PESUDO CODES FOR ROCKET PAYLOAD-TRACKER/TELEMETRY SYSTEM PROJECT

1. Receiving and reading the data send from the flight controller board (sensors) through the micro DB9 female port.

```
/*
ESp32-C3 module reads the sensor data coming through the RS232.
switch into the UART port serial interface through the DB9 female port.
--------------------------------------------------------------------------------
*/
//define ESP32 UART PINS ( Used ESP32-C3-M1 Kit)
#define ESP32_RX_PIN 27 // GPIO pin number for ESP32 RX
#define ESP32_TX_PIN 28 //GPIO pin number for ESP32 TX


//Define DB9 pinout for RS232 communication
#define ESP32_RX_PIN 3 // pin number on DB9 connector for receiving the data
#define ESP32_TX_PIN 2 // pin number on DB9 connector for transmitting the data


//Define RS232 port settings for receiving data from RS232 into the DB9 port
#define RS232_SERIAL Serial1
#define RS232_BAUDRATE 9600  //Give Baud rate that is supported by the RS232 device.
```

```
//LoRa settings for transmitting the data
#define LORA_CS_PIN && // Give the corresponding pin number for LoRa chip select pin
#define LORA__RST_IN && //Give the corresponding pin number for LoRa chip reset pin
#define LORA_INT_PIN && //Give the corresponding pin number for LoRa Digital input pin


void setup() {
Serial.begin(9600); // select the baud rate which is compatible with RS232.


//Initialize the LORA module
 LoRa.setPins(LORA_CS_PIN, LORA_RST_PIN, LORA_INT_PIN);


  if (!LoRa.begin(915E6)) // 915MHz is the specified frequency range of the LORA
                                    module & the begin function returns a  Boolean value
  {
    Serial.println("LoRa initialization failed. Please check your connections.");
    while (1);
  }
  Serial.println("LoRa initialized");
          }


void loop() {
  // Read the data from the RS232 port and if the data is received, then
  if (Serial.available()) {
    String data = Serial.readStringUntil('\n'); // received data is stored in a string variable called
                                                            'data'
```

```
/*
The code below is used for getting the GPS location using the GPS module NEO-M8N
and transmitting the data through the LORA-RF95M transceiver. Also for data logging purpose,
data has been sent to the SD card present within the PCB board.
****NB: The GPS data needs to be logged for tracking
-------------------------------------------------------------------------------------------
*/
#include <TinyGPS++.h>      //include header file to interact with the TinyGPS
                                   //library
#include <SPI.h>
#include <LoRa.h>
#include <SD.h>


// GPS module settings
#define GPS_SERIAL Serial2  // PIN where GPS module is connected
#define GPS_BAUDRATE 9600  // Set the Baud rate which is compatible with the GPS module


// LoRa settings
#define LORA_CS_PIN &&   // Provide the LoRa chip select pin number
#define LORA_RST_PIN 6  // LoRa reset pin
#define LORA_INT_PIN 14  // LoRa DIO0 pin


// SD card settings
// If an SD card/memory card is used for storing the GPS data
#define SD_CS_PIN &&  // Provide the CS pin for SD card module


// Create a TinyGPS++ object
TinyGPSPlus gps; // An object named 'gps' is created to interact with the TinyGPS plus library.
                 //This object receives useful information from the raw data strings transmitted by
                 the GPS module
```

```
// File object for SD card
File dataFile;
void setup() {
  Serial.begin(115200);  // For debugging- baud rate ( choose one which is
           //compatible)
  GPS_SERIAL.begin(GPS_BAUDRATE);  // Initialize GPS serial communication
  // Initialize LoRa module
  LoRa.setPins(LORA_CS_PIN, LORA_RST_PIN, LORA_INT_PIN);
  if (!LoRa.begin(915E6)) //915MHZ frequency for operation
  {
    Serial.println("LoRa initialization failed. Check your connections.");
    while (1);
  }
  // Initialize SD card
  if (!SD.begin(SD_CS_PIN)) {
    Serial.println("SD card initialization failed. Check your connections.");
    while (1);
  }
  Serial.println("Initialization completed");
}
void loop() {
  // Read data from GPS module
  while (GPS_SERIAL.available() > 0) {
    if (gps.encode(GPS_SERIAL.read())) {

      // Send GPS data over LoRa
      sendGPSDataOverLoRa();

      // Log GPS data to SD card
      logGPSDataToSD();
    }
  }
}
```

```
void sendGPSDataOverLoRa() {
  // Check if GPS data is valid
  if (gps.location.isValid() && gps.date.isValid() && gps.time.isValid()) {
    // Construct a message containing GPS data
    String message = String(gps.location.lat(), 6) + "," + String(gps.location.lng(), 6) + "," +
                     String(gps.date.month()) + "/" + String(gps.date.day()) + "/" + String(gps.date.year()) + " "
+
                     String(gps.time.hour()) + ":" + String(gps.time.minute()) + ":" + String(gps.time.second());
    // Send message over LoRa
    LoRa.beginPacket();
    LoRa.print(message);
    LoRa.endPacket();
  }
}
void logGPSDataToSD() {
  // Open file on SD card in append mode
  dataFile = SD.open("gps_data.txt", FILE_WRITE);
  // If the file is available, write GPS data to it
  if (dataFile) {
    if (gps.location.isValid() && gps.date.isValid() && gps.time.isValid()) {
      dataFile.print("Location: ");
      dataFile.print(gps.location.lat(), 6); // latitude formatted to 6 decimal places
      dataFile.print(",");
      dataFile.print(gps.location.lng(), 6);
      dataFile.print("  Date/Time: ");
      dataFile.print(gps.date.month());
      dataFile.print("/");
      dataFile.print(gps.date.day());
      dataFile.print("/");
      dataFile.print(gps.date.year());
      dataFile.print(" ");
```

```
 dataFile.print(" ");
      dataFile.print(gps.time.hour());
      dataFile.print(":");
      dataFile.print(gps.time.minute());
      dataFile.print(":");
      dataFile.print(gps.time.second());
      dataFile.println();
    } else {
      dataFile.println("GPS data invalid");
    }

    dataFile.close();  // Close the file
  } else {
    Serial.println("Error opening file for writing");
  }
}
```

2. Transition of ESP32 microcontroller to light sleep for reducing the power consumption.

```
/*The code below is used to set the ESP32 microcontroller to light sleep mode*/
void setup() {
  Serial.begin(115200);  // put your setup code here, to run once:
}
void loop() { // put your main code here, to run repeatedly:
  Serial.println("light_sleep_enter");
  esp_sleep_enable_timer_wakeup(1000000); //1 seconds
  int ret = esp_light_sleep_start();
  Serial.printf("light_sleep: %d\n", ret);
}
```

3. Pseudo code for radio triangulation

```
function [transmitter_x, transmitter_y, transmitter_lat, transmitter_lon] =
triangulation(receiver1_toa, receiver2_toa, receiver3_toa, receiver1_lat, receiver1_lon,
receiver2_lat, receiver2_lon, receiver3_lat, receiver3_lon)

    % Convert latitude and longitude coordinates to Cartesian coordinates (x, y)

    receiver1_location = latlon_to_cartesian(receiver1_lat, receiver1_lon);

    receiver2_location = latlon_to_cartesian(receiver2_lat, receiver2_lon);

    receiver3_location = latlon_to_cartesian(receiver3_lat, receiver3_lon);

    % Calculate distances between the transmitter and each receiver based on TOA

    % Assume speed of propagation (e.g., speed of light)

    speed_of_propagation = 299792458; % Speed of light in meters per second

    distance1 = receiver1_toa * speed_of_propagation;

    distance2 = receiver2_toa * speed_of_propagation;

    distance3 = receiver3_toa * speed_of_propagation;

    % Perform trilateration to estimate transmitter location

    transmitter_location = trilateration(receiver1_location, receiver2_location,
receiver3_location, [distance1, distance2, distance3]);

    % Extract x and y coordinates

    transmitter_x = transmitter_location(1);

    transmitter_y = transmitter_location(2);

    % Convert Cartesian coordinates back to latitude and longitude

    [transmitter_lat, transmitter_lon] = cartesian_to_latlon(transmitter_x, transmitter_y);

end

function transmitter_location = trilateration(receiver1_location, receiver2_location,
receiver3_location, distances)

    % Trilateration algorithm

    % Convert receiver locations to matrices

    P1 = receiver1_location(:)';

    P2 = receiver2_location(:)';

    P3 = receiver3_location(:)';

    % Calculate unit vectors from each receiver to the transmitter

    u1 = (P1 - P2) / norm(P1 - P2);

    u2 = (P1 - P3) / norm(P1 - P3);
```

```
    % Calculate transmitter location
    a = (distances(1)^2 - distances(2)^2 + norm(P2 - P1)^2) / (2 * norm(P2 - P1));
    b = (distances(1)^2 - distances(3)^2 + norm(P3 - P1)^2) / (2 * norm(P3 - P1));
    % Transmitter location in 2D (assuming z = 0)
    transmitter_location = P1 + a * u1 + b * u2;
end
function [latitude, longitude] = cartesian_to_latlon(x, y)
    % Convert Cartesian coordinates (x, y) to latitude and longitude
    % Assuming Earth is a perfect sphere for simplicity
    % Radius of the Earth (in meters)
    R = 6371000;
    % Calculate latitude and longitude
    lat_rad = asin(y / R);
    lon_rad = atan2(x, cos(lat_rad) * R);
    % Convert radians to degrees
    latitude = rad2deg(lat_rad);
    longitude = rad2deg(lon_rad);
end
function cartesian_location = latlon_to_cartesian(latitude, longitude)
    % Convert latitude and longitude coordinates to Cartesian coordinates (x, y)
    % Assuming Earth is a perfect sphere for simplicity
    % Radius of the Earth (in meters)
    R = 6371000;
    % Convert latitude and longitude to radians
    lat_rad = deg2rad(latitude);
    lon_rad = deg2rad(longitude);
    % Calculate Cartesian coordinates
    x = R * cos(lat_rad) * cos(lon_rad);
    y = R * cos(lat_rad) * sin(lon_rad);
    cartesian_location = [x, y];
end
```