, &udpSocket, udp_rx_cb, NULL);
    if (error != OT_ERROR_NONE)
    {
        ESP_LOGE(TAG, "UDP open error (error %d:%s)", error, otThreadErrorToString(error));
    } else
    {
        ESP_LOGI(TAG, "UDP initialized");
    }

    error = otUdpBind(thread_instance, &udpSocket, &bind_info, netif);
    if (error != OT_ERROR_NONE)
    {
        ESP_LOGE(TAG, "UDP bind error (error %d:%s)", error, otThreadErrorToString(error));
    } else{
        ESP_LOGI(TAG, "UDP binded");
    }
}

static void udp_send(void) // ff02:1, 64000, ff03:1, 2222
{
     // esp_openthread_lock_acquire(portMAX_DELAY);

     otMessageInfo messageInfo;
    otMessageSettings msgSettings;
    // otUdpSocket udpSocket;
    msgSettings.mLinkSecurityEnabled = true;
    msgSettings.mPriority = 1;
    otIp6AddressFromString("ff03::1", &messageInfo.mPeerAddr);
    messageInfo.mPeerPort = UDP_PORT;
    otMessage * message = otUdpNewMessage(esp_openthread_get_instance(), &msgSettings);
     // const char * buf = "hello";
    // const char * buf =
     "id1,18,5.90,6,5.90,32.03,49.86,0,400.00,0,0.00,2,1.77,0,0.07,14.699734,121.723847,23.55";
    const char * buf = data_payload;
    // char * buf = "";
    // snprintf(buf, sizeof(data_payload), "%s", data_payload);

    otError error = otMessageAppend(message, buf, (uint16_t) strlen(buf));
    if (error != OT_ERROR_NONE){
        ESP_LOGE(TAG, "UDP message creation fail (error %d : %s)", error, otThreadErrorToString(error));
    }
     else {
             uint16_t payloadLength = otMessageGetLength(message) - otMessageGetOffset(message);
             char buf1[payloadLength+1];
             otMessageRead(message, otMessageGetOffset(message),buf1, payloadLength);
             buf1[payloadLength]='\0';
             printf("UDP Message created: %s\n",buf1);
             // ESP_LOGI(TAG, "UDP message created.");
     }

    error = otUdpSend(esp_openthread_get_instance(), &udpSocket, message, &messageInfo);
    if (error != OT_ERROR_NONE){
        ESP_LOGE(TAG, "UDP send fail (error %d : %s)\n", error, otThreadErrorToString(error));
    }
    else {
     ESP_LOGI(TAG, "UDP sent.");
    }

    // esp_openthread_lock_release();
}
//static void udp_send_task()
//{
//   ESP_LOGI(TAG, "Transmitting sensor data...");
//     while(1)
//     {
//             udp_send();
//             vTaskDelay(5000 / portTICK_PERIOD_MS);
//     }
//}
static void ot_task_worker(void *aContext)
{
    esp_openthread_platform_config_t config = {
        .radio_config = ESP_OPENTHREAD_DEFAULT_RADIO_CONFIG(),
        .host_config = ESP_OPENTHREAD_DEFAULT_HOST_CONFIG(),
        .port_config = ESP_OPENTHREAD_DEFAULT_PORT_CONFIG(),
    };

    // Initialize the OpenThread stack
    ESP_LOGI(TAG, "initializing OpenThread Stack...");
    ESP_ERROR_CHECK(esp_openthread_init(&config));

#if CONFIG_OPENTHREAD_LOG_LEVEL_DYNAMIC
    // The OpenThread log level directly matches ESP log level
    (void)otLoggingSetLevel(CONFIG_LOG_DEFAULT_LEVEL);
```

```
#endif
    // Initialize the OpenThread cli
#if CONFIG_OPENTHREAD_CLI
    esp_openthread_cli_init();
#endif

    esp_netif_t *openthread_netif;
    // Initialize the esp_netif bindings
    openthread_netif = init_openthread_netif(&config);
    esp_netif_set_default_netif(openthread_netif);

#if CONFIG_OPENTHREAD_CLI_ESP_EXTENSION
    esp_cli_custom_command_init();
#endif // CONFIG_OPENTHREAD_CLI_ESP_EXTENSION

    // Run the main loop
    ESP_LOGI(TAG, "Starting OpenThread network...");
#if CONFIG_OPENTHREAD_CLI
    esp_openthread_cli_create_task();
#endif
#if CONFIG_OPENTHREAD_AUTO_START
    otOperationalDatasetTlvs dataset;
    otError error = otDatasetGetActiveTlvs(esp_openthread_get_instance(), &dataset);
    ESP_ERROR_CHECK(esp_openthread_auto_start((error == OT_ERROR_NONE) ? &dataset : NULL));
#endif
     udp_init();
    esp_openthread_launch_mainloop();

    // Clean up
    esp_netif_destroy(openthread_netif);
    esp_openthread_netif_glue_deinit();

    esp_vfs_eventfd_unregister();
    vTaskDelete(NULL);
    // xSemaphoreGive(semaphore);
    // vTaskSuspend(ot_handle);
    // vTaskDelay(1000/portTICK_PERIOD_MS);
}

//SD Card R/W Functions
static esp_err_t sd_card_write_file(const char *path, char *data)
{
    ESP_LOGI(TAG_SDCARD, "Opening file %s", path);
    FILE *f = fopen(path, "a+");
    if (f == NULL) {
        ESP_LOGE(TAG_SDCARD, "Failed to open file for writing");
        return ESP_FAIL;
    }
    fprintf(f, data);
    fclose(f);
    ESP_LOGI(TAG_SDCARD, "File written");

    return ESP_OK;
}
static esp_err_t sd_card_read_file(const char *path)
{
    ESP_LOGI(TAG_SDCARD, "Reading file %s", path);
    FILE *f = fopen(path, "r");
    if (f == NULL) {
        ESP_LOGE(TAG_SDCARD, "Failed to open file for reading");
        return ESP_FAIL;
    }
    char line[MAX_SIZE];
    fgets(line, sizeof(line), f);
    fclose(f);

    // strip newline
    char *pos = strchr(line, '\n');
    if (pos) {
        *pos = '\0';
    }
    ESP_LOGI(TAG_SDCARD, "Read from file: '%s'", line);

    return ESP_OK;
}
//SD Card Main Functions
void sd_card_write(void){
    esp_err_t ret;
    // Options for mounting the filesystem.
    // If format_if_mount_failed is set to true, SD card will be partitioned and
    // formatted in case when mounting fails.
    esp_vfs_fat_sdmmc_mount_config_t mount_config = {
#ifdef CONFIG_EXAMPLE_FORMAT_IF_MOUNT_FAILED
        .format_if_mount_failed = true,
#else
        .format_if_mount_failed = false,
#endif // EXAMPLE_FORMAT_IF_MOUNT_FAILED
        .max_files = 5,
        .allocation_unit_size = 16 * 1024
    };
    sdmmc_card_t *card;
    const char mount_point[] = MOUNT_POINT;
    ESP_LOGI(TAG_SDCARD, "Initializing SD card");

    // Use settings defined above to initialize SD card and mount FAT filesystem.
    ESP_LOGI(TAG_SDCARD, "Using SPI peripheral");

    // For setting a SD card frequency, use host.max_freq_khz (range 400kHz - 20MHz for SDSPI)
    sdmmc_host_t host = SDSPI_HOST_DEFAULT();

    spi_bus_config_t bus_cfg = {
        .mosi_io_num = PIN_NUM_MOSI,
        .miso_io_num = PIN_NUM_MISO,
        .sclk_io_num = PIN_NUM_CLK,
        .quadwp_io_num = -1,
        .quadhd_io_num = -1,
        .max_transfer_sz = 4000,
    };
    ret = spi_bus_initialize(host.slot, &bus_cfg, SPI_DMA_CH_AUTO);
    if (ret != ESP_OK) {
        ESP_LOGE(TAG_SDCARD, "Failed to initialize bus.");
        return;
    }

    sdspi_device_config_t slot_config = SDSPI_DEVICE_CONFIG_DEFAULT(); //SD Card Module has no CS and WP
     i/o
    slot_config.gpio_cs = PIN_NUM_CS;
    slot_config.host_id = host.slot;

    ESP_LOGI(TAG_SDCARD, "Mounting to SD Card...");
    ret = esp_vfs_fat_sdspi_mount(mount_point, &host, &slot_config, &mount_config, &card);
    if (ret != ESP_OK) {
        if (ret == ESP_FAIL) {
            ESP_LOGE(TAG_SDCARD, "Failed to mount");
        } else {
            ESP_LOGE(TAG_SDCARD, "Failed to initialize the card (%s).", esp_err_to_name(ret));
        }
        return;
    }
    ESP_LOGI(TAG_SDCARD, "SD Card mounted");

    // Card has been initialized, print its properties
    //sdmmc_card_print_info(stdout, card);

    // Directory for "aq_log.txt"
    const char *aq_log_file = MOUNT_POINT"/aq_log.csv";
    char log[MAX_SIZE];

    if (first_write == 1){
     //Write Headers
     snprintf(log, MAX_SIZE, "Time, CO (ppb), CO Index, CO2 (ppm), CO2 Index, NO2 (ppb), NO2 Index, PM2.5
     (ug/m3), PM2.5 Index, PM10 (ug/m3), PM10 Index, VOC (ppb), VOC Index, Temperature (C), RH (%s),
     Latitude, Longitude, Speed (km/h)\n", "%%");
     ret = sd_card_write_file(aq_log_file, log);
        if (ret != ESP_OK) {
            return;
        }
     first_write = 0;
    }
    // Set up data to be logged
    snprintf(log, MAX_SIZE, "%s, %.2f, %.0f, %.2f, %.0f, %.2f, %.0f, %.2f, %.0f, %.2f, %.0f, %.2f, %.0f,
     %.2f, %.2f, %.6f, %.6f, %.2f\n", Current_Date_Time, co, co_score, co2, co2_score, no2, no2_score,
     pm25, pm25_score, pm10, pm10_score, voc, voc_score, temp, rh, lat_deg, long_deg, speed_kmh);
    //printf("%s\n", log);
    // Write data log to aq_log.txt
    ret = sd_card_write_file(aq_log_file, log);
    if (ret != ESP_OK) {
        return;
    }
    //Open file for reading
/*    ret = sd_card_read_file(aq_log_file);
    if (ret != ESP_OK) {
        return;
    }*/

    // All done, unmount partition and disable SPI peripheral
    esp_vfs_fat_sdcard_unmount(mount_point, card);
    ESP_LOGI(TAG_SDCARD, "Card unmounted");

    //deinitialize the bus after all devices are removed
    spi_bus_free(host.slot);
}

//Functions for LED Alert System
void led_alert_init (void){
     gpio_reset_pin(RED_LED);
    gpio_set_direction(RED_LED, GPIO_MODE_OUTPUT);
    gpio_reset_pin(ORANGE_LED);
    gpio_set_direction(ORANGE_LED, GPIO_MODE_OUTPUT);
    gpio_reset_pin(YELLOW_LED);
    gpio_set_direction(YELLOW_LED, GPIO_MODE_OUTPUT);
    gpio_reset_pin(GREEN_LED);
    gpio_set_direction(GREEN_LED, GPIO_MODE_OUTPUT);
    gpio_set_level(RED_LED, 1);
    gpio_set_level(ORANGE_LED, 1);
    gpio_set_level(YELLOW_LED, 1);
    gpio_set_level(GREEN_LED, 1);
}
void update_led_alert (float pm25, float pm10, float rh, float temp, float co, float no2, float co2, float
    voc){
    //Condition for RED Level AQI
    if (pm25 > 30 || pm10 > 100 || co > 70 || co2 > 1500 || no2 > 250 || voc > 800){
            gpio_set_level(RED_LED, 1);
            gpio_set_level(ORANGE_LED, 0);
            gpio_set_level(YELLOW_LED, 0);
            gpio_set_level(GREEN_LED, 0);
//          led_strip_set_pixel(led_strip, 0, 255, 0, 0);
//          led_strip_refresh(led_strip);
    }
    //Condition for ORANGE Level AQI
    else if (pm25 > 20 || pm10 > 75 || co > 50 || co2 > 1150 || no2 > 175 || voc > 600){
            gpio_set_level(RED_LED, 0);
            gpio_set_level(ORANGE_LED, 1);
            gpio_set_level(YELLOW_LED, 0);
            gpio_set_level(GREEN_LED, 0);
//          led_strip_set_pixel(led_strip, 0, 255, 165, 0);
//          led_strip_refresh(led_strip);
    }
    //Condition for YELLOW Level AQI
    else if (pm25 > 15 || pm10 > 50 || co > 35 || co2 > 800 || no2 > 100 || voc > 400){
            gpio_set_level(RED_LED, 0);
            gpio_set_level(ORANGE_LED, 0);
            gpio_set_level(YELLOW_LED, 1);
            gpio_set_level(GREEN_LED, 0);
//          led_strip_set_pixel(led_strip, 0, 255, 255, 0);
//          led_strip_refresh(led_strip);
    }
    //Condition for GREEN Level AQI
    else if (pm25 > 0 || pm10 > 0 || co > 0 || co2 > 0 || no2 > 0 || voc > 0){
            gpio_set_level(RED_LED, 0);
            gpio_set_level(ORANGE_LED, 0);
            gpio_set_level(YELLOW_LED, 0);
            gpio_set_level(GREEN_LED, 1);
//          led_strip_set_pixel(led_strip, 0, 0, 255, 0);
//          led_strip_refresh(led_strip);
    }
    //negative values are encountered, possible code error
    else{
            gpio_set_level(RED_LED, 0.4);
            gpio_set_level(ORANGE_LED, 0.4);
            gpio_set_level(YELLOW_LED, 0.4);
            gpio_set_level(GREEN_LED, 0.4);
    }
}
void iaq_score(float pm25, float pm10, float co, float no2, float co2, float voc){
    float index[4][2] = {{0,50}, {51,75}, {76,100}, {101,150}};
    float co_bp[4][2] = {{0,35}, {36,50}, {51,70}, {70,1000}};
    float co2_bp[4][2] = {{0,800}, {801,1150}, {1151,1500}, {1501,60000}};
    float pm25_bp[4][2] = {{0,15}, {16,20}, {21,30}, {31,1000}};
    float pm10_bp[4][2] = {{0,50}, {51,75}, {76,100}, {101,1000}};
    float no2_bp[4][2] = {{0,100}, {101,175}, {176,250}, {251,10000}};
    float voc_bp[4][2] = {{0,400}, {401,600}, {601,800}, {801,60000}};
    //PM2.5
    for (int j=3; j>=0; j--){
            if(pm25 >= pm25_bp[j][0]){
                    pm25_score =
    (((index[j][1]-index[j][0])/(pm25_bp[j][1]-pm25_bp[j][0]))*(pm25-pm25_bp[j][0]))+(index[j][0]);
                    break;
            }
    }
    //PM10
    for (int j=3; j>=0; j--){
            if(pm10 >= pm10_bp[j][0]){
                    pm10_score =
    (((index[j][1]-index[j][0])/(pm10_bp[j][1]-pm10_bp[j][0]))*(pm10-pm10_bp[j][0]))+(index[j][0]);
                    break;
            }
    }
    //CO
    for (int j=3; j>=0; j--){
            if(co >= co_bp[j][0]){
                    co_score =
    (((index[j][1]-index[j][0])/(co_bp[j][1]-co_bp[j][0]))*(co-co_bp[j][0]))+(index[j][0]);
                    break;
            }
    }
    //NO2
    for (int j=3; j>=0; j--){
            if(no2 >= no2_bp[j][0]){
                    no2_score =
    (((index[j][1]-index[j][0])/(no2_bp[j][1]-no2_bp[j][0]))*(no2-no2_bp[j][0]))+(index[j][0]);
                    break;
            }
    }
    //CO2
    for (int j=3; j>=0; j--){
            if(co2 >= co2_bp[j][0]){
                    co2_score =
    (((index[j][1]-index[j][0])/(co2_bp[j][1]-co2_bp[j][0]))*(co2-co2_bp[j][0]))+(index[j][0]);
                    break;
            }
    }
    //VOC
    for (int j=3; j>=0; j--){
            if(voc >= voc_bp[j][0]){
                    voc_score =
    (((index[j][1]-index[j][0])/(voc_bp[j][1]-voc_bp[j][0]))*(voc-voc_bp[j][0]))+(index[j][0]);
                    break;
            }
    }

}
void driver_alert_init (void){
    gpio_reset_pin(RECIRC_LED);
    gpio_set_direction(RECIRC_LED, GPIO_MODE_OUTPUT);
    gpio_reset_pin(INTAKE_LED);
    gpio_set_direction(INTAKE_LED, GPIO_MODE_OUTPUT);
    gpio_set_level(INTAKE_LED, 0);
    gpio_set_level(RECIRC_LED, 0);
}
void driver_alert(float pm25_in, float pm10_in, float co_in, float no2_in, float co2_in, float voc_in,
                              float pm25_out, float pm10_out, float co_out, float no2_out, float
    co2_out, float voc_out){
    if (co_in > co_out || no2_in > no2_out || co2_in > co2_out || voc_in > voc_out){
            gpio_set_level(INTAKE_LED, 1); // Accumulated gas concentrations (Orange level)
            gpio_set_level(RECIRC_LED, 0);
    }
    else{
            gpio_set_level(INTAKE_LED, 0);
            gpio_set_level(RECIRC_LED, 1);
    }
}

//Warm-Up for MiCS-4514 for 3 minutes
void sensor_init(int *heated_var){
    gpio_reset_pin(PREHEAT_PIN);
    gpio_set_direction(PREHEAT_PIN, GPIO_MODE_OUTPUT);
    gpio_set_level(PREHEAT_PIN, 1);

    int i2c_master_port0 = I2C_PORT_0;
   i2c_config_t conf0 = {
       .mode = I2C_MODE_MASTER,
       .sda_io_num = I2C0_SDA,
       .scl_io_num = I2C0_SCL,
       .sda_pullup_en = GPIO_PULLUP_ENABLE,
       .scl_pullup_en = GPIO_PULLUP_ENABLE,
       .master.clk_speed = I2C_FREQUENCY,
    };
    i2c_param_config(i2c_master_port0, &conf0);
    i2c_driver_install(i2c_master_port0, conf0.mode, 0, 0, 0);
    uint8_t SGP30_INIT[2] = {0x20, 0x03}; //Initialize SGP30
    i2c_master_write_to_device(i2c_master_port0, SGP30_ADDR, SGP30_INIT, 2, 1000/portTICK_PERIOD_MS);
    if(heated_var == 0){
            printf("Preheating MiCS-4514...\n");
            vTaskDelay(180000/portTICK_PERIOD_MS); // Preheat MiCS for 3 minutes
    }
    else{
            printf("Waiting for SGP30...\n");
            vTaskDelay(1000/portTICK_PERIOD_MS);
    }
}
//For mapping MiCS-4514 raw readings
void process_mics4514(int D_co, int D_no2) {
    //Source https://myscope.net/auswertung-der-airpi-gas-sensoren/
    //For CO readings
    // printf("D_OUTS: %d, %d\n", D_co, D_no2);
    double Vo_co = 3.55 * (D_co-2113)/(4081-2113);
    double Vo_no2 = 3.55 * (D_no2-2113)/(4081-2113);
    // printf("VOLTAGE: %.5f  %.5f\n", Vo_co, Vo_no2);
    double Rs_co = (5-Vo_co) / (Vo_co/820);
    double Rs_no2 = (5-Vo_no2) / (Vo_no2/820);
    // printf("RESISTANCE: %.2f  %.2f\n", Rs_co, Rs_no2);
```

```
    double R0_co = 950; // FOR CALIBRATION (4.00 * R0 = Rs @ 0.8 ppm CO)
    co = pow(10, -1.1859 * (log(Rs_co/R0_co) / M_LN10) + 0.6201); //Curve Fitting Equation from Source
    //For NO2 readings
    double R0_no2 = 1440000; // FOR CALIBRATION (0.05*R0 = Rs @ 0.01 ppm NO2)
    no2 = pow(10, 0.9682 * (log(Rs_no2/R0_no2) / M_LN10) - 0.8108); //Curve Fitting Equation from Source
    //printf("CO: %.2f, NO2: %.2f\n", co, no2);
}
//RTOS Task for MiCS-4514 @ ADC1 Channel 1 & 2
void ADC1_Data_Task(void *params){
    //Initialize ADC1 Peripheral
    adc_oneshot_unit_handle_t adc1_handle;
    adc_oneshot_unit_init_cfg_t init_config1 = {
                .unit_id = ADC_UNIT_1,
                     .clk_src = ADC_DIGI_CLK_SRC_DEFAULT,
                     .ulp_mode = ADC_ULP_MODE_RISCV,
    };
    printf("ADC INIT\n");
    ESP_ERROR_CHECK(adc_oneshot_new_unit(&init_config1, &adc1_handle));

    //Configure the 2 ADC Channels
    adc_oneshot_chan_cfg_t config1 = {
                 .atten = ADC_ATTEN_DB_11,
                      .bitwidth = ADC_BITWIDTH_DEFAULT
    };
    ESP_ERROR_CHECK(adc_oneshot_config_channel(adc1_handle, ADC1_CH1_RED, &config1)); // For RED analog
    readings
    vTaskDelay(100/portTICK_PERIOD_MS);
    ESP_ERROR_CHECK(adc_oneshot_config_channel(adc1_handle, ADC1_CH2_NOX, &config1)); // For NOX analog
    readings
    vTaskDelay(100/portTICK_PERIOD_MS);
    while(1){
        ESP_ERROR_CHECK(adc_oneshot_read(adc1_handle, ADC1_CH1_RED, &co_adc)); // Get readings from RED
        vTaskDelay(10/portTICK_PERIOD_MS);
        ESP_ERROR_CHECK(adc_oneshot_read(adc1_handle, ADC1_CH2_NOX, &no2_adc)); // Get readings from NOX
        process_mics4514(co_adc, no2_adc);
             if(co != co){
                      co = 0;
             }
             if(no2 != no2){
                      no2 = 0;
             }
        //printf("%d", uxTaskGetStackHighWaterMark(NULL));
        // printf("MiCS-4514 Readings - CO: %.2f, NO2: %.2f\n", co, no2);
        xSemaphoreGive(semaphore);
        vTaskSuspend(mics_handle);
        vTaskDelay(1000/portTICK_PERIOD_MS);
    }
}
//Process SEN55 Data
void process_sen55(uint8_t data[24]){
    //Raw Bit Data
    //PM2.5 Bits
    uint16_t pm25_bits = data[3] << 8; //First 8 bits
                      pm25_bits |= data[4]; //Add last 8 bits
    //PM10 Bits
    uint16_t pm10_bits = data[9] << 8; //First 8 bits
                      pm10_bits |= data[10]; //Add last 8 bits
    //Humidity Bits
    int16_t rh_bits = data[12]; //First 8 bits
                     rh_bits <<= 8; //Shift by 8 bits
                     rh_bits |= data[13]; //Add last 8 bits
    //Temperature Bits
     int16_t temp_bits = data[15]; //First 8 bits
                      temp_bits <<= 8; //Shift by 8 bits
                      temp_bits |= data[16]; //Add last 8 bits


    //Divide by Scaling Factors
    float pm25_raw = pm25_bits;
    float pm10_raw = pm10_bits;
    float rh_raw = rh_bits;
    float temp_raw = temp_bits;
    pm25 = pm25_raw /10;
    pm10 = pm10_raw /10;
    rh = rh_raw / 100;
    temp = temp_raw / 200;
// CALIBRATION
    pm25 = 1.9467002394827233*pm25 + (2.5926092705655464);
    pm10 = 2.3350992860750863*pm10 + (1.1038457405169595);
    temp = 0.7311396775389937*temp + (5.1184212550840265);
    rh = 0.8405511007613504*rh + (-1.3350502127248518);
}
//SEN55 and SGP30 CRC Calculation
uint8_t CalcCrc(uint8_t data[2]) {
    uint8_t crc = 0xFF;
    for(int i = 0; i < 2; i++) {
             crc ^= data[i];
             for(uint8_t bit = 8; bit > 0; --bit) {
```

```
                        if(crc & 0x80) {
                                crc = (crc << 1) ^ 0x31u;
                        }
                        else {
                                crc = (crc << 1);
                        }
                }
        }
        return crc;
}
//RTOS Task for SEN55 @ I2C1
void I2C1_Data_Task(void *params){
    //SEN55 Initialize
    i2c_config_t conf1 = {
                        .mode = I2C_MODE_MASTER,
                .sda_io_num = I2C1_SDA,
                .scl_io_num = I2C1_SCL,
                .sda_pullup_en = GPIO_PULLUP_ENABLE,
                .scl_pullup_en = GPIO_PULLUP_ENABLE,
                .master.clk_speed = I2C_FREQUENCY,
    };
    i2c_param_config(I2C_PORT_1, &conf1);
    i2c_driver_install(I2C_PORT_1, conf1.mode, 0, 0, 0);
    uint8_t SEN55_MEAS_INIT[2] = {0x00,0x21}; //Start Measurement Mode
    uint8_t sen55_data[24];
    uint8_t SEN55_READ_DATA[2] = {0x03,0xC4}; //Get Measured Values Command
    ESP_ERROR_CHECK(i2c_master_write_to_device(I2C_PORT_1, SEN55_ADDR, SEN55_MEAS_INIT, 2,
    1000/portTICK_PERIOD_MS)); //Measurement Mode
    vTaskDelay(10/portTICK_PERIOD_MS);
    while (1) {
    //For SEN55 (Starts at Idle Mode)
    ESP_ERROR_CHECK(i2c_master_write_to_device(I2C_PORT_1, SEN55_ADDR, SEN55_READ_DATA, 2,
    1000/portTICK_PERIOD_MS)); //Send Get-Measurement Command
    vTaskDelay(20/portTICK_PERIOD_MS);
    ESP_ERROR_CHECK(i2c_master_read_from_device(I2C_PORT_1, SEN55_ADDR, sen55_data, 24,
    1000/portTICK_PERIOD_MS)); //Read Measurements
    vTaskDelay(10/portTICK_PERIOD_MS);
    process_sen55(sen55_data); //Update Global Variables
    //printf("%d", uxTaskGetStackHighWaterMark(NULL));
    // printf("SEN55 Readings : PM2.5: %.2f, PM10: %.2f, Temperature: %.02f, RH: %.2f\n", pm25, pm10,
    temp, rh);
        xSemaphoreGive(semaphore);
        vTaskSuspend(sen_handle);
        vTaskDelay(1000/portTICK_PERIOD_MS);
    }
}
//RTOS Task for SGP30 @ I2C0
void I2C0_Data_Task(void *params){
    int i2c_master_port = I2C_PORT_0;
    uint8_t sgp30_data[6];
    uint8_t SGP30_MEAS[2] = {0x20, 0x08};
    while (true) {
            ESP_ERROR_CHECK(i2c_master_write_to_device(i2c_master_port, SGP30_ADDR, SGP30_MEAS, 2,
    1000/portTICK_PERIOD_MS)); //Send Measurement Command
    vTaskDelay(15/portTICK_PERIOD_MS);
            i2c_master_read_from_device(i2c_master_port, SGP30_ADDR, sgp30_data, 6,
    1000/portTICK_PERIOD_MS); //Read Measured Values
            //Process received bit sequence
            uint16_t co2_bits = sgp30_data[0] << 8;
            co2_bits |= sgp30_data[1];
            uint16_t voc_bits = sgp30_data[3] << 8;
            voc_bits |= sgp30_data[4];
            float co2_raw = co2_bits;
            float voc_raw = voc_bits;
            co2 = co2_raw;
            voc = voc_raw;
            // printf("SGP30 Readings - CO2: %.2f, VOC: %.2f\n", co2, voc);
    xSemaphoreGive(semaphore);
    vTaskSuspend(sgp_handle); // So only 1 semaphore is given in case a task is faster than the other
    vTaskDelay(1000/portTICK_PERIOD_MS);
    }
}
//RTOS Task for All_data_collected
void Data_Complete_Task (void *params){
    while(1){
            if(uxSemaphoreGetCount(semaphore) == TASK_COUNT){
                    for (int i=0; i == TASK_COUNT; i++){
                            xSemaphoreTake(semaphore, portMAX_DELAY);
                    }
                    //printf("Data Upload/Network and Alerts Task\n\n");
                    //printf("SGP30 Task State: %d\n", eTaskGetState(sgp_handle));
                    //printf("MiCS-4514 Task State: %d\n", eTaskGetState(mics_handle));
                    //printf("SEN55 Task State: %d\n\n", eTaskGetState(sen_handle));


                    // ReadAttribute(); // For Zigbee
                    // vTaskDelay(1000/portTICK_PERIOD_MS); // For Zigbee
                    snprintf(Current_Date_Time, sizeof(Current_Date_Time), "%s", Rx_Date_Time);
```

```
                        // printf("%s\n", Current_Date_Time);
                        // Current_Date_Time = Rx_Date_Time;
                        update_led_alert(pm25, pm10, temp, rh, co, no2, co2, voc);
                        iaq_score(pm25, pm10, co, no2, co2, voc);

                //Thread
                        snprintf(data_payload, 500,
        "{\"COraw\":\"%.2f\",\"COindex\":\"%.0f\",\"CO2raw\":\"%.2f\",\"CO2index\":\"%.0f\",\"NO2raw\":\"%.2f\
        ",\"NO2index\":\"%.0f\",\"PM25raw\":\"%.2f\",\"PM25index\":\"%.0f\",\"PM10raw\":\"%.2f\",\"PM10index\"
        :\"%.0f\",\"VOCraw\":\"%.2f\",\"VOCindex\":\"%.0f\",\"T\":\"%.2f\",\"RH\":\"%.2f\",\"source\":\"Node
        1\",\"local_time\":\"%s\",\"type\":\"data\"}",co,co_score,co2,co2_score,no2,no2_score,pm25,pm25_score,
        pm10,pm10_score,voc,voc_score,temp,rh,Rx_Date_Time);
                        // snprintf(data_payload, 150,
        "1,%.0f,%.2f,%.0f,%.2f,%.2f,%.2f,%.0f,%.2f,%.0f,%.2f,%.0f,%.2f,%.0f,%.2f,%s", pm25_score, pm25,
        pm10_score, pm10, temp, rh, co2_score, co2,voc_score, voc, co_score, co, no2_score, no2,
        Current_Date_Time);
                        // printf("%s\n", data_payload);
                        // printf("Data String: %s\n", data_payload);
                        udp_send();

                        sd_card_write();
                        vTaskDelay(30000/portTICK_PERIOD_MS);
                        vTaskResume(sgp_handle);
                        vTaskResume(mics_handle);
                        vTaskResume(sen_handle);
                        //vTaskResume(uart_handle);
                }
                else{
                        vTaskDelay(1000/portTICK_PERIOD_MS);
                }
        }
}
 //GPS (UART), Driver Alert
void UART_Data_Task(void *params){
    //Start UART Here
    const uart_port_t uart_num = UART_NUM_1;
    uart_config_t uart_config = {
        .baud_rate = 9600, //GPS baud rate according to datasheet
        .data_bits = UART_DATA_8_BITS,
        .parity = UART_PARITY_DISABLE,
        .stop_bits = UART_STOP_BITS_1,
        .flow_ctrl = UART_HW_FLOWCTRL_DISABLE,
    };
    // Configure UART parameters
    ESP_ERROR_CHECK(uart_param_config(uart_num, &uart_config));
    // Set UART pins(TX: IO4, RX: IO5, RTS: IO18, CTS: IO19)
    ESP_ERROR_CHECK(uart_set_pin(uart_num, TX_pin, RX_pin, UART_PIN_NO_CHANGE, UART_PIN_NO_CHANGE)); //RTS
    and CTS pins will not be used
    // Setup UART buffered IO with event queue
    const int uart_buffer_size = (1024*3);
    QueueHandle_t uart_queue;
    // Install UART driver using an event queue here
    ESP_ERROR_CHECK(uart_driver_install(uart_num, uart_buffer_size, uart_buffer_size, 10, &uart_queue,
    0));
    vTaskDelay(100/portTICK_PERIOD_MS);
    const char* filter_str_loc = "$GPGGA";
    const char* filter_str_speed = "$GPVTG";
    char *loc = NULL;
    char *speed = NULL;
    char str1[100];
    char str2[45];
    int j=0;
    while(1){
            int length;

            ESP_ERROR_CHECK(uart_get_buffered_data_len(uart_num, (size_t*)&length));
            char gps_out[length];

            uart_read_bytes(uart_num, gps_out, length, 1000);

            loc = strstr(gps_out,filter_str_loc);
        if (loc == NULL) {
            printf("GPGGA not found in the sentence\n");
        }
        else{
                char *newline = strchr(loc, '\n');
                if (newline == NULL) {
                    printf("Incomplete NMEA Sentence\n");
                }
                else if(loc){
                            for (int i=0; loc[i] != '\n'; i++){
                                    str1[j] = loc[i];
                                    j++;
                            }
                            str1[j+1] = '\n';
                            j = 0;
                            printf("%s\n\n", str1);
                            char* token = strtok(str1, ",");
```

```
                                token = strtok(NULL, ",");
                                for (int i=0; i<4; i++){
                                        token = strtok(NULL,",");
                                        location[i] = token;
                                }
                                double latitude = atof(location[0]);
                                double longitude = atof(location[2]);

                                 latitude_decimal = floor(latitude/100);
                                 longitude_decimal = floor(longitude/100);

                                if(latitude_decimal == 0 && longitude_decimal == 0){
                                        printf("No Signal\n");
                                }
                                else{
                                        double latitude_decimal_minutes = modf(latitude/100, &latitude);
                                        latitude_decimal_minutes = latitude_decimal_minutes*100;
                                        double longitude_decimal_minutes = modf(longitude/100,
    &longitude);
                                        longitude_decimal_minutes = longitude_decimal_minutes*100;

                                        lat_deg = latitude_decimal + latitude_decimal_minutes/60;
                                        long_deg = longitude_decimal + longitude_decimal_minutes/60;
                                        //printf("%.8f, %.8f\n", lat_deg, long_deg);
                                }
                        }
        }
        //printf("%s\n\n", gps_out);
        speed_str = "0";
             speed = strstr(gps_out,filter_str_speed);
             //printf("%s", speed);
        if (speed == NULL) {
            printf("GPVTG not found in the sentence\n");
        }
        else{
                char *newline2 = strchr(speed, '\n');
                if (newline2 == NULL) {
                    printf("Incomplete NMEA Sentence\n");
                }
                else if (newline2 != NULL && latitude_decimal != 0 && longitude_decimal != 0){
                     for (int i=0; speed[i] != '\n'; i++){
                                str2[j] = speed[i];
                                j++;
                     }
                     str2[j+1] = '\n';
                     j = 0;
                     printf("%s\n\n", str2);
                     if(strlen(str2) > 20){
                                        char* token2 = strtok(str2, ",");
                                        token2 = strtok(NULL, ",");
                                        for (int i=0; i<6; i++){
                                                token2 = strtok(NULL,",");
                                                velocity[i] = token2;
                                                if (strcmp(token2, "N") == 0){
                                                        token2 = strtok(NULL,",");
                                                        speed_str = token2;
                                                        break;
                                                }
                                        }
                     }
                                printf("SPEED STRING: %s\n", speed_str);
                                printf("%.7f, %.7f\n", lat_deg, long_deg);
                                if(speed_str == NULL){
                                        strcpy(speed_str, "0");
                                }
                                speed_kmh = atof(speed_str);
                                printf("%.2f\n\n", speed_kmh);
                                if (speed_kmh < 1){
                                        speed_kmh = 1;
                                }

                }
        }

        //xSemaphoreGive(semaphore);
             //vTaskSuspend(uart_handle);
        //sd_card_write();
        char gps_data_format[30] = "00.0000000,000.0000000,00.00,";
        snprintf(gps_data_format, 30, "%.7f,%.7f,%.2f,", lat_deg, long_deg, speed_kmh);
//      esp_zb_zcl_attr_t *value_r = esp_zb_zcl_get_attribute(ZB_CLIENT_ENDPOINT_4,
    SENSOR_DATA_CLUSTER_ID, ESP_ZB_ZCL_CLUSTER_SERVER_ROLE, SENSOR_DATA_ATTR_ID_3);
//      memcpy(value_r->data_p, &gps_data_format, sizeof(gps_data_format));
        vTaskDelay(1000/portTICK_PERIOD_MS);
    }
}

void app_main(void)
{
    // Used eventfds:
    // * netif
    // * ot task queue
    // * radio driver
    esp_vfs_eventfd_config_t eventfd_config = {
        .max_fds = 3,
    };

    ESP_ERROR_CHECK(nvs_flash_init());
    ESP_ERROR_CHECK(esp_event_loop_create_default());
    ESP_ERROR_CHECK(esp_netif_init());
    ESP_ERROR_CHECK(esp_vfs_eventfd_register(&eventfd_config));
    semaphore = xSemaphoreCreateCounting(TASK_COUNT, 0);
     heated = 1;

     //configure_led();
     led_alert_init();

     // xTaskCreate(ot_task_worker, "ot_cli_main", 10240, xTaskGetCurrentTaskHandle(), 5, NULL);
    xTaskCreate(ot_task_worker, "ot_cli_main", 10240, NULL, 5, &ot_handle);
    vTaskDelay(35000/portTICK_PERIOD_MS);
     sensor_init(&heated);
    xTaskCreate(ADC1_Data_Task, "ADC1 Data Task", 1024*3, NULL, 5, &mics_handle);
    xTaskCreate(I2C0_Data_Task, "I2C0 Data Task", 1024*3, NULL, 5, &sgp_handle);
    xTaskCreate(I2C1_Data_Task, "I2C1 Data Task", 1024*2, NULL, 5, &sen_handle);
    // xTaskCreate(UART_Data_Task, "UART Data Task", 1024*8, NULL, 5, &uart_handle);
    xTaskCreate(Data_Complete_Task, "Data Upload and Save", 4096, NULL, 5, &finish_handle);
}
```

Appendix I

Code for Thread System - Border Router

Template Source Code:
https://github.com/espressif/esp-idf/tree/master/examples/openthread/ot_br

```
#include <stdio.h>
#include <string.h>

#include "esp_check.h"
#include "esp_err.h"
#include "esp_event.h"
#include "esp_log.h"
#include "esp_netif.h"
#include "esp_openthread.h"
#include "esp_openthread_border_router.h"
#include "esp_openthread_cli.h"
#include "esp_openthread_lock.h"
#include "esp_openthread_netif_glue.h"
#include "esp_openthread_types.h"
#include "esp_ot_cli_extension.h"
#include "esp_ot_config.h"
#include "esp_ot_wifi_cmd.h"
#include "esp_vfs_dev.h"
#include "esp_vfs_eventfd.h"
#include "esp_wifi.h"
#include "mdns.h"
#include "nvs_flash.h"
#include "protocol_examples_common.h"
#include "sdkconfig.h"
#include "driver/uart.h"
#include "freertos/FreeRTOS.h"
#include "freertos/task.h"
#include "hal/uart_types.h"
#include "openthread/error.h"
#include "openthread/logging.h"
#include "openthread/tasklet.h"

#include "openthread/thread.h"
#include "openthread/udp.h"
#include "esp_sntp.h"
#include "mqtt_client.h"

#define TAG "OT BR"

#define UDP_PORT 2222
#define UDP_PORT_nodes 1234

otUdpSocket udpSocket;

char Current_Date_Time[20];
esp_mqtt_client_handle_t MQTtest;
// SemaphoreHandle_t semaphore;

// char json_payload[500] = "";
// char send_payload[500] = "";

char payload_1[500] = "";
char payload_2[500] = "";
char payload_3[500] = "";
char payload_4[500] = "";

#if CONFIG_BROKER_CERTIFICATE_OVERRIDDEN == 1
static const uint8_t mqtt_eclipseprojects_io_pem_start[]  = "-----BEGIN CERTIFICATE-----\n"
    CONFIG_BROKER_CERTIFICATE_OVERRIDE "\n-----END CERTIFICATE-----";
#else
extern const uint8_t mqtt_eclipseprojects_io_pem_start[]
    asm("_binary_mqtt_eclipseprojects_io_pem_start");
#endif
extern const uint8_t mqtt_eclipseprojects_io_pem_end[]   asm("_binary_mqtt_eclipseprojects_io_pem_end");

static const char *TAG_MQTT = "MQTTS_EXAMPLE";

#if CONFIG_EXTERNAL_COEX_ENABLE
static void ot_br_external_coexist_init(void)
{
    esp_external_coex_gpio_set_t gpio_pin = ESP_OPENTHREAD_DEFAULT_EXTERNAL_COEX_CONFIG();
    esp_external_coex_set_work_mode(EXTERNAL_COEX_LEADER_ROLE);
    ESP_ERROR_CHECK(esp_enable_extern_coex_gpio_pin(CONFIG_EXTERNAL_COEX_WIRE_TYPE, gpio_pin));
}
#endif /* CONFIG_EXTERNAL_COEX_ENABLE */

void time_sync_notification_cb(struct timeval *tv)
```

```
{
    ESP_LOGI(TAG, "Notification of a time synchronization event");
}

//RETURNS the final time in string form
void Get_current_date_time(char *date_time){
     char strftime_buf[100];
     time_t now;
         struct tm timeinfo;
         time(&now);
         localtime_r(&now, &timeinfo);

              // Set timezone to Indian Standard Time
                                setenv("TZ", "UTC-08:00", 1);
                  tzset();
                  localtime_r(&now, &timeinfo);

                  strftime(strftime_buf, sizeof(strftime_buf), "%c", &timeinfo);
                  //strftime(date_time, 100, "%Y-%m-%dT%H:%M:%S", &timeinfo);
                  //ESP_LOGI(TAG, "The current date/time in Philippines is: %s", strftime_buf);

                  // Extract the time values manually
                  int year = timeinfo.tm_year + 1900;  // Years since 1900, so add 1900
                  int month = timeinfo.tm_mon + 1;     // Months are 0-based, so add 1
                  int day = timeinfo.tm_mday;
                  int hour = timeinfo.tm_hour;
                  int minute = timeinfo.tm_min;
                  int second = timeinfo.tm_sec;

                  //concatenate the string manually
                  sprintf(strftime_buf, "%d-%02d-%02dT%02d:%02d:%02d", year, month, day, hour, minute,
     second);
                strcpy(date_time,strftime_buf);
}

//initializes SNTP server settings
static void initialize_sntp(void)
{
    ESP_LOGI(TAG, "Initializing SNTP");
    esp_sntp_setoperatingmode(SNTP_OPMODE_POLL);
    esp_sntp_setservername(0, "time.google.com");
    sntp_set_time_sync_notification_cb(time_sync_notification_cb);
#ifdef CONFIG_SNTP_TIME_SYNC_METHOD_SMOOTH
    sntp_set_sync_mode(SNTP_SYNC_MODE_SMOOTH);
#endif
    esp_sntp_init();
}

// helper function that obtains timezone
static void obtain_time(void)
{
    initialize_sntp();
    // wait for time to be set
    time_t now = 0;
    struct tm timeinfo = { 0 };
    int retry = 0;
    const int retry_count = 10;
    while (sntp_get_sync_status() == SNTP_SYNC_STATUS_RESET && ++retry < retry_count) {
        ESP_LOGI(TAG, "Waiting for system time to be set... (%d/%d)", retry, retry_count);
        vTaskDelay(2000 / portTICK_PERIOD_MS);
    }
    time(&now);
    localtime_r(&now, &timeinfo);
}

//function that actually syncs with NTP server
 void Set_SystemTime_SNTP()  {

      time_t now;
         struct tm timeinfo;
         time(&now);
         localtime_r(&now, &timeinfo);
         // Is time set? If not, tm_year will be (1970 - 1900).
         if (timeinfo.tm_year < (2016 - 1900)) {
             ESP_LOGI(TAG, "Time is not set yet. Connecting to WiFi and getting time over NTP.");
             obtain_time();
             // update 'now' variable with current time
             time(&now);
         }
}

void mqtt_publish(esp_mqtt_client_handle_t client, char *mqtt_msg){
     const char *Topic_Name = "UPCARE/UNDERGRAD/ECE199_PUB2324";
     esp_mqtt_client_publish(client, Topic_Name, mqtt_msg, 0, 2, 0);
}

static void mqtt_event_handler(void *handler_args, esp_event_base_t base, int32_t event_id, void
     *event_data)
```

```
{
    ESP_LOGD(TAG_MQTT, "Event dispatched from event loop base=%s, event_id=%" PRIi32, base, event_id);
    esp_mqtt_event_handle_t event = event_data;
    esp_mqtt_client_handle_t client = event->client;
    int msg_id;
    switch ((esp_mqtt_event_id_t)event_id) {
    case MQTT_EVENT_CONNECTED:
        ESP_LOGI(TAG_MQTT, "MQTT_EVENT_CONNECTED");
        msg_id = esp_mqtt_client_subscribe(client, "/topic/qos0", 0);
        ESP_LOGI(TAG_MQTT, "sent subscribe successful, msg_id=%d", msg_id);

        break;
    case MQTT_EVENT_DISCONNECTED:
        ESP_LOGI(TAG_MQTT, "MQTT_EVENT_DISCONNECTED");
        break;
    case MQTT_EVENT_SUBSCRIBED:
     ESP_LOGI(TAG_MQTT, "MQTT_EVENT_SUBSCRIBED, msg_id=%d", event->msg_id);
        break;
    case MQTT_EVENT_UNSUBSCRIBED:
        ESP_LOGI(TAG_MQTT, "MQTT_EVENT_UNSUBSCRIBED, msg_id=%d", event->msg_id);
        break;
    case MQTT_EVENT_PUBLISHED:
        ESP_LOGI(TAG_MQTT, "MQTT_EVENT_PUBLISHED, msg_id=%d", event->msg_id);
        break;
    case MQTT_EVENT_DATA:
        ESP_LOGI(TAG_MQTT, "MQTT_EVENT_DATA");
        break;
    case MQTT_EVENT_ERROR:
        ESP_LOGI(TAG_MQTT, "MQTT_EVENT_ERROR");
        if (event->error_handle->error_type == MQTT_ERROR_TYPE_TCP_TRANSPORT) {
            ESP_LOGI(TAG_MQTT, "Last error code reported from esp-tls: 0x%x",
    event->error_handle->esp_tls_last_esp_err);
            ESP_LOGI(TAG_MQTT, "Last tls stack error number: 0x%x",
    event->error_handle->esp_tls_stack_err);
            ESP_LOGI(TAG_MQTT, "Last captured errno : %d (%s)",
    event->error_handle->esp_transport_sock_errno,
                    strerror(event->error_handle->esp_transport_sock_errno));
        } else if (event->error_handle->error_type == MQTT_ERROR_TYPE_CONNECTION_REFUSED) {
            ESP_LOGI(TAG_MQTT, "Connection refused error: 0x%x",
    event->error_handle->connect_return_code);
        } else {
            ESP_LOGW(TAG_MQTT, "Unknown error type: 0x%x", event->error_handle->error_type);
        }
        break;
    default:
        ESP_LOGI(TAG_MQTT, "Other event id:%d", event->event_id);
        break;
    }
}

static void mqtt_app_start(esp_mqtt_client_handle_t client)
{
    esp_mqtt_client_register_event(client, ESP_EVENT_ANY_ID, mqtt_event_handler, NULL);
    esp_mqtt_client_start(client);
}

void upload_data (void *params)
{
    while(1){
        if (strcmp(payload_1, "") != 0) {
            // printf("payload 1\n");
            mqtt_publish(MQTtest, payload_1);
            strcpy(payload_1, "");
        }
        else if (strcmp(payload_2, "") != 0) {
            // printf("payload 2\n");
            mqtt_publish(MQTtest, payload_2);
            strcpy(payload_2, "");
        }
        else if (strcmp(payload_3, "") != 0) {
            // printf("payload 3\n");
            mqtt_publish(MQTtest, payload_3);
            strcpy(payload_3, "");
        }
        else if (strcmp(payload_4, "") != 0) {
            // printf("payload 4\n");
            mqtt_publish(MQTtest, payload_4);
            strcpy(payload_4, "");
        }
        vTaskDelay(1000/portTICK_PERIOD_MS);
    }
}

// source: https://www.esp32.com/viewtopic.php?t=38516
static void udp_send(void) // ff02:1, 64000, ff03:1, 2222
{
     otMessageInfo messageInfo;
    otMessageSettings msgSettings;
    msgSettings.mLinkSecurityEnabled = true;
    msgSettings.mPriority = 1;
    otIp6AddressFromString("ff03::1", &messageInfo.mPeerAddr);
    messageInfo.mPeerPort = UDP_PORT_nodes;
    otMessage * message = otUdpNewMessage(esp_openthread_get_instance(), &msgSettings);
    // const char * buf =
     "id1,18,5.90,6,5.90,32.03,49.86,0,400.00,0,0.00,2,1.77,0,0.07,14.699734,121.723847,23.55";
    const char * buf = Current_Date_Time;
    // char * buf = "";
    // snprintf(buf, sizeof(data_payload), "%s", data_payload);

    otError error = otMessageAppend(message, buf, (uint16_t) strlen(buf));
    if (error != OT_ERROR_NONE){
        ESP_LOGE(TAG, "UDP message creation fail (error %d : %s)", error, otThreadErrorToString(error));
    }
     // else {
     //       uint16_t payloadLength = otMessageGetLength(message) - otMessageGetOffset(message);
     //       char buf1[payloadLength+1];
     //       otMessageRead(message, otMessageGetOffset(message),buf1, payloadLength);
     //       buf1[payloadLength]='\0';
     //       printf("UDP Message created: %s\n",buf1);
     //       ESP_LOGI(TAG, "UDP message created.");
     // }

    error = otUdpSend(esp_openthread_get_instance(), &udpSocket, message, &messageInfo);
    if (error != OT_ERROR_NONE){
        ESP_LOGE(TAG, "UDP send fail (error %d : %s)", error, otThreadErrorToString(error));
    }
    // else {
    //       ESP_LOGI(TAG, "UDP sent. Message: %s\n", buf);
    // }
}
static void udp_send_task()
{
    while(1){
        Get_current_date_time(Current_Date_Time);
        udp_send();
        vTaskDelay(1000/portTICK_PERIOD_MS);
    }
}

void udp_rx_cb(void *aContext, otMessage *aMessage, const otMessageInfo *aMessageInfo) // receive callback
     for sensor nodes
{
    uint16_t payloadLength = otMessageGetLength(aMessage) - otMessageGetOffset(aMessage);
     char buf[payloadLength+1];
     otMessageRead(aMessage, otMessageGetOffset(aMessage), buf, payloadLength);
     buf[payloadLength]='\0';
     if (strcmp(payload_1, "") == 0){
        snprintf(payload_1, 500, "%s", buf);
        // printf("payload_1\n");
    } else if ( strcmp(payload_2, "") == 0){
        snprintf(payload_2, 500, "%s", buf);
        // printf("payload_2\n");
    } else if ( strcmp(payload_3, "") == 0){
        snprintf(payload_3, 500, "%s", buf);
        // printf("payload_3\n");
    } else{
        snprintf(payload_4, 500, "%s", buf);
        // printf("payload_4\n");
    }
    ESP_LOGI(TAG, "UDP received successfully");
    // snprintf(json_payload, 500, "%s", buf);

    // snprintf(json_payload, 500, "%s", buf);
    // printf("%s\n", json_payload);
    // mqtt_publish(MQTtest, buf); // BR Hangs eventually
}

static void udp_init(void)
{
    otInstance * thread_instance = esp_openthread_get_instance();
    otSockAddr bind_info;
//    otUdpSocket udpSocket; // Declare this globally
    otNetifIdentifier netif = OT_NETIF_THREAD;
    memset(&bind_info, 0, sizeof(otSockAddr));
    otIp6AddressFromString("::", &bind_info.mAddress);
    bind_info.mPort = UDP_PORT;

    otError error = otUdpOpen(thread_instance, &udpSocket, udp_rx_cb, NULL);
    if (error != OT_ERROR_NONE)
    {
        ESP_LOGE(TAG, "UDP open error (error %d:%s)", error, otThreadErrorToString(error));
    } else
    {
        ESP_LOGI(TAG, "UDP initialized");
    }
    error = otUdpBind(thread_instance, &udpSocket, &bind_info, netif);
    if (error != OT_ERROR_NONE)
    {
        ESP_LOGE(TAG, "UDP bind error (error %d:%s)", error, otThreadErrorToString(error));
    }else{
        ESP_LOGI(TAG, "UDP binded");
    }
}

static void ot_task_worker(void *aContext)
{
    esp_openthread_platform_config_t config = {
        .radio_config = ESP_OPENTHREAD_DEFAULT_RADIO_CONFIG(),
        .host_config = ESP_OPENTHREAD_DEFAULT_HOST_CONFIG(),
        .port_config = ESP_OPENTHREAD_DEFAULT_PORT_CONFIG(),
    };

    esp_netif_config_t cfg = ESP_NETIF_DEFAULT_OPENTHREAD();
    esp_netif_t       *openthread_netif = esp_netif_new(&cfg);
    assert(openthread_netif != NULL);

    // Initialize the OpenThread stack
    ESP_ERROR_CHECK(esp_openthread_init(&config));

    // Initialize border routing features
    esp_openthread_lock_acquire(portMAX_DELAY);
    ESP_ERROR_CHECK(esp_netif_attach(openthread_netif, esp_openthread_netif_glue_init(&config)));

    (void)otLoggingSetLevel(CONFIG_LOG_DEFAULT_LEVEL);
    esp_openthread_cli_init();

//    esp_cli_custom_command_init();

//    esp_openthread_cli_create_task();

#if CONFIG_OPENTHREAD_BR_AUTO_START
    ESP_ERROR_CHECK(esp_openthread_border_router_init());
    otOperationalDatasetTlvs dataset;
    otError error = otDatasetGetActiveTlvs(esp_openthread_get_instance(), &dataset);
    ESP_ERROR_CHECK(esp_openthread_auto_start((error == OT_ERROR_NONE) ? &dataset : NULL));
#endif // CONFIG_OPENTHREAD_BR_AUTO_START

    esp_cli_custom_command_init();
//    esp_openthread_lock_release();

    // Run the main loop
    esp_openthread_cli_create_task();
    udp_init();
    esp_openthread_lock_release();
    esp_openthread_launch_mainloop();

    // Clean up
    esp_netif_destroy(openthread_netif);
    esp_openthread_netif_glue_deinit();
    esp_vfs_eventfd_unregister();
    vTaskDelete(NULL);
}

static void wifi_event_handler(void *event_handler_arg, esp_event_base_t event_base, int32_t event_id,
    void *event_data)
{
    switch (event_id)
    {
    case WIFI_EVENT_STA_START:
        printf("WiFi connecting ... \n");
        break;
    case WIFI_EVENT_STA_CONNECTED:
        printf("WiFi connected ... \n");
        break;
    case WIFI_EVENT_STA_DISCONNECTED:
        printf("WiFi lost connection ... \n");
        esp_wifi_connect();
        break;
    case IP_EVENT_STA_GOT_IP:
        printf("WiFi got IP...\n\n");
        break;
    default:
        break;
    }
}

void wifi_connection()
{
    // 1 - Wi-Fi/LwIP Init Phase
    esp_netif_init();                    // TCP/IP initiation
    esp_netif_create_default_wifi_sta(); // WiFi station
    wifi_init_config_t wifi_initiation = WIFI_INIT_CONFIG_DEFAULT();
    esp_wifi_init(&wifi_initiation); // s1.4
    esp_event_handler_register(WIFI_EVENT, ESP_EVENT_ANY_ID, wifi_event_handler, NULL);
    esp_event_handler_register(IP_EVENT, IP_EVENT_STA_GOT_IP, wifi_event_handler, NULL);
    wifi_config_t wifi_configuration = {
        .sta = {
                      .ssid = "*INSERT SSID",
                      .password = "*INSERT PASSWORD"}};
    esp_wifi_set_mode(WIFI_MODE_STA);


    esp_wifi_set_config(WIFI_IF_STA, &wifi_configuration);
    // 3 - Wi-Fi Start Phase
    printf("WiFi Starting...\n");
    esp_wifi_start();
    // 4- Wi-Fi Connect Phase
    esp_wifi_connect();
}

void app_main(void)
{
    // Used eventfds:
    // * netif
    // * task queue
    // * border router
    esp_vfs_eventfd_config_t eventfd_config = {
#if CONFIG_OPENTHREAD_RADIO_NATIVE
        // * radio driver (A native radio device needs a eventfd for radio driver.)
        .max_fds = 4,
#else
        .max_fds = 3,
#endif
    };
    ESP_ERROR_CHECK(esp_vfs_eventfd_register(&eventfd_config));

    ESP_ERROR_CHECK(nvs_flash_init());
    ESP_ERROR_CHECK(esp_netif_init());
    ESP_ERROR_CHECK(esp_event_loop_create_default());

#if CONFIG_EXAMPLE_CONNECT_WIFI
#if CONFIG_OPENTHREAD_BR_AUTO_START
    ESP_ERROR_CHECK(example_connect());
    // wifi_connection();
    vTaskDelay(8000 / portTICK_PERIOD_MS);
    Set_SystemTime_SNTP();
    const esp_mqtt_client_config_t mqtt_cfg = {
     //broker URL or URI
        .broker = {
         .address.uri = "*INSERT URI",
            .verification.certificate = (const char *)mqtt_eclipseprojects_io_pem_start
        },
        //this is where I put the credentials
     //I added this
     .credentials = {
            .username="*INSERT USERNAME",
             .authentication = {
               .password="*INSERT PASSWORD"
            }
        },

    };
    MQTtest = esp_mqtt_client_init(&mqtt_cfg);
    mqtt_app_start(MQTtest);
#if CONFIG_ESP_COEX_SW_COEXIST_ENABLE && CONFIG_OPENTHREAD_RADIO_NATIVE
    ESP_ERROR_CHECK(esp_wifi_set_ps(WIFI_PS_MIN_MODEM));
    ESP_ERROR_CHECK(esp_coex_wifi_i154_enable());
#else
    ESP_ERROR_CHECK(esp_wifi_set_ps(WIFI_PS_NONE));

#if CONFIG_EXTERNAL_COEX_ENABLE
    ot_br_external_coexist_init();
#endif // CONFIG_EXTERNAL_COEX_ENABLE

#endif
    esp_openthread_set_backbone_netif(get_example_netif());
#else
    esp_ot_wifi_netif_init();
    esp_openthread_set_backbone_netif(esp_netif_get_handle_from_ifkey("WIFI_STA_DEF"));
#endif // CONFIG_OPENTHREAD_BR_AUTO_START
#elif CONFIG_EXAMPLE_CONNECT_ETHERNET
    ESP_ERROR_CHECK(example_connect());
    esp_openthread_set_backbone_netif(get_example_netif());
#else
    ESP_LOGE(TAG, "ESP-Openthread has not set backbone netif");
#endif // CONFIG_EXAMPLE_CONNECT_WIFI

    // semaphore = xSemaphoreCreateCounting(1,0);
    ESP_ERROR_CHECK(mdns_init());
    ESP_ERROR_CHECK(mdns_hostname_set("esp-ot-br"));
    xTaskCreate(ot_task_worker, "ot_br_main", 20480, xTaskGetCurrentTaskHandle(), 5, NULL);
    vTaskDelay(15000/portTICK_PERIOD_MS);
    xTaskCreate(udp_send_task, "udp_send_task", 4096, NULL, 5, NULL);
    xTaskCreate(upload_data, "mqtt_upload", 4096, NULL, 5, NULL);
}
```

Appendix J
Code for BLE System - Node

Template Source Code:
https://github.com/espressif/esp-idf/tree/master/examples/bluetooth/esp_ble_mesh/vendor_models/vendor_server

```
#include <stdio.h>
#include <string.h>
#include <inttypes.h>

#include "esp_log.h"
#include "nvs_flash.h"
#include "esp_bt.h"

#include "esp_ble_mesh_defs.h"
#include "esp_ble_mesh_common_api.h"
#include "esp_ble_mesh_networking_api.h"
#include "esp_ble_mesh_provisioning_api.h"
#include "esp_ble_mesh_config_model_api.h"
#include "esp_ble_mesh_local_data_operation_api.h"

#include "board.h"
#include "ble_mesh_example_init.h"

#include "driver/i2c.h"
#include "driver/gpio.h"
#include <sys/unistd.h>
#include <sys/stat.h>
#include "esp_vfs_fat.h"
#include "sdmmc_cmd.h"
#include "driver/sdspi_host.h"
#include <stdlib.h>
#include "soc/soc_caps.h"
#include "esp_adc/adc_oneshot.h"
#include "esp_adc/adc_cali.h"
#include "esp_adc/adc_cali_scheme.h"
#include <math.h>
#include "driver/uart.h"

#define TAG "EXAMPLE"

#define CID_ESP     0x02E5

#define ESP_BLE_MESH_VND_MODEL_ID_CLIENT    0x0000
#define ESP_BLE_MESH_VND_MODEL_ID_SERVER    0x0001

#define ESP_BLE_MESH_VND_MODEL_OP_SEND      ESP_BLE_MESH_MODEL_OP_3(0x00, CID_ESP)
#define ESP_BLE_MESH_VND_MODEL_OP_STATUS    ESP_BLE_MESH_MODEL_OP_3(0x01, CID_ESP)

static uint8_t dev_uuid[ESP_BLE_MESH_OCTET16_LEN] = { 0x32, 0x10 };

uint16_t my_net_idx;
uint16_t my_addr;

char Current_Date_Time[20];
int latitude_decimal;
int longitude_decimal;
// SemaphoreHandle_t semaphore;
TaskHandle_t sen_handle, mics_handle, dummy, sgp_handle, finish_handle, uart_handle, ot_handle;

static const char *TAG_SDCARD = "SD_CARD";
int heated = 0;
#define TASK_COUNT 3

//GPIO Pins for LED
#define RED_LED 22
#define ORANGE_LED 23
#define YELLOW_LED 27
#define GREEN_LED 26
#define RECIRC_LED 30 //CHANGE ACCORDINGLY
#define INTAKE_LED 30

//I2C0 for SGP30
#define I2C_PORT_0 0
#define I2C0_SCL 10
#define I2C0_SDA 11

//I2C1 for SEN55 and MiCS-4514
#define I2C_PORT_1 1
#define I2C1_SCL 12
#define I2C1_SDA 8
```

```
//For Sensor I2C Addresses
#define SEN55_ADDR 0x69
#define SGP30_ADDR 0x58

//General I2C parameters
#define I2C_FREQUENCY 100000

//Global Variables for Sensor Readings
static float pm25, pm10, rh, temp, co, no2, co2, voc;
int no2_adc, co_adc;
float pm25_score, pm10_score, co_score, no2_score, co2_score, voc_score, iaq, speed_kmh;
float speed_kmh = 0;
char* location[4];
char* velocity[10];
double lat_deg = 0;
double long_deg = 0;
char* speed_str = "0";

//SPI for SD Card Module
#define PIN_NUM_MISO  0
#define PIN_NUM_MOSI  5
#define PIN_NUM_CLK   4
#define PIN_NUM_CS    1
#define MAX_SIZE 220
#define MOUNT_POINT "/sdcard"
int first_write = 1;

//ADC1 for MiCS-4514
adc_channel_t ADC1_CH1_RED = ADC_CHANNEL_1; // For RED ADC Readings
adc_channel_t ADC1_CH2_NOX = ADC_CHANNEL_2; // For NOX ADC Readings
#define PREHEAT_PIN 13 // Pin to pre-heat the module

//UART1 for Ublox Neo 6M V2
#define TX_pin 26
#define RX_pin 27

char data_payload[500] =
    "{\"COraw\":\"1000.00\",\"COindex\":\"150\",\"CO2raw\":\"60000.00\",\"CO2index\":\"150\",\"NO2raw\":\"
    1000.00\",\"NO2index\":\"150\",\"PM25raw\":\"1000.00\",\"PM25index\":\"150\",\"PM10raw\":\"1000.00\",\
    "PM10index\":\"150\",\"VOCraw\":\"10000.00\",\"VOCindex\":\"150\",\"T\":\"100.00\",\"RH\":\"100.00\",\
    "source\":\"Node 1\",\"local_time\":\"2024-05-05T12:12:12\",\"type\":\"data\"}";

//SD Card R/W Functions
static esp_err_t sd_card_write_file(const char *path, char *data)
{
    ESP_LOGI(TAG_SDCARD, "Opening file %s", path);
    FILE *f = fopen(path, "a+");
    if (f == NULL) {
        ESP_LOGE(TAG_SDCARD, "Failed to open file for writing");
        return ESP_FAIL;
    }
    fprintf(f, data);
    fclose(f);
    ESP_LOGI(TAG_SDCARD, "File written");

    return ESP_OK;
}
static esp_err_t sd_card_read_file(const char *path)
{
    ESP_LOGI(TAG_SDCARD, "Reading file %s", path);
    FILE *f = fopen(path, "r");
    if (f == NULL) {
        ESP_LOGE(TAG_SDCARD, "Failed to open file for reading");
        return ESP_FAIL;
    }
    char line[MAX_SIZE];
    fgets(line, sizeof(line), f);
    fclose(f);

    // strip newline
    char *pos = strchr(line, '\n');
    if (pos) {
        *pos = '\0';
    }
    ESP_LOGI(TAG_SDCARD, "Read from file: '%s'", line);

    return ESP_OK;
}
//SD Card Main Functions
void sd_card_write(void){
    esp_err_t ret;
    // Options for mounting the filesystem.
    // If format_if_mount_failed is set to true, SD card will be partitioned and
    // formatted in case when mounting fails.
    esp_vfs_fat_sdmmc_mount_config_t mount_config = {
#ifdef CONFIG_EXAMPLE_FORMAT_IF_MOUNT_FAILED
        .format_if_mount_failed = true,
#else
```

```
        .format_if_mount_failed = false,
#endif // EXAMPLE_FORMAT_IF_MOUNT_FAILED
        .max_files = 5,
        .allocation_unit_size = 16 * 1024
    };
    sdmmc_card_t *card;
    const char mount_point[] = MOUNT_POINT;
    ESP_LOGI(TAG_SDCARD, "Initializing SD card");

    // Use settings defined above to initialize SD card and mount FAT filesystem.
    ESP_LOGI(TAG_SDCARD, "Using SPI peripheral");

    // For setting a SD card frequency, use host.max_freq_khz (range 400kHz - 20MHz for SDSPI)
    sdmmc_host_t host = SDSPI_HOST_DEFAULT();

    spi_bus_config_t bus_cfg = {
        .mosi_io_num = PIN_NUM_MOSI,
        .miso_io_num = PIN_NUM_MISO,
        .sclk_io_num = PIN_NUM_CLK,
        .quadwp_io_num = -1,
        .quadhd_io_num = -1,
        .max_transfer_sz = 4000,
    };
    ret = spi_bus_initialize(host.slot, &bus_cfg, SPI_DMA_CH_AUTO);
    if (ret != ESP_OK) {
        ESP_LOGE(TAG_SDCARD, "Failed to initialize bus.");
        return;
    }

    sdspi_device_config_t slot_config = SDSPI_DEVICE_CONFIG_DEFAULT(); //SD Card Module has no CS and WP
     i/o
    slot_config.gpio_cs = PIN_NUM_CS;
    slot_config.host_id = host.slot;

    ESP_LOGI(TAG_SDCARD, "Mounting to SD Card...");
    ret = esp_vfs_fat_sdspi_mount(mount_point, &host, &slot_config, &mount_config, &card);
    if (ret != ESP_OK) {
        if (ret == ESP_FAIL) {
            ESP_LOGE(TAG_SDCARD, "Failed to mount");
        } else {
            ESP_LOGE(TAG_SDCARD, "Failed to initialize the card (%s).", esp_err_to_name(ret));
        }
        return;
    }
    ESP_LOGI(TAG_SDCARD, "SD Card mounted");

    // Card has been initialized, print its properties
    //sdmmc_card_print_info(stdout, card);

    // Directory for "aq_log.txt"
    const char *aq_log_file = MOUNT_POINT"/aq_log.csv";
    char log[MAX_SIZE];

    if (first_write == 1){
     //Write Headers
     snprintf(log, MAX_SIZE, "Time, CO (ppb), CO Index, CO2 (ppm), CO2 Index, NO2 (ppb), NO2 Index, PM2.5
     (ug/m3), PM2.5 Index, PM10 (ug/m3), PM10 Index, VOC (ppb), VOC Index, Temperature (C), RH (%s),
     Latitude, Longitude, Speed (km/h)\n", "%%");
     ret = sd_card_write_file(aq_log_file, log);
        if (ret != ESP_OK) {
            return;
        }
     first_write = 0;
    }
    // Set up data to be logged
    snprintf(log, MAX_SIZE, "%s, %.2f, %.0f, %.2f, %.0f, %.2f, %.0f, %.2f, %.0f, %.2f, %.0f, %.2f, %.0f,
     %.2f, %.2f, %.6f, %.6f, %.2f\n", Current_Date_Time, co, co_score, co2, co2_score, no2, no2_score,
     pm25, pm25_score, pm10, pm10_score, voc, voc_score, temp, rh, lat_deg, long_deg, speed_kmh);
    //printf("%s\n", log);
    // Write data log to aq_log.txt
    ret = sd_card_write_file(aq_log_file, log);
    if (ret != ESP_OK) {
        return;
    }
    //Open file for reading
/*    ret = sd_card_read_file(aq_log_file);
    if (ret != ESP_OK) {
        return;
    }*/

    // All done, unmount partition and disable SPI peripheral
    esp_vfs_fat_sdcard_unmount(mount_point, card);
    ESP_LOGI(TAG_SDCARD, "Card unmounted");

    //deinitialize the bus after all devices are removed
    spi_bus_free(host.slot);
}
```

```
//Functions for LED Alert System
void led_alert_init (void){
    gpio_reset_pin(RED_LED);
    gpio_set_direction(RED_LED, GPIO_MODE_OUTPUT);
    gpio_reset_pin(ORANGE_LED);
    gpio_set_direction(ORANGE_LED, GPIO_MODE_OUTPUT);
    gpio_reset_pin(YELLOW_LED);
    gpio_set_direction(YELLOW_LED, GPIO_MODE_OUTPUT);
    gpio_reset_pin(GREEN_LED);
    gpio_set_direction(GREEN_LED, GPIO_MODE_OUTPUT);
    gpio_set_level(RED_LED, 1);
    gpio_set_level(ORANGE_LED, 1);
    gpio_set_level(YELLOW_LED, 1);
    gpio_set_level(GREEN_LED, 1);
}
void update_led_alert (float pm25, float pm10, float rh, float temp, float co, float no2, float co2, float
    voc){
    //Condition for RED Level AQI
    if (pm25 > 30 || pm10 > 100 || co > 70 || co2 > 1500 || no2 > 250 || voc > 800){
            gpio_set_level(RED_LED, 1);
            gpio_set_level(ORANGE_LED, 0);
            gpio_set_level(YELLOW_LED, 0);
            gpio_set_level(GREEN_LED, 0);
//          led_strip_set_pixel(led_strip, 0, 255, 0, 0);
//          led_strip_refresh(led_strip);
    }
    //Condition for ORANGE Level AQI
    else if (pm25 > 20 || pm10 > 75 || co > 50 || co2 > 1150 || no2 > 175 || voc > 600){
            gpio_set_level(RED_LED, 0);
            gpio_set_level(ORANGE_LED, 1);
            gpio_set_level(YELLOW_LED, 0);
            gpio_set_level(GREEN_LED, 0);
//          led_strip_set_pixel(led_strip, 0, 255, 165, 0);
//          led_strip_refresh(led_strip);
    }
    //Condition for YELLOW Level AQI
    else if (pm25 > 15 || pm10 > 50 || co > 35 || co2 > 800 || no2 > 100 || voc > 400){
            gpio_set_level(RED_LED, 0);
            gpio_set_level(ORANGE_LED, 0);
            gpio_set_level(YELLOW_LED, 1);
            gpio_set_level(GREEN_LED, 0);
//          led_strip_set_pixel(led_strip, 0, 255, 255, 0);
//          led_strip_refresh(led_strip);
    }
    //Condition for GREEN Level AQI
    else if (pm25 > 0 || pm10 > 0 || co > 0 || co2 > 0 || no2 > 0 || voc > 0){
            gpio_set_level(RED_LED, 0);
            gpio_set_level(ORANGE_LED, 0);
            gpio_set_level(YELLOW_LED, 0);
            gpio_set_level(GREEN_LED, 1);
//          led_strip_set_pixel(led_strip, 0, 0, 255, 0);
//          led_strip_refresh(led_strip);
    }
    //negative values are encountered, possible code error
    else{
            gpio_set_level(RED_LED, 0.4);
            gpio_set_level(ORANGE_LED, 0.4);
            gpio_set_level(YELLOW_LED, 0.4);
            gpio_set_level(GREEN_LED, 0.4);
    }
}
void iaq_score(float pm25, float pm10, float co, float no2, float co2, float voc){
    float index[4][2] = {{0,50}, {51,75}, {76,100}, {101,150}};
    float co_bp[4][2] = {{0,35}, {36,50}, {51,70}, {70,1000}};
    float co2_bp[4][2] = {{0,800}, {801,1150}, {1151,1500}, {1501,60000}};
    float pm25_bp[4][2] = {{0,15}, {16,20}, {21,30}, {31,1000}};
    float pm10_bp[4][2] = {{0,50}, {51,75}, {76,100}, {101,1000}};
    float no2_bp[4][2] = {{0,100}, {101,175}, {176,250}, {251,10000}};
    float voc_bp[4][2] = {{0,400}, {401,600}, {601,800}, {801,60000}};
    //PM2.5
    for (int j=3; j>=0; j--){
            if(pm25 >= pm25_bp[j][0]){
                    pm25_score =
    (((index[j][1]-index[j][0])/(pm25_bp[j][1]-pm25_bp[j][0]))*(pm25-pm25_bp[j][0]))+(index[j][0]);
                    break;
            }
    }
    //PM10
    for (int j=3; j>=0; j--){
            if(pm10 >= pm10_bp[j][0]){
                    pm10_score =
    (((index[j][1]-index[j][0])/(pm10_bp[j][1]-pm10_bp[j][0]))*(pm10-pm10_bp[j][0]))+(index[j][0]);
                    break;
            }
    }
    //CO
    for (int j=3; j>=0; j--){
            if(co >= co_bp[j][0]){
                    co_score =
    (((index[j][1]-index[j][0])/(co_bp[j][1]-co_bp[j][0]))*(co-co_bp[j][0]))+(index[j][0]);
                    break;
            }
    }
    //NO2
    for (int j=3; j>=0; j--){
            if(no2 >= no2_bp[j][0]){
                    no2_score =
    (((index[j][1]-index[j][0])/(no2_bp[j][1]-no2_bp[j][0]))*(no2-no2_bp[j][0]))+(index[j][0]);
                    break;
            }
    }
    //CO2
    for (int j=3; j>=0; j--){
            if(co2 >= co2_bp[j][0]){
                    co2_score =
    (((index[j][1]-index[j][0])/(co2_bp[j][1]-co2_bp[j][0]))*(co2-co2_bp[j][0]))+(index[j][0]);
                    break;
            }
    }
    //VOC
    for (int j=3; j>=0; j--){
            if(voc >= voc_bp[j][0]){
                    voc_score =
    (((index[j][1]-index[j][0])/(voc_bp[j][1]-voc_bp[j][0]))*(voc-voc_bp[j][0]))+(index[j][0]);
                    break;
            }
    }

}
void driver_alert_init (void){
    gpio_reset_pin(RECIRC_LED);
    gpio_set_direction(RECIRC_LED, GPIO_MODE_OUTPUT);
    gpio_reset_pin(INTAKE_LED);
    gpio_set_direction(INTAKE_LED, GPIO_MODE_OUTPUT);
    gpio_set_level(INTAKE_LED, 0);
    gpio_set_level(RECIRC_LED, 0);
}
void driver_alert(float pm25_in, float pm10_in, float co_in, float no2_in, float co2_in, float voc_in,
                                float pm25_out, float pm10_out, float co_out, float no2_out, float
    co2_out, float voc_out){
    if (co_in > co_out || no2_in > no2_out || co2_in > co2_out || voc_in > voc_out){
            gpio_set_level(INTAKE_LED, 1); // Accumulated gas concentrations (Orange level)
            gpio_set_level(RECIRC_LED, 0);
    }
    else{
            gpio_set_level(INTAKE_LED, 0);
            gpio_set_level(RECIRC_LED, 1);
    }
}

//Warm-Up for MiCS-4514 for 3 minutes
void sensor_init(int *heated_var){
    gpio_reset_pin(PREHEAT_PIN);
    gpio_set_direction(PREHEAT_PIN, GPIO_MODE_OUTPUT);
    gpio_set_level(PREHEAT_PIN, 1);

    int i2c_master_port0 = I2C_PORT_0;
    i2c_config_t conf0 = {
        .mode = I2C_MODE_MASTER,
        .sda_io_num = I2C0_SDA,
        .scl_io_num = I2C0_SCL,
        .sda_pullup_en = GPIO_PULLUP_ENABLE,
        .scl_pullup_en = GPIO_PULLUP_ENABLE,
        .master.clk_speed = I2C_FREQUENCY,
    };
    i2c_param_config(i2c_master_port0, &conf0);
    i2c_driver_install(i2c_master_port0, conf0.mode, 0, 0, 0);
    uint8_t SGP30_INIT[2] = {0x20, 0x03}; //Initialize SGP30
    i2c_master_write_to_device(i2c_master_port0, SGP30_ADDR, SGP30_INIT, 2, 1000/portTICK_PERIOD_MS);
    if(heated_var == 0){
            printf("Preheating MiCS-4514...\n");
            vTaskDelay(180000/portTICK_PERIOD_MS); // Preheat MiCS for 3 minutes
    }
    else{
            printf("Waiting for SGP30...\n");
            vTaskDelay(1000/portTICK_PERIOD_MS);
    }
}
//For mapping MiCS-4514 raw readings
void process_mics4514(int D_co, int D_no2) {
    //Source https://myscope.net/auswertung-der-airpi-gas-sensoren/
    //For CO readings
    // printf("D_OUTS: %d, %d\n", D_co, D_no2);
    double Vo_co = 3.55 * (D_co-2113)/(4081-2113);
    double Vo_no2 = 3.55 * (D_no2-2113)/(4081-2113);
    // printf("VOLTAGE: %.5f  %.5f\n", Vo_co, Vo_no2);
```

```
     double Rs_co = (5-Vo_co) / (Vo_co/820);
     double Rs_no2 = (5-Vo_no2) / (Vo_no2/820);
     // printf("RESISTANCE: %.2f  %.2f\n", Rs_co, Rs_no2);
     double R0_co = 950; // FOR CALIBRATION (4.00 * R0 = Rs @ 0.8 ppm CO)
     co = pow(10, -1.1859 * (log(Rs_co/R0_co) / M_LN10) + 0.6201); //Curve Fitting Equation from Source
     //For NO2 readings
     double R0_no2 = 1440000; // FOR CALIBRATION (0.05*R0 = Rs @ 0.01 ppm NO2)
     no2 = pow(10, 0.9682 * (log(Rs_no2/R0_no2) / M_LN10) - 0.8108); //Curve Fitting Equation from Source
     //printf("CO: %.2f, NO2: %.2f\n", co, no2);
}
//RTOS Task for MiCS-4514 @ ADC1 Channel 1 & 2
void ADC1_Data_Task(void *params){
     //Initialize ADC1 Peripheral
     adc_oneshot_unit_handle_t adc1_handle;
     adc_oneshot_unit_init_cfg_t init_config1 = {
                 .unit_id = ADC_UNIT_1,
                      .clk_src = ADC_DIGI_CLK_SRC_DEFAULT,
                      .ulp_mode = ADC_ULP_MODE_RISCV,
     };
     printf("ADC INIT\n");
     ESP_ERROR_CHECK(adc_oneshot_new_unit(&init_config1, &adc1_handle));

     //Configure the 2 ADC Channels
     adc_oneshot_chan_cfg_t config1 = {
                  .atten = ADC_ATTEN_DB_11,
                       .bitwidth = ADC_BITWIDTH_DEFAULT
     };
     ESP_ERROR_CHECK(adc_oneshot_config_channel(adc1_handle, ADC1_CH1_RED, &config1)); // For RED analog
     readings
     vTaskDelay(100/portTICK_PERIOD_MS);
     ESP_ERROR_CHECK(adc_oneshot_config_channel(adc1_handle, ADC1_CH2_NOX, &config1)); // For NOX analog
     readings
    vTaskDelay(100/portTICK_PERIOD_MS);
    while(1){
        ESP_ERROR_CHECK(adc_oneshot_read(adc1_handle, ADC1_CH1_RED, &co_adc)); // Get readings from RED
        vTaskDelay(10/portTICK_PERIOD_MS);
        ESP_ERROR_CHECK(adc_oneshot_read(adc1_handle, ADC1_CH2_NOX, &no2_adc)); // Get readings from NOX
        process_mics4514(co_adc, no2_adc);
             if(co != co){
                      co = 0;
             }
             if(no2 != no2){
                      no2 = 0;
             }
        //printf("%d", uxTaskGetStackHighWaterMark(NULL));
        // printf("MiCS-4514 Readings - CO: %.2f, NO2: %.2f\n", co, no2);
        // xSemaphoreGive(semaphore);
        // vTaskSuspend(mics_handle);
        vTaskDelay(1000/portTICK_PERIOD_MS);
    }
}
//Process SEN55 Data
void process_sen55(uint8_t data[24]){
     //Raw Bit Data
     //PM2.5 Bits
     uint16_t pm25_bits = data[3] << 8; //First 8 bits
                       pm25_bits |= data[4]; //Add last 8 bits
     //PM10 Bits
     uint16_t pm10_bits = data[9] << 8; //First 8 bits
                       pm10_bits |= data[10]; //Add last 8 bits
     //Humidity Bits
     int16_t rh_bits = data[12]; //First 8 bits
                      rh_bits <<= 8; //Shift by 8 bits
                      rh_bits |= data[13]; //Add last 8 bits
     //Temperature Bits
      int16_t temp_bits = data[15]; //First 8 bits
                       temp_bits <<= 8; //Shift by 8 bits
                       temp_bits |= data[16]; //Add last 8 bits


     //Divide by Scaling Factors
     float pm25_raw = pm25_bits;
     float pm10_raw = pm10_bits;
     float rh_raw = rh_bits;
     float temp_raw = temp_bits;
     pm25 = pm25_raw /10;
     pm10 = pm10_raw /10;
     rh = rh_raw / 100;
     temp = temp_raw / 200;
// CALIBRATION
     pm25 = 1.9467002394827233*pm25 + (2.5926092705655464);
     pm10 = 2.3350992860750863*pm10 + (1.1038457405169595);
     temp = 0.7311396775389937*temp + (5.1184212550840265);
     rh = 0.8405511007613504*rh + (-1.3350502127248518);
}
//SEN55 and SGP30 CRC Calculation
uint8_t CalcCrc(uint8_t data[2]) {
     uint8_t crc = 0xFF;
    for(int i = 0; i < 2; i++) {
            crc ^= data[i];
            for(uint8_t bit = 8; bit > 0; --bit) {
                    if(crc & 0x80) {
                            crc = (crc << 1) ^ 0x31u;
                    }
                    else {
                            crc = (crc << 1);
                    }
            }
     }
     return crc;
}
//RTOS Task for SEN55 @ I2C1
void I2C1_Data_Task(void *params){
    //SEN55 Initialize
    i2c_config_t conf1 = {
                    .mode = I2C_MODE_MASTER,
            .sda_io_num = I2C1_SDA,
            .scl_io_num = I2C1_SCL,
            .sda_pullup_en = GPIO_PULLUP_ENABLE,
            .scl_pullup_en = GPIO_PULLUP_ENABLE,
            .master.clk_speed = I2C_FREQUENCY,
    };
    i2c_param_config(I2C_PORT_1, &conf1);
    i2c_driver_install(I2C_PORT_1, conf1.mode, 0, 0, 0);
    uint8_t SEN55_MEAS_INIT[2] = {0x00,0x21}; //Start Measurement Mode
    uint8_t sen55_data[24];
    uint8_t SEN55_READ_DATA[2] = {0x03,0xC4}; //Get Measured Values Command
    ESP_ERROR_CHECK(i2c_master_write_to_device(I2C_PORT_1, SEN55_ADDR, SEN55_MEAS_INIT, 2,
    1000/portTICK_PERIOD_MS)); //Measurement Mode
    vTaskDelay(10/portTICK_PERIOD_MS);
    while (1) {
    //For SEN55 (Starts at Idle Mode)
    ESP_ERROR_CHECK(i2c_master_write_to_device(I2C_PORT_1, SEN55_ADDR, SEN55_READ_DATA, 2,
    1000/portTICK_PERIOD_MS)); //Send Get-Measurement Command
    vTaskDelay(20/portTICK_PERIOD_MS);
    ESP_ERROR_CHECK(i2c_master_read_from_device(I2C_PORT_1, SEN55_ADDR, sen55_data, 24,
    1000/portTICK_PERIOD_MS)); //Read Measurements
    vTaskDelay(10/portTICK_PERIOD_MS);
    process_sen55(sen55_data); //Update Global Variables
    //printf("%d", uxTaskGetStackHighWaterMark(NULL));
    // printf("SEN55 Readings : PM2.5: %.2f, PM10: %.2f, Temperature: %.02f, RH: %.2f\n", pm25, pm10,
    temp, rh);
        // xSemaphoreGive(semaphore);
        // vTaskSuspend(sen_handle);
        vTaskDelay(1000/portTICK_PERIOD_MS);
    }
}
//RTOS Task for SGP30 @ I2C0
void I2C0_Data_Task(void *params){
    int i2c_master_port = I2C_PORT_0;
    uint8_t sgp30_data[6];
    uint8_t SGP30_MEAS[2] = {0x20, 0x08};
    while (true) {
            ESP_ERROR_CHECK(i2c_master_write_to_device(i2c_master_port, SGP30_ADDR, SGP30_MEAS, 2,
    1000/portTICK_PERIOD_MS)); //Send Measurement Command
    vTaskDelay(15/portTICK_PERIOD_MS);
            i2c_master_read_from_device(i2c_master_port, SGP30_ADDR, sgp30_data, 6,
    1000/portTICK_PERIOD_MS); //Read Measured Values
            //Process received bit sequence
            uint16_t co2_bits = sgp30_data[0] << 8;
            co2_bits |= sgp30_data[1];
            uint16_t voc_bits = sgp30_data[3] << 8;
            voc_bits |= sgp30_data[4];
            float co2_raw = co2_bits;
            float voc_raw = voc_bits;
            co2 = co2_raw;
            voc = voc_raw;
            // printf("SGP30 Readings - CO2: %.2f, VOC: %.2f\n", co2, voc);
    // xSemaphoreGive(semaphore);
    // vTaskSuspend(sgp_handle); // So only 1 semaphore is given in case a task is faster than the other
    vTaskDelay(1000/portTICK_PERIOD_MS);
    }
}
//RTOS Task for All_data_collected
// void Data_Complete_Task (void *params){
//  while(1){
//          if(uxSemaphoreGetCount(semaphore) == TASK_COUNT){
//                  for (int i=0; i == TASK_COUNT; i++){
//                          xSemaphoreTake(semaphore, portMAX_DELAY);
//                  }
//                  //printf("Data Upload/Network and Alerts Task\n\n");
//                  //printf("SGP30 Task State: %d\n", eTaskGetState(sgp_handle));
//                  //printf("MiCS-4514 Task State: %d\n", eTaskGetState(mics_handle));
//                  //printf("SEN55 Task State: %d\n\n", eTaskGetState(sen_handle));
//                  // ReadAttribute(); // For Zigbee
//                  // vTaskDelay(1000/portTICK_PERIOD_MS); // For Zigbee
//                  // snprintf(Current_Date_Time, sizeof(Current_Date_Time), "%s", Rx_Date_Time);
//                  // printf("%s\n", Current_Date_Time);
//                  // Current_Date_Time = Rx_Date_Time;
//                  update_led_alert(pm25, pm10, temp, rh, co, no2, co2, voc);
//                  iaq_score(pm25, pm10, co, no2, co2, voc);

//          // ZIGBEE
//                  // snprintf(data1, sizeof(data1), "%.0f,%.2f,%.0f,%.2f,%.2f,%.2f,%.0f,%.2f",
    pm25_score, pm25, pm10_score, pm10, temp, rh, co2_score, co2);
//                  // snprintf(data2, sizeof(data2), "%.0f,%.2f,%.0f,%.2f,%.0f,%.2f", voc_score, voc,
    co_score, co, no2_score, no2);
//                  // snprintf(data3, sizeof(data3), "%.7f,%.7f,%.2f,%s", lat_deg, long_deg, speed_kmh,
    Current_Date_Time); // Lat, Long, Speed, Time
//              // esp_zb_zcl_attr_t *value_r = esp_zb_zcl_get_attribute(ZB_CLIENT_ENDPOINT_4,
    SENSOR_DATA_CLUSTER_ID, ESP_ZB_ZCL_CLUSTER_SERVER_ROLE, SENSOR_DATA_ATTR_ID_3);
//              // snprintf(data3, sizeof(data3), "%s%s", (char *)value_r->data_p, Current_Date_Time);
//                  // printf("%s\n", data1);
//                  // printf("%s\n", data2);
//                  // printf("%s\n", data3);
//                  // vTaskDelay(100/portTICK_PERIOD_MS);
//                  // ReportAttribute(SENSOR_DATA_ATTR_ID);
//                  // vTaskDelay(100/portTICK_PERIOD_MS);
//                  // ReportAttribute(SENSOR_DATA_ATTR_ID_2);
//                  // vTaskDelay(100/portTICK_PERIOD_MS);
//                  // ReportAttribute(SENSOR_DATA_ATTR_ID_3);

//          //Thread
//                  // snprintf(data_payload, 500,
    "{\"COraw\":\"%.2f\",\"COindex\":\"%.0f\",\"CO2raw\":\"%.2f\",\"CO2index\":\"%.0f\",\"NO2raw\":\"%.2f\
    ",\"NO2index\":\"%.0f\",\"PM25raw\":\"%.2f\",\"PM25index\":\"%.0f\",\"PM10raw\":\"%.2f\",\"PM10index\"
    :\"%.0f\",\"VOCraw\":\"%.2f\",\"VOCindex\":\"%.0f\",\"T\":\"%.2f\",\"RH\":\"%.2f\",\"source\":\"Node
    1\",\"local_time\":\"%s\",\"type\":\"data\"}",co,co_score,co2,co2_score,no2,no2_score,pm25,pm25_score,
    pm10,pm10_score,voc,voc_score,temp,rh,Current_Date_Time);
//                  // snprintf(data_payload, 150,
    "1,%.0f,%.2f,%.0f,%.2f,%.2f,%.2f,%.0f,%.2f,%.0f,%.2f,%.0f,%.2f,%.0f,%.2f,%s", pm25_score, pm25,
    pm10_score, pm10, temp, rh, co2_score, co2,voc_score, voc, co_score, co, no2_score, no2,
    Current_Date_Time);
//                  // printf("%s\n", data_payload);
//                  // printf("Data String: %s\n", data_payload);
//                  // udp_send();

//           snprintf(data_payload, 500,
    "{\"COraw\":\"%.2f\",\"COindex\":\"%.0f\",\"CO2raw\":\"%.2f\",\"CO2index\":\"%.0f\",\"NO2raw\":\"%.2f\
    ",\"NO2index\":\"%.0f\",\"PM25raw\":\"%.2f\",\"PM25index\":\"%.0f\",\"PM10raw\":\"%.2f\",\"PM10index\"
    :\"%.0f\",\"VOCraw\":\"%.2f\",\"VOCindex\":\"%.0f\",\"T\":\"%.2f\",\"RH\":\"%.2f\",\"source\":\"Node
    1\",\"local_time\":\"%s\",\"type\":\"data\"}",co,co_score,co2,co2_score,no2,no2_score,pm25,pm25_score,
    pm10,pm10_score,voc,voc_score,temp,rh,Current_Date_Time);

//                  sd_card_write();
//                  vTaskDelay(30000/portTICK_PERIOD_MS);
//                  vTaskResume(sgp_handle);
//                  vTaskResume(mics_handle);
//                  vTaskResume(sen_handle);
//                  //vTaskResume(uart_handle);
//          }
//          else{
//                  vTaskDelay(1000/portTICK_PERIOD_MS);
//          }
//  }
// }
 //GPS (UART), Driver Alert
void UART_Data_Task(void *params){
    //Start UART Here
    const uart_port_t uart_num = UART_NUM_1;
    uart_config_t uart_config = {
        .baud_rate = 9600, //GPS baud rate according to datasheet
        .data_bits = UART_DATA_8_BITS,
        .parity = UART_PARITY_DISABLE,
        .stop_bits = UART_STOP_BITS_1,
        .flow_ctrl = UART_HW_FLOWCTRL_DISABLE,
    };
    // Configure UART parameters
    ESP_ERROR_CHECK(uart_param_config(uart_num, &uart_config));
    // Set UART pins(TX: IO4, RX: IO5, RTS: IO18, CTS: IO19)
    ESP_ERROR_CHECK(uart_set_pin(uart_num, TX_pin, RX_pin, UART_PIN_NO_CHANGE, UART_PIN_NO_CHANGE)); //RTS
    and CTS pins will not be used
    // Setup UART buffered IO with event queue
    const int uart_buffer_size = (1024*3);
    QueueHandle_t uart_queue;
    // Install UART driver using an event queue here
    ESP_ERROR_CHECK(uart_driver_install(uart_num, uart_buffer_size, uart_buffer_size, 10, &uart_queue,
    0));
    vTaskDelay(100/portTICK_PERIOD_MS);
    const char* filter_str_loc = "$GPGGA";
    const char* filter_str_speed = "$GPVTG";
    char *loc = NULL;
    char *speed = NULL;
    char str1[100];
    char str2[45];
    int j=0;
    while(1){
             int length;

             ESP_ERROR_CHECK(uart_get_buffered_data_len(uart_num, (size_t*)&length));
             char gps_out[length];

             uart_read_bytes(uart_num, gps_out, length, 1000);

             loc = strstr(gps_out,filter_str_loc);
        if (loc == NULL) {
            printf("GPGGA not found in the sentence\n");
        }
        else{
                 char *newline = strchr(loc, '\n');
                 if (newline == NULL) {
                     printf("Incomplete NMEA Sentence\n");
                 }
                 else if(loc){
                                for (int i=0; loc[i] != '\n'; i++){
                                         str1[j] = loc[i];
                                         j++;
                                }
                                str1[j+1] = '\n';
                                j = 0;
                                printf("%s\n\n", str1);
                                char* token = strtok(str1, ",");
                                token = strtok(NULL, ",");
                                for (int i=0; i<4; i++){
                                         token = strtok(NULL,",");
                                         location[i] = token;
                                }
                                double latitude = atof(location[0]);
                                double longitude = atof(location[2]);

                                 latitude_decimal = floor(latitude/100);
                                 longitude_decimal = floor(longitude/100);

                                if(latitude_decimal == 0 && longitude_decimal == 0){
                                         printf("No Signal\n");
                                }
                                else{
                                         double latitude_decimal_minutes = modf(latitude/100, &latitude);
                                         latitude_decimal_minutes = latitude_decimal_minutes*100;
                                         double longitude_decimal_minutes = modf(longitude/100,
    &longitude);

                                         longitude_decimal_minutes = longitude_decimal_minutes*100;

                                         lat_deg = latitude_decimal + latitude_decimal_minutes/60;
                                         long_deg = longitude_decimal + longitude_decimal_minutes/60;
                                         //printf("%.8f, %.8f\n", lat_deg, long_deg);
                                }
                      }
        }
        //printf("%s\n\n", gps_out);
        speed_str = "0";
             speed = strstr(gps_out,filter_str_speed);
             //printf("%s", speed);
        if (speed == NULL) {
            printf("GPVTG not found in the sentence\n");
        }
        else{
                 char *newline2 = strchr(speed, '\n');
                 if (newline2 == NULL) {
                     printf("Incomplete NMEA Sentence\n");
                 }
                 else if (newline2 != NULL && latitude_decimal != 0 && longitude_decimal != 0){
                      for (int i=0; speed[i] != '\n'; i++){
                                str2[j] = speed[i];
                                j++;
                      }
                      str2[j+1] = '\n';
                      j = 0;
                      printf("%s\n\n", str2);
                      if(strlen(str2) > 20){
                                         char* token2 = strtok(str2, ",");
                                         token2 = strtok(NULL, ",");
                                         for (int i=0; i<6; i++){
                                                  token2 = strtok(NULL,",");
                                                  velocity[i] = token2;
                                                  if (strcmp(token2, "N") == 0){
                                                           token2 = strtok(NULL,",");
                                                           speed_str = token2;
                                                           break;
                                                  }
```

```
                                        }
                        }
                                printf("SPEED STRING: %s\n", speed_str);
                                printf("%.7f, %.7f\n", lat_deg, long_deg);
                                if(speed_str == NULL){
                                        strcpy(speed_str, "0");
                                }
                                speed_kmh = atof(speed_str);
                                printf("%.2f\n\n", speed_kmh);
                                if (speed_kmh < 1){
                                        speed_kmh = 1;
                                }

                }
        }

        //xSemaphoreGive(semaphore);
            //vTaskSuspend(uart_handle);
        //sd_card_write();
        char gps_data_format[30] = "00.0000000,000.0000000,00.00,";
        snprintf(gps_data_format, 30, "%.7f,%.7f,%.2f,", lat_deg, long_deg, speed_kmh);
//      esp_zb_zcl_attr_t *value_r = esp_zb_zcl_get_attribute(ZB_CLIENT_ENDPOINT_4,
    SENSOR_DATA_CLUSTER_ID, ESP_ZB_ZCL_CLUSTER_SERVER_ROLE, SENSOR_DATA_ATTR_ID_3);
//      memcpy(value_r->data_p, &gps_data_format, sizeof(gps_data_format));
        vTaskDelay(1000/portTICK_PERIOD_MS);
    }
}

static esp_ble_mesh_cfg_srv_t config_server = {
    /* 3 transmissions with 20ms interval */
    .net_transmit = ESP_BLE_MESH_TRANSMIT(2, 20),
    .relay = ESP_BLE_MESH_RELAY_DISABLED,
    .relay_retransmit = ESP_BLE_MESH_TRANSMIT(2, 20),
    .beacon = ESP_BLE_MESH_BEACON_ENABLED,
#if defined(CONFIG_BLE_MESH_GATT_PROXY_SERVER)
    .gatt_proxy = ESP_BLE_MESH_GATT_PROXY_ENABLED,
#else
    .gatt_proxy = ESP_BLE_MESH_GATT_PROXY_NOT_SUPPORTED,
#endif
#if defined(CONFIG_BLE_MESH_FRIEND)
    .friend_state = ESP_BLE_MESH_FRIEND_ENABLED,
#else
    .friend_state = ESP_BLE_MESH_FRIEND_NOT_SUPPORTED,
#endif
    .default_ttl = 7,
};

static esp_ble_mesh_model_t root_models[] = {
    ESP_BLE_MESH_MODEL_CFG_SRV(&config_server),
};

static esp_ble_mesh_model_op_t vnd_op[] = {
    ESP_BLE_MESH_MODEL_OP(ESP_BLE_MESH_VND_MODEL_OP_SEND, 2),
    ESP_BLE_MESH_MODEL_OP_END,
};

static esp_ble_mesh_model_t vnd_models[] = {
    ESP_BLE_MESH_VENDOR_MODEL(CID_ESP, ESP_BLE_MESH_VND_MODEL_ID_SERVER,
    vnd_op, NULL, NULL),
};

static esp_ble_mesh_elem_t elements[] = {
    ESP_BLE_MESH_ELEMENT(0, root_models, vnd_models),
};

static esp_ble_mesh_comp_t composition = {
    .cid = CID_ESP,
    .element_count = ARRAY_SIZE(elements),
    .elements = elements,
};

static esp_ble_mesh_prov_t provision = {
    .uuid = dev_uuid,
};

static void prov_complete(uint16_t net_idx, uint16_t addr, uint8_t flags, uint32_t iv_index)
{
    ESP_LOGI(TAG, "net_idx 0x%03x, addr 0x%04x", net_idx, addr);
    ESP_LOGI(TAG, "flags 0x%02x, iv_index 0x%08" PRIx32, flags, iv_index);
    board_led_operation(LED_G, LED_OFF);
}

static void example_ble_mesh_provisioning_cb(esp_ble_mesh_prov_cb_event_t event,
                                             esp_ble_mesh_prov_cb_param_t *param)
{
    switch (event) {
    case ESP_BLE_MESH_PROV_REGISTER_COMP_EVT:
        ESP_LOGI(TAG, "ESP_BLE_MESH_PROV_REGISTER_COMP_EVT, err_code %d",
    param->prov_register_comp.err_code);
        break;
    case ESP_BLE_MESH_NODE_PROV_ENABLE_COMP_EVT:
        ESP_LOGI(TAG, "ESP_BLE_MESH_NODE_PROV_ENABLE_COMP_EVT, err_code %d",
    param->node_prov_enable_comp.err_code);
        break;
    case ESP_BLE_MESH_NODE_PROV_LINK_OPEN_EVT:
        ESP_LOGI(TAG, "ESP_BLE_MESH_NODE_PROV_LINK_OPEN_EVT, bearer %s",
            param->node_prov_link_open.bearer == ESP_BLE_MESH_PROV_ADV ? "PB-ADV" : "PB-GATT");
        break;
    case ESP_BLE_MESH_NODE_PROV_LINK_CLOSE_EVT:
        ESP_LOGI(TAG, "ESP_BLE_MESH_NODE_PROV_LINK_CLOSE_EVT, bearer %s",
            param->node_prov_link_close.bearer == ESP_BLE_MESH_PROV_ADV ? "PB-ADV" : "PB-GATT");
        break;
    case ESP_BLE_MESH_NODE_PROV_COMPLETE_EVT:
        ESP_LOGI(TAG, "ESP_BLE_MESH_NODE_PROV_COMPLETE_EVT");
        prov_complete(param->node_prov_complete.net_idx, param->node_prov_complete.addr,
            param->node_prov_complete.flags, param->node_prov_complete.iv_index);

        my_net_idx = param->node_prov_complete.net_idx;
        my_addr = param->node_prov_complete.addr;

        break;
    case ESP_BLE_MESH_NODE_PROV_RESET_EVT:
        ESP_LOGI(TAG, "ESP_BLE_MESH_NODE_PROV_RESET_EVT");
        break;
    case ESP_BLE_MESH_NODE_SET_UNPROV_DEV_NAME_COMP_EVT:
        ESP_LOGI(TAG, "ESP_BLE_MESH_NODE_SET_UNPROV_DEV_NAME_COMP_EVT, err_code %d",
    param->node_set_unprov_dev_name_comp.err_code);
        break;
    default:
        break;
    }
}

static void example_ble_mesh_config_server_cb(esp_ble_mesh_cfg_server_cb_event_t event,
                                              esp_ble_mesh_cfg_server_cb_param_t *param)
{
    if (event == ESP_BLE_MESH_CFG_SERVER_STATE_CHANGE_EVT) {
        switch (param->ctx.recv_op) {
        case ESP_BLE_MESH_MODEL_OP_APP_KEY_ADD:
            ESP_LOGI(TAG, "ESP_BLE_MESH_MODEL_OP_APP_KEY_ADD");
            ESP_LOGI(TAG, "net_idx 0x%04x, app_idx 0x%04x",
                param->value.state_change.appkey_add.net_idx,
                param->value.state_change.appkey_add.app_idx);
            ESP_LOG_BUFFER_HEX("AppKey", param->value.state_change.appkey_add.app_key, 16);
            break;
        case ESP_BLE_MESH_MODEL_OP_MODEL_APP_BIND:
            ESP_LOGI(TAG, "ESP_BLE_MESH_MODEL_OP_MODEL_APP_BIND");
            ESP_LOGI(TAG, "elem_addr 0x%04x, app_idx 0x%04x, cid 0x%04x, mod_id 0x%04x",
                param->value.state_change.mod_app_bind.element_addr,
                param->value.state_change.mod_app_bind.app_idx,
                param->value.state_change.mod_app_bind.company_id,
                param->value.state_change.mod_app_bind.model_id);
            break;
        default:
            break;
        }
    }
}

static void example_ble_mesh_custom_model_cb(esp_ble_mesh_model_cb_event_t event,
                                             esp_ble_mesh_model_cb_param_t *param)
{
    switch (event) {
    case ESP_BLE_MESH_MODEL_OPERATION_EVT:
        // if (param->model_operation.opcode == ESP_BLE_MESH_VND_MODEL_OP_SEND) {
        //     uint16_t tid = *(uint16_t *)param->model_operation.msg;
        //     ESP_LOGI(TAG, "Recv 0x%06" PRIx32 ", tid 0x%04x", param->model_operation.opcode, tid);
        //     esp_err_t err = esp_ble_mesh_server_model_send_msg(&vnd_models[0],
        //             param->model_operation.ctx, ESP_BLE_MESH_VND_MODEL_OP_STATUS,
        //             sizeof(tid), (uint8_t *)&tid);
        //     if (err) {
        //         ESP_LOGE(TAG, "Failed to send message 0x%06x", ESP_BLE_MESH_VND_MODEL_OP_STATUS);
        //     }
        // }
        if (param->model_operation.opcode == ESP_BLE_MESH_VND_MODEL_OP_SEND) {
            update_led_alert(pm25, pm10, temp, rh, co, no2, co2, voc);
                      iaq_score(pm25, pm10, co, no2, co2, voc);
            uint16_t tid = *(uint16_t *)param->model_operation.msg;
            // char *mydata =
    "id1,18,5.90,6,5.90,32.03,49.86,0,400.00,0,0.00,2,1.77,0,0.07,14.699734,121.723847,23.55";
    //"id1,18,5.90,6,5.90,32.03,49.86,0,400.00,0,0.00,2,1.77,0,0.07,14.699734,121.723847,23.55"
            // char *mydata =
    "{\"COraw\":\"0000.00\",\"COindex\":\"000\",\"CO2raw\":\"0000.00\",\"CO2index\":\"000\",\"NO2raw\":\"0
    000.00\",\"NO2index\":\"000\",\"PM25raw\":\"0000.00\",\"PM25index\":\"000\",\"PM10raw\":\"0000.00\",\"
    PM10index\":\"000\",\"VOCraw\":\"0000.00\",\"VOCindex\":\"000\",\"T\":\"0000.00\",\"RH\":\"0000.00\",\
    "Lat\":\"000.0000000\",\"Long\":\"000.0000000\",\"speed\":\"000.00\"\"source\":\"Node
    4\",\"local_time\":\"2024-06-01T15:15:15\",\"type\":\"data\"}";

            ESP_LOGI(TAG, "Recv 0x%06" PRIx32 ", tid 0x%04x", param->model_operation.opcode, tid);
            snprintf(Current_Date_Time, 20, "%s", (char *)param->model_operation.msg);
            ESP_LOGI(TAG, "%s", Current_Date_Time);
            snprintf(data_payload, 500,
    "{\"COraw\":\"%.2f\",\"COindex\":\"%.0f\",\"CO2raw\":\"%.2f\",\"CO2index\":\"%.0f\",\"NO2raw\":\"%.2f\
    ",\"NO2index\":\"%.0f\",\"PM25raw\":\"%.2f\",\"PM25index\":\"%.0f\",\"PM10raw\":\"%.2f\",\"PM10index\"
    :\"%.0f\",\"VOCraw\":\"%.2f\",\"VOCindex\":\"%.0f\",\"T\":\"%.2f\",\"RH\":\"%.2f\",\"Lat\":\"%.7f\",\"
    Long\":\"%.7f\",\"speed\":\"%.2f\",\"source\":\"Node
    4\",\"local_time\":\"%s\",\"type\":\"data\"}",co,co_score,co2,co2_score,no2,no2_score,pm25,pm25_score,
    pm10,pm10_score,voc,voc_score,temp,rh,lat_deg,long_deg,speed_kmh,Current_Date_Time);
                      sd_card_write();
            esp_err_t err = esp_ble_mesh_server_model_send_msg(&vnd_models[0],
                    param->model_operation.ctx, ESP_BLE_MESH_VND_MODEL_OP_STATUS,
//                    sizeof(tid), (uint8_t *)&tid);
                      strlen(data_payload)+1, (uint8_t *)data_payload);
            if (err) {
                ESP_LOGE(TAG, "Failed to send message 0x%06x", ESP_BLE_MESH_VND_MODEL_OP_STATUS);
            }
            printf("string sent: %s\n", data_payload);
        }
        break;
    case ESP_BLE_MESH_MODEL_SEND_COMP_EVT:
        if (param->model_send_comp.err_code) {
            ESP_LOGE(TAG, "Failed to send message 0x%06" PRIx32, param->model_send_comp.opcode);
            break;
        }
        ESP_LOGI(TAG, "Send 0x%06" PRIx32, param->model_send_comp.opcode);
        break;
    // case ESP_BLE_MESH_MODEL_PUBLISH_COMP_EVT:
    //     //time
    //     break;
    default:
        break;
    }
}

static esp_err_t ble_mesh_init(void)
{
    esp_err_t err;

    esp_ble_mesh_register_prov_callback(example_ble_mesh_provisioning_cb);
    esp_ble_mesh_register_config_server_callback(example_ble_mesh_config_server_cb);
    esp_ble_mesh_register_custom_model_callback(example_ble_mesh_custom_model_cb);

    err = esp_ble_mesh_init(&provision, &composition);
    if (err != ESP_OK) {
        ESP_LOGE(TAG, "Failed to initialize mesh stack");
        return err;
    }

    err = esp_ble_mesh_node_prov_enable((esp_ble_mesh_prov_bearer_t)(ESP_BLE_MESH_PROV_ADV |
    ESP_BLE_MESH_PROV_GATT));
    if (err != ESP_OK) {
        ESP_LOGE(TAG, "Failed to enable mesh node");
        return err;
    }

    board_led_operation(LED_G, LED_ON);

    ESP_LOGI(TAG, "BLE Mesh Node initialized");

    return ESP_OK;
}

void example_ble_mesh_send_vendor_message(void)
{
    esp_ble_mesh_msg_ctx_t ctx = {0};
    uint32_t opcode;
    esp_err_t err;

    ctx.net_idx = my_net_idx;
    ctx.app_idx = 0x0000;
    ctx.addr = 0x0001;
    ctx.send_ttl = 3;
    opcode = ESP_BLE_MESH_VND_MODEL_OP_SEND;

    char *mydata = "{\"PM25raw\":\"0000.00\",\"PM25index\":\"000\",\"}";

    err = esp_ble_mesh_server_model_send_msg(&vnd_models[0], &ctx, opcode, strlen(mydata)+1, (uint8_t
    *)mydata);
    if (err != ESP_OK) {
        ESP_LOGE(TAG, "Failed to send vendor message 0x%06" PRIx32, opcode);
        return;
    }
}
```

```
void app_main(void)
{
    esp_err_t err;

    ESP_LOGI(TAG, "Initializing...");

    err = nvs_flash_init();
    if (err == ESP_ERR_NVS_NO_FREE_PAGES) {
        ESP_ERROR_CHECK(nvs_flash_erase());
        err = nvs_flash_init();
    }
    ESP_ERROR_CHECK(err);

    board_init();

    err = bluetooth_init();
    if (err) {
        ESP_LOGE(TAG, "esp32_bluetooth_init failed (err %d)", err);
        return;
    }

    ble_mesh_get_dev_uuid(dev_uuid);

    /* Initialize the Bluetooth Mesh Subsystem */
    err = ble_mesh_init();
    if (err) {
        ESP_LOGE(TAG, "Bluetooth mesh init failed (err %d)", err);
    }

    // semaphore = xSemaphoreCreateCounting(TASK_COUNT, 0);
     heated = 1;

     //configure_led();
     led_alert_init();

     sensor_init(&heated);
    xTaskCreate(ADC1_Data_Task, "ADC1 Data Task", 1024*3, NULL, 5, &mics_handle);
    xTaskCreate(I2C0_Data_Task, "I2C0 Data Task", 1024*3, NULL, 5, &sgp_handle);
    xTaskCreate(I2C1_Data_Task, "I2C1 Data Task", 1024*2, NULL, 5, &sen_handle);
    xTaskCreate(UART_Data_Task, "UART Data Task", 1024*8, NULL, 5, &uart_handle);
    // xTaskCreate(Data_Complete_Task, "Data Upload and Save", 4096, NULL, 5, &finish_handle);
}
```

Appendix K
Code for BLE System - Gateway

Template Code Source:
https://github.com/espressif/esp-idf/tree/master/examples/bluetooth/esp_ble_mesh/vendor_models/vendor_client

```
#include <stdio.h>
#include <string.h>
#include <inttypes.h>

#include "esp_log.h"
#include "nvs_flash.h"
#include "esp_bt.h"
#include "esp_timer.h"

#include "esp_ble_mesh_defs.h"
#include "esp_ble_mesh_common_api.h"
#include "esp_ble_mesh_provisioning_api.h"
#include "esp_ble_mesh_networking_api.h"
#include "esp_ble_mesh_config_model_api.h"

#include "ble_mesh_example_init.h"
#include "ble_mesh_example_nvs.h"
//#include "board.h"

#include "mqtt_client.h"
#include <esp_wifi_types.h>
#include "freertos/FreeRTOS.h"
#include "freertos/task.h"
#include "esp_netif.h"
#include "esp_wifi.h"
#include "esp_sntp.h"

#define TAG "EXAMPLE"

#define CID_ESP             0x02E5

#define PROV_OWN_ADDR       0x0001

#define MSG_SEND_TTL        3
#define MSG_TIMEOUT         0
#define MSG_ROLE            ROLE_PROVISIONER

#define COMP_DATA_PAGE_0    0x00

#define APP_KEY_IDX         0x0000
#define APP_KEY_OCTET       0x12

#define COMP_DATA_1_OCTET(msg, offset)      (msg[offset])
#define COMP_DATA_2_OCTET(msg, offset)      (msg[offset + 1] << 8 | msg[offset])

#define ESP_BLE_MESH_VND_MODEL_ID_CLIENT    0x0000
#define ESP_BLE_MESH_VND_MODEL_ID_SERVER    0x0001

#define ESP_BLE_MESH_VND_MODEL_OP_SEND      ESP_BLE_MESH_MODEL_OP_3(0x00, CID_ESP)
#define ESP_BLE_MESH_VND_MODEL_OP_STATUS    ESP_BLE_MESH_MODEL_OP_3(0x01, CID_ESP)

static uint8_t dev_uuid[ESP_BLE_MESH_OCTET16_LEN];

esp_mqtt_client_handle_t MQTtest;

#if CONFIG_BROKER_CERTIFICATE_OVERRIDDEN == 1
    static const uint8_t mqtt_eclipseprojects_io_pem_start[]  = "-----BEGIN CERTIFICATE-----\n"
    CONFIG_BROKER_CERTIFICATE_OVERRIDE "\n-----END CERTIFICATE-----";
#else
    extern const uint8_t mqtt_eclipseprojects_io_pem_start[]
    asm("_binary_mqtt_eclipseprojects_io_pem_start");
#endif
    extern const uint8_t mqtt_eclipseprojects_io_pem_end[]
    asm("_binary_mqtt_eclipseprojects_io_pem_end");

static const char *TAG_MQTT = "MQTTS_EXAMPLE";


char payload_1[500] = "";
char payload_2[500] = "";
char payload_3[500] = "";
char payload_4[500] = "";

char Current_Date_Time[20];

void time_sync_notification_cb(struct timeval *tv)
```

```
{
    ESP_LOGI(TAG, "Notification of a time synchronization event");
}

//RETURNS the final time in string form
void Get_current_date_time(char *date_time){
     char strftime_buf[100];
     time_t now;
         struct tm timeinfo;
         time(&now);
         localtime_r(&now, &timeinfo);

              // Set timezone to Indian Standard Time
                               setenv("TZ", "UTC-08:00", 1);
                  tzset();
                  localtime_r(&now, &timeinfo);

                  strftime(strftime_buf, sizeof(strftime_buf), "%c", &timeinfo);
                  //strftime(date_time, 100, "%Y-%m-%dT%H:%M:%S", &timeinfo);
                  //ESP_LOGI(TAG, "The current date/time in Philippines is: %s", strftime_buf);

                  // Extract the time values manually
                  int year = timeinfo.tm_year + 1900;  // Years since 1900, so add 1900
                  int month = timeinfo.tm_mon + 1;     // Months are 0-based, so add 1
                  int day = timeinfo.tm_mday;
                  int hour = timeinfo.tm_hour;
                  int minute = timeinfo.tm_min;
                  int second = timeinfo.tm_sec;

                  //concatenate the string manually
                  sprintf(strftime_buf, "%d-%02d-%02dT%02d:%02d:%02d", year, month, day, hour, minute,
     second);
                strcpy(date_time,strftime_buf);
}

//initializes SNTP server settings
static void initialize_sntp(void)
{
    ESP_LOGI(TAG, "Initializing SNTP");
    esp_sntp_setoperatingmode(SNTP_OPMODE_POLL);
    esp_sntp_setservername(0, "time.google.com");
    sntp_set_time_sync_notification_cb(time_sync_notification_cb);
#ifdef CONFIG_SNTP_TIME_SYNC_METHOD_SMOOTH
    sntp_set_sync_mode(SNTP_SYNC_MODE_SMOOTH);
#endif
    esp_sntp_init();
}

// helper function that obtains timezone
static void obtain_time(void)
{
    initialize_sntp();
    // wait for time to be set
    time_t now = 0;
    struct tm timeinfo = { 0 };
    int retry = 0;
    const int retry_count = 10;
    while (sntp_get_sync_status() == SNTP_SYNC_STATUS_RESET && ++retry < retry_count) {
        ESP_LOGI(TAG, "Waiting for system time to be set... (%d/%d)", retry, retry_count);
        vTaskDelay(2000 / portTICK_PERIOD_MS);
    }
    time(&now);
    localtime_r(&now, &timeinfo);
}

//function that actually syncs with NTP server
 void Set_SystemTime_SNTP()  {

      time_t now;
         struct tm timeinfo;
         time(&now);
         localtime_r(&now, &timeinfo);
         // Is time set? If not, tm_year will be (1970 - 1900).
         if (timeinfo.tm_year < (2016 - 1900)) {
             ESP_LOGI(TAG, "Time is not set yet. Connecting to WiFi and getting time over NTP.");
             obtain_time();
             // update 'now' variable with current time
             time(&now);
         }
}

static void wifi_event_handler(void *event_handler_arg, esp_event_base_t event_base, int32_t event_id,
     void *event_data)
{
    switch (event_id)
    {
    case WIFI_EVENT_STA_START:
        printf("WiFi connecting ... \n");
        break;
    case WIFI_EVENT_STA_CONNECTED:
        printf("WiFi connected ... \n");
        break;
    case WIFI_EVENT_STA_DISCONNECTED:
        printf("WiFi lost connection ... \n");
        esp_wifi_connect();
        break;
    case IP_EVENT_STA_GOT_IP:
        printf("WiFi got IP...\n\n");
        break;
    default:
        break;
    }
}

void wifi_connection()
{
    // 1 - Wi-Fi/LwIP Init Phase
    esp_netif_init();                    // TCP/IP initiation
    esp_netif_create_default_wifi_sta(); // WiFi station
    wifi_init_config_t wifi_initiation = WIFI_INIT_CONFIG_DEFAULT();
    esp_wifi_init(&wifi_initiation); // s1.4
    esp_event_handler_register(WIFI_EVENT, ESP_EVENT_ANY_ID, wifi_event_handler, NULL);
    esp_event_handler_register(IP_EVENT, IP_EVENT_STA_GOT_IP, wifi_event_handler, NULL);
    wifi_config_t wifi_configuration = {
        .sta = {
                      .ssid = "*INSERT SSID",
                      .password = "*INSERT PASSWORD"}};
    esp_wifi_set_mode(WIFI_MODE_STA);


    esp_wifi_set_config(WIFI_IF_STA, &wifi_configuration);
    // 3 - Wi-Fi Start Phase
    printf("WiFi Starting...\n");
    esp_wifi_start();
    // 4- Wi-Fi Connect Phase
    esp_wifi_connect();
}

void mqtt_publish(esp_mqtt_client_handle_t client, char *mqtt_msg){
     const char *Topic_Name = "UPCARE/UNDERGRAD/ECE199_PUB2324";
     esp_mqtt_client_publish(client, Topic_Name, mqtt_msg, 0, 2, 0);
}

// void mqtt_publish_node_specific(esp_mqtt_client_handle_t client, int node_num){
//   const char *Topic_Name = "UPCARE/UNDERGRAD/ECE199_PUB2324";
//   if(node_num == 1){
//           esp_mqtt_client_publish(client, Topic_Name, json_payload_1, 0, 2, 0);
//   }
//   else if (node_num == 3){
//           esp_mqtt_client_publish(client, Topic_Name, json_payload_3, 0, 2, 0);
//   }
//   else if (node_num == 4){
//           esp_mqtt_client_publish(client, Topic_Name, json_payload_4, 0, 2, 0);
//   }
// }

static void mqtt_event_handler(void *handler_args, esp_event_base_t base, int32_t event_id, void
     *event_data)
{
    ESP_LOGD(TAG_MQTT, "Event dispatched from event loop base=%s, event_id=%" PRIi32, base, event_id);
    esp_mqtt_event_handle_t event = event_data;
    esp_mqtt_client_handle_t client = event->client;
    int msg_id;
    switch ((esp_mqtt_event_id_t)event_id) {
    case MQTT_EVENT_CONNECTED:
        ESP_LOGI(TAG_MQTT, "MQTT_EVENT_CONNECTED");
        msg_id = esp_mqtt_client_subscribe(client, "/topic/qos0", 0);
        ESP_LOGI(TAG_MQTT, "sent subscribe successful, msg_id=%d", msg_id);

        break;
    case MQTT_EVENT_DISCONNECTED:
        ESP_LOGI(TAG_MQTT, "MQTT_EVENT_DISCONNECTED");
        break;
    case MQTT_EVENT_SUBSCRIBED:
     ESP_LOGI(TAG_MQTT, "MQTT_EVENT_SUBSCRIBED, msg_id=%d", event->msg_id);
        break;
    case MQTT_EVENT_UNSUBSCRIBED:
        ESP_LOGI(TAG_MQTT, "MQTT_EVENT_UNSUBSCRIBED, msg_id=%d", event->msg_id);
        break;
    case MQTT_EVENT_PUBLISHED:
        ESP_LOGI(TAG_MQTT, "MQTT_EVENT_PUBLISHED, msg_id=%d", event->msg_id);
        break;
    case MQTT_EVENT_DATA:
        ESP_LOGI(TAG_MQTT, "MQTT_EVENT_DATA");
        break;
    case MQTT_EVENT_ERROR:
        ESP_LOGI(TAG_MQTT, "MQTT_EVENT_ERROR");
        if (event->error_handle->error_type == MQTT_ERROR_TYPE_TCP_TRANSPORT) {
            ESP_LOGI(TAG_MQTT, "Last error code reported from esp-tls: 0x%x",
    event->error_handle->esp_tls_last_esp_err);
            ESP_LOGI(TAG_MQTT, "Last tls stack error number: 0x%x",
    event->error_handle->esp_tls_stack_err);
            ESP_LOGI(TAG_MQTT, "Last captured errno : %d (%s)",
    event->error_handle->esp_transport_sock_errno,
                     strerror(event->error_handle->esp_transport_sock_errno));
        } else if (event->error_handle->error_type == MQTT_ERROR_TYPE_CONNECTION_REFUSED) {
            ESP_LOGI(TAG_MQTT, "Connection refused error: 0x%x",
    event->error_handle->connect_return_code);
        } else {
            ESP_LOGW(TAG_MQTT, "Unknown error type: 0x%x", event->error_handle->error_type);
        }
        break;
    default:
        ESP_LOGI(TAG_MQTT, "Other event id:%d", event->event_id);
        break;
    }
}

static void mqtt_app_start(esp_mqtt_client_handle_t client)
{
    esp_mqtt_client_register_event(client, ESP_EVENT_ANY_ID, mqtt_event_handler, NULL);
    esp_mqtt_client_start(client);
}

void upload_data (void *params)
{
    while(1){
        if (strcmp(payload_1, "") != 0) {
            // printf("payload 1\n");
            mqtt_publish(MQTtest, payload_1);
            strcpy(payload_1, "");
        }
        else if (strcmp(payload_2, "") != 0) {
            // printf("payload 2\n");
            mqtt_publish(MQTtest, payload_2);
            strcpy(payload_2, "");
        }
        else if (strcmp(payload_3, "") != 0) {
            // printf("payload 3\n");
            mqtt_publish(MQTtest, payload_3);
            strcpy(payload_3, "");
        }
        else if (strcmp(payload_4, "") != 0) {
            // printf("payload 4\n");
            mqtt_publish(MQTtest, payload_4);
            strcpy(payload_4, "");
        }
        vTaskDelay(1000/portTICK_PERIOD_MS);
    }
}

static struct example_info_store {
    uint16_t server_addr;   /* Vendor server unicast address */
    uint16_t vnd_tid;       /* TID contained in the vendor message */
} store = {
    .server_addr = ESP_BLE_MESH_ADDR_UNASSIGNED,
    .vnd_tid = 0,
};

static nvs_handle_t NVS_HANDLE;
static const char * NVS_KEY = "vendor_client";

static struct esp_ble_mesh_key {
    uint16_t net_idx;
    uint16_t app_idx;
    uint8_t  app_key[ESP_BLE_MESH_OCTET16_LEN];
} prov_key;

static esp_ble_mesh_cfg_srv_t config_server = {
    /* 3 transmissions with 20ms interval */
    .net_transmit = ESP_BLE_MESH_TRANSMIT(2, 20),
    .relay = ESP_BLE_MESH_RELAY_DISABLED,
    .relay_retransmit = ESP_BLE_MESH_TRANSMIT(2, 20),
    .beacon = ESP_BLE_MESH_BEACON_DISABLED,
#if defined(CONFIG_BLE_MESH_FRIEND)
    .friend_state = ESP_BLE_MESH_FRIEND_ENABLED,
#else
    .friend_state = ESP_BLE_MESH_FRIEND_NOT_SUPPORTED,
#endif
    .default_ttl = 7,
};

static esp_ble_mesh_client_t config_client;
static const esp_ble_mesh_client_op_pair_t vnd_op_pair[] = {
    { ESP_BLE_MESH_VND_MODEL_OP_SEND, ESP_BLE_MESH_VND_MODEL_OP_STATUS },
};

static esp_ble_mesh_client_t vendor_client = {
    .op_pair_size = ARRAY_SIZE(vnd_op_pair),
    .op_pair = vnd_op_pair,
};

static esp_ble_mesh_model_op_t vnd_op[] = {
    ESP_BLE_MESH_MODEL_OP(ESP_BLE_MESH_VND_MODEL_OP_STATUS, 2),
    ESP_BLE_MESH_MODEL_OP_END,
};

static esp_ble_mesh_model_t root_models[] = {
    ESP_BLE_MESH_MODEL_CFG_SRV(&config_server),
    ESP_BLE_MESH_MODEL_CFG_CLI(&config_client),
};

static esp_ble_mesh_model_t vnd_models[] = {
    ESP_BLE_MESH_VENDOR_MODEL(CID_ESP, ESP_BLE_MESH_VND_MODEL_ID_CLIENT,
    vnd_op, NULL, &vendor_client),
};

static esp_ble_mesh_elem_t elements[] = {
    ESP_BLE_MESH_ELEMENT(0, root_models, vnd_models),
};

static esp_ble_mesh_comp_t composition = {
    .cid = CID_ESP,
    .element_count = ARRAY_SIZE(elements),
    .elements = elements,
};

static esp_ble_mesh_prov_t provision = {
    .prov_uuid          = dev_uuid,
    .prov_unicast_addr  = PROV_OWN_ADDR,
    .prov_start_address = 0x0005,
};

static void mesh_example_info_store(void)
{
    ble_mesh_nvs_store(NVS_HANDLE, NVS_KEY, &store, sizeof(store));
}

static void mesh_example_info_restore(void)
{
    esp_err_t err = ESP_OK;
    bool exist = false;

    err = ble_mesh_nvs_restore(NVS_HANDLE, NVS_KEY, &store, sizeof(store), &exist);
    if (err != ESP_OK) {
        return;
    }

    if (exist) {
        ESP_LOGI(TAG, "Restore, server_addr 0x%04x, vnd_tid 0x%04x", store.server_addr, store.vnd_tid);
    }
}

static void example_ble_mesh_set_msg_common(esp_ble_mesh_client_common_param_t *common,
                                            esp_ble_mesh_node_t *node,
                                            esp_ble_mesh_model_t *model, uint32_t opcode)
{
    common->opcode = opcode;
    common->model = model;
    common->ctx.net_idx = prov_key.net_idx;
    common->ctx.app_idx = prov_key.app_idx;
    common->ctx.addr = node->unicast_addr;
    common->ctx.send_ttl = MSG_SEND_TTL;
    common->msg_timeout = MSG_TIMEOUT;
#if ESP_IDF_VERSION < ESP_IDF_VERSION_VAL(5, 2, 0)
    common->msg_role = MSG_ROLE;
#endif
}

static esp_err_t prov_complete(uint16_t node_index, const esp_ble_mesh_octet16_t uuid,
                               uint16_t primary_addr, uint8_t element_num, uint16_t net_idx)
{
    esp_ble_mesh_client_common_param_t common = {0};
    esp_ble_mesh_cfg_client_get_state_t get = {0};
    esp_ble_mesh_node_t *node = NULL;
    char name[10] = {'\0'};
    esp_err_t err;

    ESP_LOGI(TAG, "node_index %u, primary_addr 0x%04x, element_num %u, net_idx 0x%03x",
        node_index, primary_addr, element_num, net_idx);
    ESP_LOG_BUFFER_HEX("uuid", uuid, ESP_BLE_MESH_OCTET16_LEN);
    store.server_addr = primary_addr;
    mesh_example_info_store(); /* Store proper mesh example info */

    sprintf(name, "%s%02x", "NODE-", node_index);
    err = esp_ble_mesh_provisioner_set_node_name(node_index, name);
    if (err != ESP_OK) {
        ESP_LOGE(TAG, "Failed to set node name");
        return ESP_FAIL;
    }

    node = esp_ble_mesh_provisioner_get_node_with_addr(primary_addr);
    if (node == NULL) {
        ESP_LOGE(TAG, "Failed to get node 0x%04x info", primary_addr);
        return ESP_FAIL;
    }

    example_ble_mesh_set_msg_common(&common, node, config_client.model,
     ESP_BLE_MESH_MODEL_OP_COMPOSITION_DATA_GET);
    get.comp_data_get.page = COMP_DATA_PAGE_0;
    err = esp_ble_mesh_config_client_get_state(&common, &get);
    if (err != ESP_OK) {
        ESP_LOGE(TAG, "Failed to send Config Composition Data Get");
        return ESP_FAIL;
    }

    return ESP_OK;
}

static void recv_unprov_adv_pkt(uint8_t dev_uuid[ESP_BLE_MESH_OCTET16_LEN], uint8_t addr[BD_ADDR_LEN],
                                esp_ble_mesh_addr_type_t addr_type, uint16_t oob_info,
                                uint8_t adv_type, esp_ble_mesh_prov_bearer_t bearer)
{
    esp_ble_mesh_unprov_dev_add_t add_dev = {0};
    esp_err_t err;

    /* Due to the API esp_ble_mesh_provisioner_set_dev_uuid_match, Provisioner will only
     * use this callback to report the devices, whose device UUID starts with 0xdd & 0xdd,
     * to the application layer.
     */

    ESP_LOG_BUFFER_HEX("Device address", addr, BD_ADDR_LEN);
    ESP_LOGI(TAG, "Address type 0x%02x, adv type 0x%02x", addr_type, adv_type);
    ESP_LOG_BUFFER_HEX("Device UUID", dev_uuid, ESP_BLE_MESH_OCTET16_LEN);
    ESP_LOGI(TAG, "oob info 0x%04x, bearer %s", oob_info, (bearer & ESP_BLE_MESH_PROV_ADV) ? "PB-ADV" :
     "PB-GATT");

    memcpy(add_dev.addr, addr, BD_ADDR_LEN);
    add_dev.addr_type = (esp_ble_mesh_addr_type_t)addr_type;
    memcpy(add_dev.uuid, dev_uuid, ESP_BLE_MESH_OCTET16_LEN);
    add_dev.oob_info = oob_info;
    add_dev.bearer = (esp_ble_mesh_prov_bearer_t)bearer;
    /* Note: If unprovisioned device adv packets have not been received, we should not add
             device with ADD_DEV_START_PROV_NOW_FLAG set. */
    err = esp_ble_mesh_provisioner_add_unprov_dev(&add_dev,
            ADD_DEV_RM_AFTER_PROV_FLAG | ADD_DEV_START_PROV_NOW_FLAG | ADD_DEV_FLUSHABLE_DEV_FLAG);
    if (err != ESP_OK) {
        ESP_LOGE(TAG, "Failed to start provisioning device");
    }
}

static void example_ble_mesh_provisioning_cb(esp_ble_mesh_prov_cb_event_t event,
                                             esp_ble_mesh_prov_cb_param_t *param)
{
    switch (event) {
    case ESP_BLE_MESH_PROV_REGISTER_COMP_EVT:
        ESP_LOGI(TAG, "ESP_BLE_MESH_PROV_REGISTER_COMP_EVT, err_code %d",
     param->prov_register_comp.err_code);
        mesh_example_info_restore(); /* Restore proper mesh example info */
        break;
    case ESP_BLE_MESH_PROVISIONER_PROV_ENABLE_COMP_EVT:
        ESP_LOGI(TAG, "ESP_BLE_MESH_PROVISIONER_PROV_ENABLE_COMP_EVT, err_code %d",
     param->provisioner_prov_enable_comp.err_code);
        break;
    case ESP_BLE_MESH_PROVISIONER_PROV_DISABLE_COMP_EVT:
        ESP_LOGI(TAG, "ESP_BLE_MESH_PROVISIONER_PROV_DISABLE_COMP_EVT, err_code %d",
     param->provisioner_prov_disable_comp.err_code);
        break;
    case ESP_BLE_MESH_PROVISIONER_RECV_UNPROV_ADV_PKT_EVT:
        ESP_LOGI(TAG, "ESP_BLE_MESH_PROVISIONER_RECV_UNPROV_ADV_PKT_EVT");
        recv_unprov_adv_pkt(param->provisioner_recv_unprov_adv_pkt.dev_uuid,
     param->provisioner_recv_unprov_adv_pkt.addr,
                            param->provisioner_recv_unprov_adv_pkt.addr_type,
     param->provisioner_recv_unprov_adv_pkt.oob_info,
                            param->provisioner_recv_unprov_adv_pkt.adv_type,
     param->provisioner_recv_unprov_adv_pkt.bearer);
        break;
    case ESP_BLE_MESH_PROVISIONER_PROV_LINK_OPEN_EVT:
        ESP_LOGI(TAG, "ESP_BLE_MESH_PROVISIONER_PROV_LINK_OPEN_EVT, bearer %s",
            param->provisioner_prov_link_open.bearer == ESP_BLE_MESH_PROV_ADV ? "PB-ADV" : "PB-GATT");
        break;
    case ESP_BLE_MESH_PROVISIONER_PROV_LINK_CLOSE_EVT:
        ESP_LOGI(TAG, "ESP_BLE_MESH_PROVISIONER_PROV_LINK_CLOSE_EVT, bearer %s, reason 0x%02x",
            param->provisioner_prov_link_close.bearer == ESP_BLE_MESH_PROV_ADV ? "PB-ADV" : "PB-GATT",
    param->provisioner_prov_link_close.reason);
        break;
    case ESP_BLE_MESH_PROVISIONER_PROV_COMPLETE_EVT:
        prov_complete(param->provisioner_prov_complete.node_idx,
    param->provisioner_prov_complete.device_uuid,
                      param->provisioner_prov_complete.unicast_addr,
    param->provisioner_prov_complete.element_num,
                      param->provisioner_prov_complete.netkey_idx);
        break;
    case ESP_BLE_MESH_PROVISIONER_ADD_UNPROV_DEV_COMP_EVT:
        ESP_LOGI(TAG, "ESP_BLE_MESH_PROVISIONER_ADD_UNPROV_DEV_COMP_EVT, err_code %d",
    param->provisioner_add_unprov_dev_comp.err_code);
        break;
    case ESP_BLE_MESH_PROVISIONER_SET_DEV_UUID_MATCH_COMP_EVT:
        ESP_LOGI(TAG, "ESP_BLE_MESH_PROVISIONER_SET_DEV_UUID_MATCH_COMP_EVT, err_code %d",
    param->provisioner_set_dev_uuid_match_comp.err_code);
        break;
    case ESP_BLE_MESH_PROVISIONER_SET_NODE_NAME_COMP_EVT:
        ESP_LOGI(TAG, "ESP_BLE_MESH_PROVISIONER_SET_NODE_NAME_COMP_EVT, err_code %d",
    param->provisioner_set_node_name_comp.err_code);
        if (param->provisioner_set_node_name_comp.err_code == 0) {
            const char *name =
    esp_ble_mesh_provisioner_get_node_name(param->provisioner_set_node_name_comp.node_index);
            if (name) {
                ESP_LOGI(TAG, "Node %d name %s", param->provisioner_set_node_name_comp.node_index, name);
            }
        }
        break;
    case ESP_BLE_MESH_PROVISIONER_ADD_LOCAL_APP_KEY_COMP_EVT:
        ESP_LOGI(TAG, "ESP_BLE_MESH_PROVISIONER_ADD_LOCAL_APP_KEY_COMP_EVT, err_code %d",
    param->provisioner_add_app_key_comp.err_code);
        if (param->provisioner_add_app_key_comp.err_code == 0) {
            prov_key.app_idx = param->provisioner_add_app_key_comp.app_idx;
            esp_err_t err = esp_ble_mesh_provisioner_bind_app_key_to_local_model(PROV_OWN_ADDR,
    prov_key.app_idx,
                    ESP_BLE_MESH_VND_MODEL_ID_CLIENT, CID_ESP);
            if (err != ESP_OK) {
                ESP_LOGE(TAG, "Failed to bind AppKey to vendor client");
            }
        }
        break;
    case ESP_BLE_MESH_PROVISIONER_BIND_APP_KEY_TO_MODEL_COMP_EVT:
        ESP_LOGI(TAG, "ESP_BLE_MESH_PROVISIONER_BIND_APP_KEY_TO_MODEL_COMP_EVT, err_code %d",
    param->provisioner_bind_app_key_to_model_comp.err_code);
        break;
    case ESP_BLE_MESH_PROVISIONER_STORE_NODE_COMP_DATA_COMP_EVT:
        ESP_LOGI(TAG, "ESP_BLE_MESH_PROVISIONER_STORE_NODE_COMP_DATA_COMP_EVT, err_code %d",
    param->provisioner_store_node_comp_data_comp.err_code);
        break;
    default:
        break;
    }
}

static void example_ble_mesh_parse_node_comp_data(const uint8_t *data, uint16_t length)
{
    uint16_t cid, pid, vid, crpl, feat;
    uint16_t loc, model_id, company_id;
    uint8_t nums, numv;
    uint16_t offset;
    int i;

    cid = COMP_DATA_2_OCTET(data, 0);
    pid = COMP_DATA_2_OCTET(data, 2);
    vid = COMP_DATA_2_OCTET(data, 4);
    crpl = COMP_DATA_2_OCTET(data, 6);
    feat = COMP_DATA_2_OCTET(data, 8);
    offset = 10;

    ESP_LOGI(TAG, "********************** Composition Data Start **********************");
    ESP_LOGI(TAG, "* CID 0x%04x, PID 0x%04x, VID 0x%04x, CRPL 0x%04x, Features 0x%04x *", cid, pid, vid,
    crpl, feat);
    for (; offset < length; ) {
        loc = COMP_DATA_2_OCTET(data, offset);
        nums = COMP_DATA_1_OCTET(data, offset + 2);
        numv = COMP_DATA_1_OCTET(data, offset + 3);
        offset += 4;
        ESP_LOGI(TAG, "* Loc 0x%04x, NumS 0x%02x, NumV 0x%02x *", loc, nums, numv);
        for (i = 0; i < nums; i++) {
            model_id = COMP_DATA_2_OCTET(data, offset);
            ESP_LOGI(TAG, "* SIG Model ID 0x%04x *", model_id);
            offset += 2;
        }
        for (i = 0; i < numv; i++) {
            company_id = COMP_DATA_2_OCTET(data, offset);
            model_id = COMP_DATA_2_OCTET(data, offset + 2);
            ESP_LOGI(TAG, "* Vendor Model ID 0x%04x, Company ID 0x%04x *", model_id, company_id);
            offset += 4;
        }
    }
    ESP_LOGI(TAG, "********************** Composition Data End **********************");
}

static void example_ble_mesh_config_client_cb(esp_ble_mesh_cfg_client_cb_event_t event,
                                              esp_ble_mesh_cfg_client_cb_param_t *param)
{
    esp_ble_mesh_client_common_param_t common = {0};
    esp_ble_mesh_cfg_client_set_state_t set = {0};
    esp_ble_mesh_node_t *node = NULL;
    esp_err_t err;

    ESP_LOGI(TAG, "Config client, err_code %d, event %u, addr 0x%04x, opcode 0x%04" PRIx32,
        param->error_code, event, param->params->ctx.addr, param->params->opcode);

    if (param->error_code) {
        ESP_LOGE(TAG, "Send config client message failed, opcode 0x%04" PRIx32, param->params->opcode);
        return;
    }

    node = esp_ble_mesh_provisioner_get_node_with_addr(param->params->ctx.addr);
    if (!node) {
        ESP_LOGE(TAG, "Failed to get node 0x%04x info", param->params->ctx.addr);
        return;
    }

    switch (event) {
    case ESP_BLE_MESH_CFG_CLIENT_GET_STATE_EVT:
        if (param->params->opcode == ESP_BLE_MESH_MODEL_OP_COMPOSITION_DATA_GET) {
            ESP_LOG_BUFFER_HEX("Composition data",
    param->status_cb.comp_data_status.composition_data->data,
                param->status_cb.comp_data_status.composition_data->len);

    example_ble_mesh_parse_node_comp_data(param->status_cb.comp_data_status.composition_data->data,
                param->status_cb.comp_data_status.composition_data->len);
            err = esp_ble_mesh_provisioner_store_node_comp_data(param->params->ctx.addr,
                param->status_cb.comp_data_status.composition_data->data,
                param->status_cb.comp_data_status.composition_data->len);
            if (err != ESP_OK) {
                ESP_LOGE(TAG, "Failed to store node composition data");
                break;
            }

            example_ble_mesh_set_msg_common(&common, node, config_client.model,
    ESP_BLE_MESH_MODEL_OP_APP_KEY_ADD);
            set.app_key_add.net_idx = prov_key.net_idx;
            set.app_key_add.app_idx = prov_key.app_idx;
            memcpy(set.app_key_add.app_key, prov_key.app_key, ESP_BLE_MESH_OCTET16_LEN);
            err = esp_ble_mesh_config_client_set_state(&common, &set);
            if (err != ESP_OK) {
                ESP_LOGE(TAG, "Failed to send Config AppKey Add");
            }
        }
        break;
    case ESP_BLE_MESH_CFG_CLIENT_SET_STATE_EVT:
        if (param->params->opcode == ESP_BLE_MESH_MODEL_OP_APP_KEY_ADD) {
            example_ble_mesh_set_msg_common(&common, node, config_client.model,
    ESP_BLE_MESH_MODEL_OP_MODEL_APP_BIND);
            set.model_app_bind.element_addr = node->unicast_addr;
            set.model_app_bind.model_app_idx = prov_key.app_idx;
            set.model_app_bind.model_id = ESP_BLE_MESH_VND_MODEL_ID_SERVER;
            set.model_app_bind.company_id = CID_ESP;
            err = esp_ble_mesh_config_client_set_state(&common, &set);
            if (err != ESP_OK) {
                ESP_LOGE(TAG, "Failed to send Config Model App Bind");
            }
        } else if (param->params->opcode == ESP_BLE_MESH_MODEL_OP_MODEL_APP_BIND) {
            ESP_LOGW(TAG, "%s, Provision and config successfully", __func__);

        }
        break;
    case ESP_BLE_MESH_CFG_CLIENT_PUBLISH_EVT:
        if (param->params->opcode == ESP_BLE_MESH_MODEL_OP_COMPOSITION_DATA_STATUS) {
            ESP_LOG_BUFFER_HEX("Composition data",
    param->status_cb.comp_data_status.composition_data->data,
                param->status_cb.comp_data_status.composition_data->len);
        }
        break;
    case ESP_BLE_MESH_CFG_CLIENT_TIMEOUT_EVT:
        switch (param->params->opcode) {
        case ESP_BLE_MESH_MODEL_OP_COMPOSITION_DATA_GET: {
            esp_ble_mesh_cfg_client_get_state_t get = {0};
            example_ble_mesh_set_msg_common(&common, node, config_client.model,
    ESP_BLE_MESH_MODEL_OP_COMPOSITION_DATA_GET);
            get.comp_data_get.page = COMP_DATA_PAGE_0;
            err = esp_ble_mesh_config_client_get_state(&common, &get);
            if (err != ESP_OK) {
                ESP_LOGE(TAG, "Failed to send Config Composition Data Get");
            }
            break;
        }
        case ESP_BLE_MESH_MODEL_OP_APP_KEY_ADD:
            example_ble_mesh_set_msg_common(&common, node, config_client.model,
    ESP_BLE_MESH_MODEL_OP_APP_KEY_ADD);
            set.app_key_add.net_idx = prov_key.net_idx;
            set.app_key_add.app_idx = prov_key.app_idx;
            memcpy(set.app_key_add.app_key, prov_key.app_key, ESP_BLE_MESH_OCTET16_LEN);
            err = esp_ble_mesh_config_client_set_state(&common, &set);
            if (err != ESP_OK) {
                ESP_LOGE(TAG, "Failed to send Config AppKey Add");
            }
            break;
        case ESP_BLE_MESH_MODEL_OP_MODEL_APP_BIND:
            example_ble_mesh_set_msg_common(&common, node, config_client.model,
    ESP_BLE_MESH_MODEL_OP_MODEL_APP_BIND);
            set.model_app_bind.element_addr = node->unicast_addr;
            set.model_app_bind.model_app_idx = prov_key.app_idx;
            set.model_app_bind.model_id = ESP_BLE_MESH_VND_MODEL_ID_SERVER;
            set.model_app_bind.company_id = CID_ESP;
            err = esp_ble_mesh_config_client_set_state(&common, &set);
            if (err != ESP_OK) {
                ESP_LOGE(TAG, "Failed to send Config Model App Bind");
            }
            break;
        default:
            break;
        }
        break;
    default:
        ESP_LOGE(TAG, "Invalid config client event %u", event);
        break;
    }
}

void example_ble_mesh_send_vendor_message(bool resend)
{
    esp_ble_mesh_msg_ctx_t ctx = {0};
    uint32_t opcode;
    esp_err_t err;

    ctx.net_idx = prov_key.net_idx;
    ctx.app_idx = prov_key.app_idx;
    ctx.addr = 0xffff;
    ctx.send_ttl = MSG_SEND_TTL;
    opcode = ESP_BLE_MESH_VND_MODEL_OP_SEND;

    if (resend == false) {
        store.vnd_tid++;
    }

    char mydata[20];
    Get_current_date_time(Current_Date_Time);
    snprintf(mydata, 20, "%s", Current_Date_Time);

    err = esp_ble_mesh_client_model_send_msg(vendor_client.model, &ctx, opcode,
            strlen(mydata)+1, (uint8_t *)mydata, MSG_TIMEOUT, true, MSG_ROLE);
    if (err != ESP_OK) {
        ESP_LOGE(TAG, "Failed to send vendor message 0x%06" PRIx32, opcode);
        return;
    }

    mesh_example_info_store(); /* Store proper mesh example info */
}

void send_time(void *params)
{
    while(1){
        example_ble_mesh_send_vendor_message(true);
        vTaskDelay(10000 / portTICK_PERIOD_MS);
    }
}

static void example_ble_mesh_custom_model_cb(esp_ble_mesh_model_cb_event_t event,
                                             esp_ble_mesh_model_cb_param_t *param)
{
    static int64_t start_time;

    switch (event) {
    case ESP_BLE_MESH_MODEL_OPERATION_EVT:
        if (param->model_operation.opcode == ESP_BLE_MESH_VND_MODEL_OP_SEND) {
            int64_t end_time = esp_timer_get_time();
            //ESP_LOGI(TAG, "Recv 0x06%" PRIx32 ", tid 0x%04x, time %lldus",
                    //param->model_operation.opcode, store.vnd_tid, end_time - start_time);
            ESP_LOGI(TAG, "%s", (char *)param->model_operation.msg);
        }
        if (param->model_operation.opcode == ESP_BLE_MESH_VND_MODEL_OP_STATUS) {
            int64_t end_time = esp_timer_get_time();
            uint8_t *d = param->model_operation.msg;
            char *levalue = (char *)d;
            ESP_LOGI(TAG, "Recv 0x06%" PRIx32 ", tid 0x%04x, time %lldus",
                     param->model_operation.opcode, store.vnd_tid, end_time - start_time);
            printf("string received: %s\n", levalue);
        }
        break;
    case ESP_BLE_MESH_MODEL_SEND_COMP_EVT:
        if (param->model_send_comp.err_code) {
            ESP_LOGE(TAG, "Failed to send message 0x%06" PRIx32, param->model_send_comp.opcode);
            break;
        }
        start_time = esp_timer_get_time();
        ESP_LOGI(TAG, "Send 0x%06" PRIx32, param->model_send_comp.opcode);
        break;
    case ESP_BLE_MESH_CLIENT_MODEL_RECV_PUBLISH_MSG_EVT:
        ESP_LOGI(TAG, "Receive publish message 0x%06" PRIx32, param->client_recv_publish_msg.opcode);
        // ESP_LOGI(TAG, "%s", (char *)param->client_recv_publish_msg.msg);
        // printf("%s\n", (char *)param->client_recv_publish_msg.msg);
        char * buf = (char *)param->client_recv_publish_msg.msg;
        if (strcmp(payload_1, "") == 0){
        snprintf(payload_1, 500, "%s", buf);
        // printf("payload_1\n");
        } else if ( strcmp(payload_2, "") == 0){
            snprintf(payload_2, 500, "%s", buf);
            // printf("payload_2\n");
        } else if ( strcmp(payload_3, "") == 0){
            snprintf(payload_3, 500, "%s", buf);
            // printf("payload_3\n");
        } else{
            snprintf(payload_4, 500, "%s", buf);
            // printf("payload_4\n");
        }
        break;
    case ESP_BLE_MESH_CLIENT_MODEL_SEND_TIMEOUT_EVT:
        ESP_LOGW(TAG, "Client message 0x%06" PRIx32 " timeout", param->client_send_timeout.opcode);
        example_ble_mesh_send_vendor_message(true);
        break;
    default:
        break;
    }
}

static esp_err_t ble_mesh_init(void)
{
    uint8_t match[2] = { 0x32, 0x10 };
    esp_err_t err;

    prov_key.net_idx = ESP_BLE_MESH_KEY_PRIMARY;
    prov_key.app_idx = APP_KEY_IDX;
    memset(prov_key.app_key, APP_KEY_OCTET, sizeof(prov_key.app_key));

    esp_ble_mesh_register_prov_callback(example_ble_mesh_provisioning_cb);
    esp_ble_mesh_register_config_client_callback(example_ble_mesh_config_client_cb);
    esp_ble_mesh_register_custom_model_callback(example_ble_mesh_custom_model_cb);

    err = esp_ble_mesh_init(&provision, &composition);
    if (err != ESP_OK) {
        ESP_LOGE(TAG, "Failed to initialize mesh stack");
        return err;
    }

    err = esp_ble_mesh_client_model_init(&vnd_models[0]);
    if (err) {
        ESP_LOGE(TAG, "Failed to initialize vendor client");
        return err;
    }

    err = esp_ble_mesh_provisioner_set_dev_uuid_match(match, sizeof(match), 0x0, false);
    if (err != ESP_OK) {
        ESP_LOGE(TAG, "Failed to set matching device uuid");
        return err;
    }

    err = esp_ble_mesh_provisioner_prov_enable((esp_ble_mesh_prov_bearer_t)(ESP_BLE_MESH_PROV_ADV |
    ESP_BLE_MESH_PROV_GATT));
    if (err != ESP_OK) {
        ESP_LOGE(TAG, "Failed to enable mesh provisioner");
        return err;
    }
    err = esp_ble_mesh_provisioner_add_local_app_key(prov_key.app_key, prov_key.net_idx,
    prov_key.app_idx);
    if (err != ESP_OK) {
        ESP_LOGE(TAG, "Failed to add local AppKey");
        return err;
    }

    ESP_LOGI(TAG, "ESP BLE Mesh Provisioner initialized");

    return ESP_OK;
}

// static int64_t start;

void app_main(void)
{
    esp_err_t err;



    ESP_LOGI(TAG, "Initializing...");

    err = nvs_flash_init();
    if (err == ESP_ERR_NVS_NO_FREE_PAGES) {
        ESP_ERROR_CHECK(nvs_flash_erase());
        err = nvs_flash_init();
    }
    ESP_ERROR_CHECK(esp_netif_init());
    ESP_ERROR_CHECK(esp_event_loop_create_default());
    ESP_ERROR_CHECK(err);
    wifi_connection();
    vTaskDelay(5*1000/portTICK_PERIOD_MS);
    Set_SystemTime_SNTP();

    const esp_mqtt_client_config_t mqtt_cfg = {
    //broker URL or URI
        .broker = {
            .address.uri = "*INSERT URI",
            .verification.certificate = (const char *)mqtt_eclipseprojects_io_pem_start
        },
        //this is where I put the credentials
        //I added this
        .credentials = {
            .username="*INSERT USERNAME",
            .authentication = {
                .password="*INSERT PASSWORD"
            }
        },
    };
    MQTtest = esp_mqtt_client_init(&mqtt_cfg);
    mqtt_app_start(MQTtest);



    err = bluetooth_init();
    if (err != ESP_OK) {
        ESP_LOGE(TAG, "esp32_bluetooth_init failed (err %d)", err);
        return;
    }

    /* Open nvs namespace for storing/restoring mesh example info */
    err = ble_mesh_nvs_open(&NVS_HANDLE);
    if (err) {
        return;
    }

    ble_mesh_get_dev_uuid(dev_uuid);

    /* Initialize the Bluetooth Mesh Subsystem */
    err = ble_mesh_init();
    if (err != ESP_OK) {
        ESP_LOGE(TAG, "Bluetooth mesh init failed (err %d)", err);
    }

    xTaskCreate(send_time, "ble_send", 4096, NULL, 5, NULL);
    xTaskCreate(upload_data, "mqtt_upload", 4096, NULL, 5, NULL);
}
```

Appendix L
Code for MQTT on Espressif, including WiFi Functionality and Time Server Clock Synchronization

Tutorial for MQTT and WiFi Functionality:
https://www.youtube.com/watch?v=YrdnJWVK1ag
Source Reference for MQTT and WiFi Functionality:
https://github.com/SIMS-IOT-Devices/FreeRTOS-ESP-IDF-MQTT/blob/main/mqtt_tcp_pub_sub.c
Official Documentation for MQTT Functionality:
https://docs.espressif.com/projects/esp-idf/en/stable/esp32/api-reference/protocols/mqtt.html
Tutorial for using HiveMQ Data Broker:
https://docs.hivemq.com/hivemq/latest/user-guide/getting-started.html
Tutorial for Time Synchronization with NTP Servers:
https://www.youtube.com/watch?v=HR-Kcj5j_Uc
Source Reference for Time Synchronization with NTP Servers:
https://github.com/vinothkannan369/ESP32/blob/main/SNTP/set_clk.c

```
/* MQTT over SSL Example

   This example code is in the Public Domain (or CC0 licensed, at your option.)

   Unless required by applicable law or agreed to in writing, this
   software is distributed on an "AS IS" BASIS, WITHOUT WARRANTIES OR
   CONDITIONS OF ANY KIND, either express or implied.
*/

#include <stdio.h>
#include <stdint.h>
#include <stddef.h>
#include <string.h>
#include <sys/param.h>
#include <stdlib.h>
#include <time.h>

#include "esp_system.h"
#include "esp_partition.h"
#include "nvs_flash.h"
#include "esp_event.h"
#include "esp_netif.h"
#include "protocol_examples_common.h"
#include "esp_log.h"
#include "mqtt_client.h"
#include "esp_tls.h"
#include "esp_ota_ops.h"
#include "esp_wifi.h"

#include "freertos/FreeRTOS.h"
#include "freertos/task.h"
#include "freertos/semphr.h"
#include "freertos/queue.h"

#include "lwip/sockets.h"
#include "lwip/dns.h"
#include "lwip/netdb.h"


int getRandomInteger(int min, int max) {
    if (min >= max) {
        // Invalid range, return an error value or handle it appropriately
        return -1;
    }

    // Seed the random number generator with the current time
    srand(time(NULL));

    // Generate a random integer within the specified range
    int randomValueInRange = rand() % (max - min + 1) + min;

    return randomValueInRange;
}

static void wifi_event_handler(void *event_handler_arg, esp_event_base_t event_base, int32_t event_id,
     void *event_data)
{
    switch (event_id)
    {
    case WIFI_EVENT_STA_START:
        printf("WiFi connecting ... \n");
        break;
    case WIFI_EVENT_STA_CONNECTED:
        printf("WiFi connected ... \n");
        break;
    case WIFI_EVENT_STA_DISCONNECTED:
        printf("WiFi lost connection ... \n");
        esp_wifi_connect();
        break;
    case IP_EVENT_STA_GOT_IP:
        printf("WiFi got IP ... \n\n");
        break;
    default:
        break;
    }
}

void wifi_connection()
{
    // 1 - Wi-Fi/LwIP Init Phase
    esp_netif_init();                    // TCP/IP initiation                                          s1.1
    esp_event_loop_create_default();     // event loop                                      s1.2
    esp_netif_create_default_wifi_sta(); // WiFi station                        s1.3
    wifi_init_config_t wifi_initiation = WIFI_INIT_CONFIG_DEFAULT();
    esp_wifi_init(&wifi_initiation); //                                                      s1.4
    // 2 - Wi-Fi Configuration Phase
    esp_event_handler_register(WIFI_EVENT, ESP_EVENT_ANY_ID, wifi_event_handler, NULL);
    esp_event_handler_register(IP_EVENT, IP_EVENT_STA_GOT_IP, wifi_event_handler, NULL);
    wifi_config_t wifi_configuration = {
        .sta = {
        //!Readme Wifi Name and Password
             //.ssid = "INSERT WiFi SSID HERE",
             //.password = "INSERT WiFi PASSWORD HERE"}};
                      .ssid = "ESP32Test",
                      .password = "Sundowner"}};

    esp_wifi_set_config(ESP_IF_WIFI_STA, &wifi_configuration);
    // 3 - Wi-Fi Start Phase
    esp_wifi_start();
    // 4- Wi-Fi Connect Phase
    esp_wifi_connect();
}



//global variables
int test_count=0;
char payload[500]="This is a test"; //500 characters max
//esp_mqtt_client_handle_t client;

//edit this function to what data you want to upload
void custom_data(){
             //see: "payload" global variable

             //sample log: IAQ: %.0f, PM2.5: %.2f, PM10: %.2f, Temperature: %.02f, RH: %.2f, CO2: %.2f,
     VOC: %.2f, CO: %.2f, NO2: %.2f

             //generate random integers
             int AQI = getRandomInteger(0, 150);
             int PM25 = getRandomInteger(0, 1000);
             int PM10 = getRandomInteger(0, 1000);
             int Temp = getRandomInteger(25, 32);
             int Hum = getRandomInteger(50, 75);
             int CO2 = getRandomInteger(400, 1200);
             int VOC = getRandomInteger(0, 1000);
             int CO = getRandomInteger(0, 1000);
             int NO = getRandomInteger(0, 10);


             //payload is being concatenated
             //sprintf(payload, "This is upload number %d", test_count);

             sprintf(payload, "{ \"Log ID\": %d, \"AQI\":%d, \"PM2.5\": %d, \"PM10\": %d, \"Temperature\":
     %d, \"RH\": %d, \"CO2\": %d, \"VOC\": %d, \"CO\": %d, \"NO2\": %d}", test_count, AQI, PM25, PM10,
     Temp, Hum, CO2, VOC, CO, NO);
             //
}
```

```
//function for actually uploading
//edit the topic if you want to change
void custom_upload(esp_mqtt_client_handle_t client){

            //change this variable to change the topic name
            //const char *HiveMQ = "/test/helloworld";
            const char *HiveMQ = "/test/simulation/log_data";

            //for debugging
            printf("Upload attempt...\r\n");

            //first subscribe to the target topic
            //esp_mqtt_client_subscribe(client, HiveMQ, 2);

            //actual upload
            esp_mqtt_client_publish(client, HiveMQ, payload, 0, 2, 0);

            //unsubscribe to end connection until next upload attempt
            //esp_mqtt_client_unsubscribe(client, HiveMQ);

}

//!-------

static const char *TAG = "MQTTS_EXAMPLE";


#if CONFIG_BROKER_CERTIFICATE_OVERRIDDEN == 1
static const uint8_t mqtt_eclipseprojects_io_pem_start[]  = "-----BEGIN CERTIFICATE-----\n"
    CONFIG_BROKER_CERTIFICATE_OVERRIDE "\n-----END CERTIFICATE-----";
#else
extern const uint8_t mqtt_eclipseprojects_io_pem_start[]
    asm("_binary_mqtt_eclipseprojects_io_pem_start");
#endif
extern const uint8_t mqtt_eclipseprojects_io_pem_end[]   asm("_binary_mqtt_eclipseprojects_io_pem_end");

//
// Note: this function is for testing purposes only publishing part of the active partition
//           (to be checked against the original binary)
//
static void send_binary(esp_mqtt_client_handle_t client)
{
    esp_partition_mmap_handle_t out_handle;
    const void *binary_address;
    const esp_partition_t *partition = esp_ota_get_running_partition();
    esp_partition_mmap(partition, 0, partition->size, ESP_PARTITION_MMAP_DATA, &binary_address,
    &out_handle);
    // sending only the configured portion of the partition (if it's less than the partition size)
    int binary_size = MIN(CONFIG_BROKER_BIN_SIZE_TO_SEND, partition->size);
    int msg_id = esp_mqtt_client_publish(client, "/topic/binary", binary_address, binary_size, 0, 0);
    ESP_LOGI(TAG, "binary sent with msg_id=%d", msg_id);
}

/*
 * @brief Event handler registered to receive MQTT events
 *
 *  This function is called by the MQTT client event loop.
 *
 * @param handler_args user data registered to the event.
 * @param base Event base for the handler(always MQTT Base in this example).
 * @param event_id The id for the received event.
 * @param event_data The data for the event, esp_mqtt_event_handle_t.
 */
static void mqtt_event_handler(void *handler_args, esp_event_base_t base, int32_t event_id, void
    *event_data)
{
    ESP_LOGD(TAG, "Event dispatched from event loop base=%s, event_id=%" PRIi32, base, event_id);
    esp_mqtt_event_handle_t event = event_data;
    esp_mqtt_client_handle_t client = event->client;
    int msg_id;
    switch ((esp_mqtt_event_id_t)event_id) {
    case MQTT_EVENT_CONNECTED:
        ESP_LOGI(TAG, "MQTT_EVENT_CONNECTED");
        msg_id = esp_mqtt_client_subscribe(client, "/topic/qos0", 0);
        ESP_LOGI(TAG, "sent subscribe successful, msg_id=%d", msg_id);

        //msg_id = esp_mqtt_client_subscribe(client, "/topic/qos1", 1);
        //ESP_LOGI(TAG, "sent subscribe successful, msg_id=%d", msg_id);

        //msg_id = esp_mqtt_client_unsubscribe(client, "/topic/qos1");
        //ESP_LOGI(TAG, "sent unsubscribe successful, msg_id=%d", msg_id);
        break;
    case MQTT_EVENT_DISCONNECTED:
        ESP_LOGI(TAG, "MQTT_EVENT_DISCONNECTED");
        break;
    case MQTT_EVENT_SUBSCRIBED:
            ESP_LOGI(TAG, "MQTT_EVENT_SUBSCRIBED, msg_id=%d", event->msg_id);
        msg_id = esp_mqtt_client_publish(client, "/topic/qos0", "data", 0, 0, 0);
        //msg_id = esp_mqtt_client_publish(client, "/topic/qos0", payload, 0, 0, 0);
        ESP_LOGI(TAG, "sent publish successful, msg_id=%d", msg_id);
        break;
    case MQTT_EVENT_UNSUBSCRIBED:
        ESP_LOGI(TAG, "MQTT_EVENT_UNSUBSCRIBED, msg_id=%d", event->msg_id);
        break;
    case MQTT_EVENT_PUBLISHED:
        ESP_LOGI(TAG, "MQTT_EVENT_PUBLISHED, msg_id=%d", event->msg_id);
        break;
    case MQTT_EVENT_DATA:
        ESP_LOGI(TAG, "MQTT_EVENT_DATA");
        printf("TOPIC=%.*s\r\n", event->topic_len, event->topic);
        printf("DATA=%.*s\r\n", event->data_len, event->data);
        if (strncmp(event->data, "send binary please", event->data_len) == 0) {
            ESP_LOGI(TAG, "Sending the binary");
            send_binary(client);
        }
        break;
    case MQTT_EVENT_ERROR:
        ESP_LOGI(TAG, "MQTT_EVENT_ERROR");
        if (event->error_handle->error_type == MQTT_ERROR_TYPE_TCP_TRANSPORT) {
            ESP_LOGI(TAG, "Last error code reported from esp-tls: 0x%x",
     event->error_handle->esp_tls_last_esp_err);
            ESP_LOGI(TAG, "Last tls stack error number: 0x%x", event->error_handle->esp_tls_stack_err);
            ESP_LOGI(TAG, "Last captured errno : %d (%s)",
     event->error_handle->esp_transport_sock_errno,
                     strerror(event->error_handle->esp_transport_sock_errno));
        } else if (event->error_handle->error_type == MQTT_ERROR_TYPE_CONNECTION_REFUSED) {
            ESP_LOGI(TAG, "Connection refused error: 0x%x", event->error_handle->connect_return_code);
        } else {
            ESP_LOGW(TAG, "Unknown error type: 0x%x", event->error_handle->error_type);
        }
        break;
    default:
        ESP_LOGI(TAG, "Other event id:%d", event->event_id);
        break;
    }
}


static void mqtt_app_start(esp_mqtt_client_handle_t client)
{
    /* The last argument may be used to pass data to the event handler, in this example mqtt_event_handler
     */
    esp_mqtt_client_register_event(client, ESP_EVENT_ANY_ID, mqtt_event_handler, NULL);
    esp_mqtt_client_start(client);

    //debug
    esp_mqtt_client_publish(client, "/topic/qos0", "experiment", 0, 0, 0);
}


void app_main(void)
{
    ESP_LOGI(TAG, "[APP] Startup..");
    ESP_LOGI(TAG, "[APP] Free memory: %" PRIu32 " bytes", esp_get_free_heap_size());
    ESP_LOGI(TAG, "[APP] IDF version: %s", esp_get_idf_version());

    esp_log_level_set("*", ESP_LOG_INFO);
    esp_log_level_set("esp-tls", ESP_LOG_VERBOSE);
    esp_log_level_set("MQTT_CLIENT", ESP_LOG_VERBOSE);
    esp_log_level_set("MQTT_EXAMPLE", ESP_LOG_VERBOSE);
    esp_log_level_set("TRANSPORT_BASE", ESP_LOG_VERBOSE);
    esp_log_level_set("TRANSPORT", ESP_LOG_VERBOSE);
    esp_log_level_set("OUTBOX", ESP_LOG_VERBOSE);

    ESP_ERROR_CHECK(nvs_flash_init());
    ESP_ERROR_CHECK(esp_netif_init());
    ESP_ERROR_CHECK(esp_event_loop_create_default());

    //!-----------
    wifi_connection();
    vTaskDelay(2000 / portTICK_PERIOD_MS);
    printf("WIFI was initiated ...........\n");
    //!----------


    /* This helper function configures Wi-Fi or Ethernet, as selected in menuconfig.
     * Read "Establishing Wi-Fi or Ethernet Connection" section in
     * examples/protocols/README.md for more information about this function.
     */

    const esp_mqtt_client_config_t mqtt_cfg = {
            //broker url or uri
        .broker = {
            .address.uri = "mqtts://(CHANGE TO YOUR ACTUAL URL).s1.eu.hivemq.cloud:8883/mqtt",
            //CONFIG_BROKER_URI,
            .verification.certificate = (const char *)mqtt_eclipseprojects_io_pem_start
        },
        //this is where I put the credentials
            //I added this
            .credentials = {
            .username="INSERT MQTT CREDENTIALS",
            .authentication = {
            .password="INSERT MQTT PASSWORD"
            }
        },

    };

    //ESP_LOGI(TAG, "[APP] Free memory: %" PRIu32 " bytes", esp_get_free_heap_size());
    esp_mqtt_client_handle_t MQTtest = esp_mqtt_client_init(&mqtt_cfg);


    mqtt_app_start(MQTtest);




    while(1){
            //count for debugging
            test_count++;

            //this function builds the "payload" string to be sent
            //edit it in order to change the payload
            custom_data();
            //this function is responsible for sending the payload variable
            custom_upload(MQTtest);

            //delay
            vTaskDelay(5000 / portTICK_PERIOD_MS);

    }
}
```

Appendix M

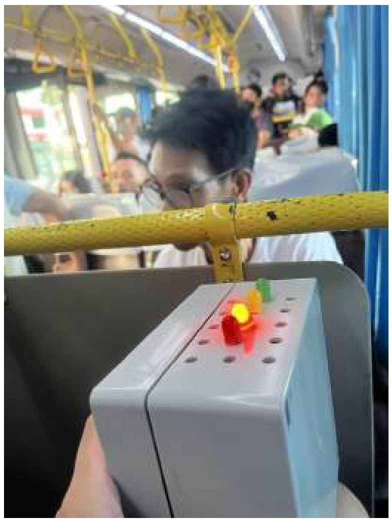
Code for Occupancy Counting (Wi-Fi Scanning)

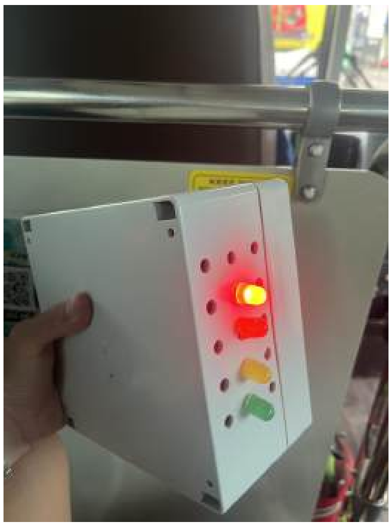
WiFi Scanner Template Source Code:
https://www.hackster.io/p99will/esp32-wifi-mac-scanner-sniffer-promiscuous-4c12f4
ESP32 WiFi Documentation (for implementing RSSI):
https://docs.espressif.com/projects/esp-idf/en/stable/esp32/api-reference/network/esp_wifi.html?highlight=wifi_promiscuous_pkt_t#_CPPv422wifi_promiscuous_pkt_t
Reference for Implementing Time:
https://www.circuitbasics.com/using-an-arduino-ethernet-shield-for-timekeeping
Reference for SD Card Module:
https://randomnerdtutorials.com/esp32-microsd-card-arduino/
Reference for UART Communication (sending/receiving string data):
https://forum.arduino.cc/t/serial-input-basics-updated/382007/2

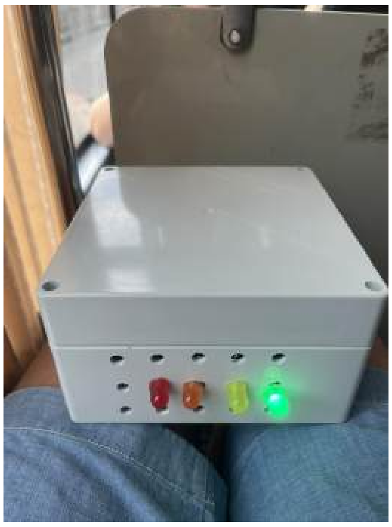
```
#include <WiFi.h>
#include <Wire.h>
#include "esp_wifi.h"
#include <TimeLib.h>
#include "FS.h"
#include "SD.h"
#include "SPI.h"

#define RXD2 17 // orange wire
#define TXD2 16 // yellow wire

//START: Declaration of List and Variables

//------mac address variables------------------------------------------
String maclist[200][4];//list of MAC Addresses
int listcount = 0; //index count of list
int onlinecount = 0; //counts the number of people

String defaultTTL = "60"; // Maximum time (Apx seconds) elapsed before device is considered offline


//------integers for buffer time-------------------------------------------
int on_buffer=5;

//CHANGE buffer based on travel speed
int low_buffer=5;
int high_buffer=5;
int rssi_filter=-60;

//------integers for capacity----------------------------------------
int max_capacity=50;//seating capacity (number of seats)
int percent=0;//percent capacity


//------integers for speed-------------------------------------------
int gps_threshold=10; //km per hour
int gps_speed=5; //placeholder default


//-------more declarations---------------------------------------------------------

//wifi struct object
const wifi_promiscuous_filter_t filt={
    .filter_mask=WIFI_PROMIS_FILTER_MASK_MGMT|WIFI_PROMIS_FILTER_MASK_DATA
};

typedef struct {
  uint8_t mac[6];
} __attribute__((packed)) MacAddr;

typedef struct {
  int16_t fctl;
  int16_t duration;
  MacAddr da;
  MacAddr sa;
  MacAddr bssid;
  int16_t seqctl;
  unsigned char payload[];
} __attribute__((packed)) WifiMgmtHdr;
```

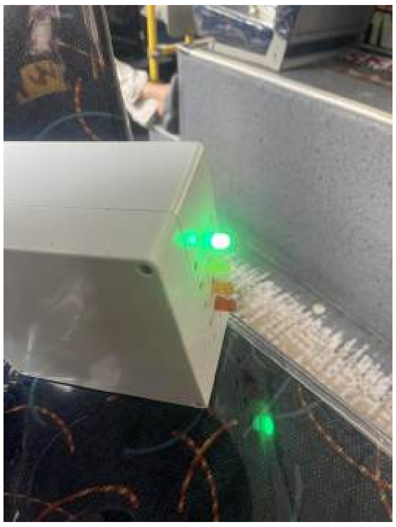
```
//max Channel -> US = 11, EU = 13, Japan = 14
#define maxCh 13
int curChannel = 1;


//------------------------------------------------------------------------------------
//------------------------------------------------------------------------------------


//====================== Wifi scanner function ========================================================
void sniffer(void* buf, wifi_promiscuous_pkt_type_t type) { //This is where packets end up after they get
    sniffed
  wifi_promiscuous_pkt_t *p = (wifi_promiscuous_pkt_t*)buf;
  int len = p->rx_ctrl.sig_len;
  WifiMgmtHdr *wh = (WifiMgmtHdr*)p->payload;
  len -= sizeof(WifiMgmtHdr);
  if (len < 0){
    Serial.println("Received 0");
    return;
  }
  String packet;
  String mac;
  int fctl = ntohs(wh->fctl);
  for(int i=8;i<=8+6+1;i++){ // This reads the first couple of bytes of the packet. This is where you can
    read the whole packet replaceing the "8+6+1" with "p->rx_ctrl.sig_len"
     packet += String(p->payload[i],HEX);
  }
  for(int i=4;i<=15;i++){ // This removes the 'nibble' bits from the stat and end of the data we want. So
    we only get the mac address.
    mac += packet[i];
  }
  mac.toUpperCase();

  int rssi = p->rx_ctrl.rssi; // NEW: get RSSI of signal

  int added = 0;

  for(int i=0;i<=199;i++){ // checks if the MAC address has been added before

    if(mac == maclist[i][0]){
      //"if added: set timer to 60. Set offline to 0"
      maclist[i][1] = defaultTTL;
      if(maclist[i][2] == "OFFLINE"){
        maclist[i][2] = "0";
      }
      maclist[i][3] = String(rssi); // NEW: update RSSI of wifi
      added = 1;
    }
  }

  if(added == 0){ // If its new. add it to the array.
    maclist[listcount][0] = mac;
    maclist[listcount][1] = defaultTTL;
    maclist[listcount][3] = String(rssi); // NEW: Store RSSI value
    //Serial.println(mac);
    listcount ++;
    if(listcount >= 200){
      Serial.println("Too many addresses");
      listcount = 0;
    }
  }
}



//=======================SD Card Functions ==========================================================
void listDir(fs::FS &fs, const char * dirname, uint8_t levels){
    Serial.printf("Listing directory: %s\n", dirname);

    File root = fs.open(dirname);
    if(!root){
        Serial.println("Failed to open directory");
        return;
    }
    if(!root.isDirectory()){
        Serial.println("Not a directory");
        return;
    }

    File file = root.openNextFile();
    while(file){
        if(file.isDirectory()){
             Serial.print("  DIR : ");
             Serial.println(file.name());
             if(levels){
             listDir(fs, file.path(), levels -1);
             }
        } else {
            Serial.print("  FILE: ");
            Serial.print(file.name());
            Serial.print("  SIZE: ");
            Serial.println(file.size());
        }
        file = root.openNextFile();
    }
}

void createDir(fs::FS &fs, const char * path){
    Serial.printf("Creating Dir: %s\n", path);
    if(fs.mkdir(path)){
        Serial.println("Dir created");
    } else {
        Serial.println("mkdir failed");
    }
}

void removeDir(fs::FS &fs, const char * path){
    Serial.printf("Removing Dir: %s\n", path);
    if(fs.rmdir(path)){
        Serial.println("Dir removed");
    } else {
        Serial.println("rmdir failed");
    }
}

void readFile(fs::FS &fs, const char * path){
    Serial.printf("Reading file: %s\n", path);

    File file = fs.open(path);
    if(!file){
        Serial.println("Failed to open file for reading");
        return;
    }

    Serial.print("Read from file: ");
    while(file.available()){
        Serial.write(file.read());
    }
    file.close();
}

void writeFile(fs::FS &fs, const char * path, const char * message){
    Serial.printf("Writing file: %s\n", path);

    File file = fs.open(path, FILE_WRITE);
    if(!file){
        Serial.println("Failed to open file for writing");
        return;
    }
    if(file.print(message)){
        Serial.println("File written");
    } else {
        Serial.println("Write failed");
    }
    file.close();
}

void appendFile(fs::FS &fs, const char * path, const char * message){
    Serial.printf("Appending to file: %s\n", path);

    File file = fs.open(path, FILE_APPEND);
    if(!file){
        Serial.println("Failed to open file for appending");
        return;
    }
    if(file.print(message)){
        Serial.println("Message appended");
    } else {
        Serial.println("Append failed");
    }
    file.close();
}

void renameFile(fs::FS &fs, const char * path1, const char * path2){
    Serial.printf("Renaming file %s to %s\n", path1, path2);
    if (fs.rename(path1, path2)) {
        Serial.println("File renamed");
    } else {
        Serial.println("Rename failed");
    }
}

void deleteFile(fs::FS &fs, const char * path){
    Serial.printf("Deleting file: %s\n", path);
    if(fs.remove(path)){
        Serial.println("File deleted");
    } else {
        Serial.println("Delete failed");
    }
}

void testFileIO(fs::FS &fs, const char * path){
    File file = fs.open(path);
    static uint8_t buf[512];
    size_t len = 0;
    uint32_t start = millis();
    uint32_t end = start;
    if(file){
        len = file.size();
        size_t flen = len;
        start = millis();
        while(len){
             size_t toRead = len;
             if(toRead > 512){
             toRead = 512;
             }
             file.read(buf, toRead);
             len -= toRead;
        }
        end = millis() - start;
        Serial.printf("%u bytes read for %u ms\n", flen, end);
        file.close();
    } else {
        Serial.println("Failed to open file for reading");
    }


    file = fs.open(path, FILE_WRITE);
    if(!file){
        Serial.println("Failed to open file for writing");
        return;
    }

    size_t i;
    start = millis();
    for(i=0; i<2048; i++){
        file.write(buf, 512);
    }
    end = millis() - start;
    Serial.printf("%u bytes written for %u ms\n", 2048 * 512, end);
    file.close();
}

void SD_setup(){
    if(!SD.begin()){
        Serial.println("Card Mount Failed");
        return;
    }
    uint8_t cardType = SD.cardType();

    if(cardType == CARD_NONE){
        Serial.println("No SD card attached");
        return;
    }

    Serial.print("SD Card Type: ");
    if(cardType == CARD_MMC){
        Serial.println("MMC");
    } else if(cardType == CARD_SD){
        Serial.println("SDSC");
    } else if(cardType == CARD_SDHC){
        Serial.println("SDHC");
    } else {
        Serial.println("UNKNOWN");
    }

    uint64_t cardSize = SD.cardSize() / (1024 * 1024);
    Serial.printf("SD Card Size: %lluMB\n", cardSize);

    listDir(SD, "/", 0);
    createDir(SD, "/mydir");
    listDir(SD, "/", 0);
    removeDir(SD, "/mydir");
    listDir(SD, "/", 2);
    writeFile(SD, "/hello.txt", "Hello ");
    appendFile(SD, "/hello.txt", "World!\n");
    readFile(SD, "/hello.txt");
    deleteFile(SD, "/foo.txt");
    renameFile(SD, "/hello.txt", "/foo.txt");
    readFile(SD, "/foo.txt");
    testFileIO(SD, "/test.txt");
    Serial.printf("Total space: %lluMB\n", SD.totalBytes() / (1024 * 1024));
    Serial.printf("Used space: %lluMB\n", SD.usedBytes() / (1024 * 1024));
    //bus setup
    appendFile(SD, "/Occupancy_Log.txt", "Data Logging Begins Now:\n");
}

//========================SD Card END ===========================================================//


//========================Wifi Scanner Functions
    ===========================================================//

/*
Purge Function
- checks if not empty
- if offline, ignore
- if active, subtract 1 second from TTL
- store back to list as string

*/

void purge(){ // This manages the TTL
  for(int i=0;i<=199;i++){
    if(!(maclist[i][0] == "")){ //if not empty, subtract time
      int ttl = (maclist[i][1].toInt());
      ttl --;
      if(ttl <= 0){
        Serial.println("OFFLINE: " + maclist[i][0]);
        maclist[i][2] = "OFFLINE";
        maclist[i][1] = defaultTTL;
      }else{
        maclist[i][1] = String(ttl);
      }
    }
  }
}

/*
Update Function
- if not online, increase alive time "timehere"
- return as string

*/

void updatetime(){ // This updates the time the device has been online for
  for(int i=0;i<=199;i++){
    if(!(maclist[i][0] == "")){
      if(maclist[i][2] == "")maclist[i][2] = "0";
      if(!(maclist[i][2] == "OFFLINE")){
              int timehere = (maclist[i][2].toInt());
              timehere ++;
              maclist[i][2] = String(timehere);
      }

      //Serial.println(maclist[i][0] + " : " + maclist[i][2]);

    }
  }
}

/*
Update Function
- lots of Print Debugging
- "onlinecount" counts how many people
- online_buffer checks how many seconds alive
- lots of print debugging

*/


void showpeople(){ // This checks if the MAC is in the reckonized list and then displays it on the OLED
    and/or prints it to serial.
  String forScreen = "";
  Serial.print("\n reset \n");
  Serial.print("------------ \n");
  Serial.print("\n \n \n \n \n");
  onlinecount = 0;
  for(int i=0;i<=199;i++){
    String tmp1 = maclist[i][0];
    if(!(tmp1 == "")){
      if(!(maclist[i][2]== "OFFLINE")){
        //new if condition: buffer time
        //count how many seconds the device has been active before considering it as ONLINE

        //extract integer
        int timehere = (maclist[i][2].toInt());

        //compare with on_buffer
        if(timehere>=on_buffer){
            int rssi_check = (maclist[i][3].toInt());
            if(rssi_check >= rssi_filter){
            Serial.print(maclist[i][0] + "      RSSI:");
            Serial.print(maclist[i][3]);
            Serial.print("   (DEBUG) Time Alive:");
            Serial.print(timehere);
            Serial.print("\n");
            onlinecount++;
            }

        }

        //Serial.print(maclist[i][0] + "\n");
        //onlinecount++;

      }
    }
  }

  //calculate seating capacity % !!disable for now
  percent = onlinecount*100 / max_capacity;

  // update_screen_text(forScreen);

  //print current Online Buffer Time and current Bus Speed
  Serial.print("\nBuffer Time:");
  Serial.print(on_buffer);
  Serial.println(" seconds");
  Serial.print("Bus Speed:");
  Serial.print(gps_speed);
  Serial.println("km/h");

  //print the number of passengers on the bus
  Serial.print("Passengers on Board: ");
  Serial.println(onlinecount);

  //print the seating status in percent !!disable for now
  Serial.print("Seating Status:" );
  Serial.print(String(percent));
  Serial.print("% \n");
}

//========================Wifi Scanner Functions END
    ===========================================================//

//====================SETUP=======================================================================//
void setup() {

  /* start Serial */
  Serial.begin(115200);

  // setupUART();
  Serial2.begin(9600, SERIAL_8N1, RXD2, TXD2);

  //setup SD Card
  SD_setup();


  /* setup wifi */
  wifi_init_config_t cfg = WIFI_INIT_CONFIG_DEFAULT();
  esp_wifi_init(&cfg);
  esp_wifi_set_storage(WIFI_STORAGE_RAM);
  esp_wifi_set_mode(WIFI_MODE_NULL);
  esp_wifi_start();
  esp_wifi_set_promiscuous(true);
  esp_wifi_set_promiscuous_filter(&filt);
  esp_wifi_set_promiscuous_rx_cb(&sniffer);
  esp_wifi_set_channel(curChannel, WIFI_SECOND_CHAN_NONE);

  //start time
  setTime(0, 0, 0, 1, 1, 2024);

  Serial.println("starting!");
}



//==============occupancy SD Card
    log==============================================================================================
    ==============================================================
char currentTimeString[9];  // HH:MM:SS\0
char occupancy_payload[500];

void occupancy_log(){
  time_t t = now();
  sprintf(currentTimeString, "%02d:%02d:%02d", hour(t), minute(t), second(t));
  sprintf(occupancy_payload, "Time:  %s, Passengers: %d, Seating Status: %d%%, Bus Speed: %d km/h, Buffer
    Time: %d seconds\n", currentTimeString, onlinecount, percent, gps_speed, on_buffer);
  Serial.print(occupancy_payload);
  appendFile(SD, "/Occupancy_Log.txt", occupancy_payload);
}

//====================speed checker===============================================================//
int UARTholder = 0;

//Buffer time is adjusted based on bus speed
void speed_check(){
  // Debugging output
  //Serial.print("Bytes available in Serial2 buffer: ");
  //Serial.println(Serial2.available());

  //check current speed from border router
  if (Serial2.available() > 0) {
    Serial.print("!---------Buffer Detected--------------!\n");
    //Serial.print();
    UARTholder=Serial2.parseInt();

    if(UARTholder>0){
      Serial.print("!---------NON ZERO DETECTED--------------!");
      Serial.println(UARTholder);
      gps_speed=UARTholder;
    }

  }

  if(gps_speed<gps_threshold){
    //if bus is slow or stopped, increase buffer time in order to minimize false positives
    //change to high buffer time
    on_buffer=high_buffer;
  } else {
    //if bus is moving above the required speed, return back to original buffer time
    //change to low buffer time
    on_buffer=low_buffer;
  }

}

void send_occupancy(){
  //send as string
  Serial2.print(String(onlinecount));
  Serial2.print("\n");

}

//===== LOOP =====//
void loop() {
    //Serial.println("Changed channel:" + String(curChannel));
    if(curChannel > maxCh){
      curChannel = 1;
    }
    esp_wifi_set_channel(curChannel, WIFI_SECOND_CHAN_NONE);
    delay(1000);
    speed_check();
    updatetime();
    purge();
    showpeople();
    occupancy_log();
    send_occupancy();
    curChannel++;

}
```

Appendix N
Bus Field Testing Documentation









Appendix O
Car and E-Trike Field Testing Documentation