

(Company No. 101067-P)

الجامعة الإسلامية العالمية ماليزيا  
INTERNATIONAL ISLAMIC UNIVERSITY MALAYSIA  
يُونِيسَيْتِي إِسْلَامِيَّةٌ أَنْتَارَايَغْسِيَا مَلَيْسِيَا

*Garden of Knowledge and Virtue*

DEPARTMENT OF MECHATRONICS ENGINEERING

**MINI PROJECT:**  
**REAL-TIME ROCKET TELEMETRY SYSTEM**

---

MCTE 4324

REAL TIME SYSTEMS

SECTION 1

SEM 2, 20/21

---

**LECTURER:**

DR. ZULKIFLI ZAINAL ABIDIN

PREPARED BY:

---

AHMAD FAAIZ BIN AZMAN SHAH	1721421
----------------------------	---------

---

DATE SUBMITTED:

26 JUNE 2021

## **EXECUTIVE SUMMARY**

This report is a project documentation of designing a low-cost real-time rocket telemetry system. The requirements and equipment used for the projects are stated in the early part of the report. The general objective of this project is to design a rocket telemetry system that captures data in real-time and the data are analysed to gain further information of the rocket. By implementing Real Time Operating System (RTOS) components in the programming of the microcontroller, I get to capture data from the GPS in real time and achieve the objective of the project. Details of the implementation of the RTOS components are explained in this report that suited for the project. Some safety issues and precautions are stated so that everyone take cautions on the matter. At the end of the report, conclusions are made based on what have been achieved and some future works are discussed.

## TABLE OF CONTENTS

EXECUTIVE SUMMARY .....	2
TABLE OF CONTENTS.....	3
BACKGROUND .....	4
OBJECTIVES .....	5
DETAILS.....	6
CIRCUIT CONFIGURATIONS .....	6
CODING REFERENCES.....	8
OVERALL SYSTEM FLOWCHART .....	9
IMPLEMENTATION OF RTOS COMPONENTS .....	11
1. Task Scheduling .....	11
2. Memory Management.....	12
3. Queue.....	13
4. Mutex.....	13
5. Semaphore .....	14
6. Software Timer .....	15
SERIAL MONITOR (RECEIVER).....	15
SAFETY & PRECAUTIONS.....	16
CONCLUSION & FUTURE WORKS.....	16
APPENDIX A: TRANSMITTER CODE.....	17
APPENDIX B: RECEIVER CODE.....	23

## BACKGROUND

This project aims to design a rocket telemetry system that utilizes RTOS components inside the programming for receiving data and transmitting data in real time between two microcontrollers. One of the microcontrollers, which is attached to the rocket will send or transmit data from GPS and is called Transmitter throughout this report. The other microcontroller is the receiver which will receive the data transmit from the transmitter and inside the receiver code, the code creates a user interface at the serial monitor so that the user can interact with it and get specific information. The microcontroller used for the transmitter is ESP32 Devkit which also connects to the GPS NEO 6M and LoRa SX1278 while the other microcontroller used for the receiver is a standalone TTGO SX1278 LoRa ESP32.

Requirements:

- The transmitter circuit board must fit the size of casting as stated below:
  - Outer Diameter: 70mm
  - Length: 15cm
- Smaller antenna is preferable
- Output data format required:
  - Time
  - Height/Altitudes
  - Speed
  - Coordinate
    - Latitude
    - Longitude
- Long range telemetry
- Telemetry system must be in real time

Hardware/Equipment:

*Table 1: Items used for the project*

Item	Cost per unit (RM)	Unit
LoRa SX1278	24.90	1
433MHZ Antenna	3.50	2
ESP32 Devkit V1	29.9	1
TTGO SX1278 LoRa ESP32	50.09	1
GPS Module NEO 6M	21.90	1

Total cost of the crucial items for the project.

Total Cost = RM 133.79

Estimated total cost including other necessary items, services, and others.

Estimated Total Cost = RM 150+

## OBJECTIVES

1. To design a low-cost real-time rocket telemetry system.
2. To fulfil the design requirement for the rocket telemetry system.
3. To capture time, altitude, speed, and coordinate information from the GPS and transmit the data to a said ground station.
4. To test for real time data transmission between two microcontrollers and record data from GPS.

## DETAILS

### CIRCUIT CONFIGURATIONS

#### Transmitter

Referring to the figure below, the RST changed to GPIO4 so that every wire ended up at only one side of the ESP32 for convenience purposes.

LoRa Transmitter Pins:

SS = 5

RST = 4

DIO0 = 2

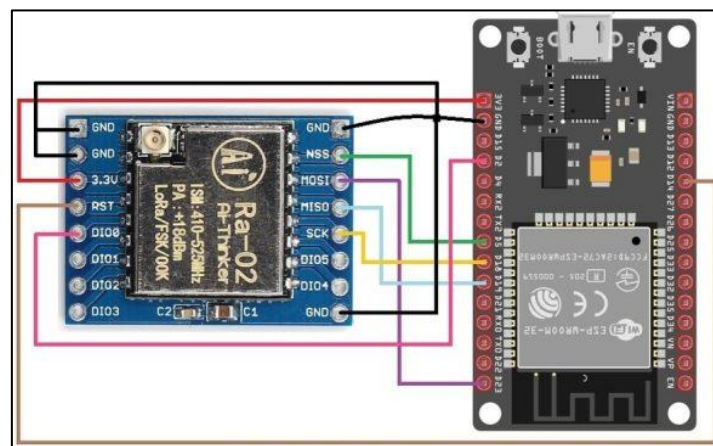


Figure 1: Wire connections between ESP32 and LoRa SX1278

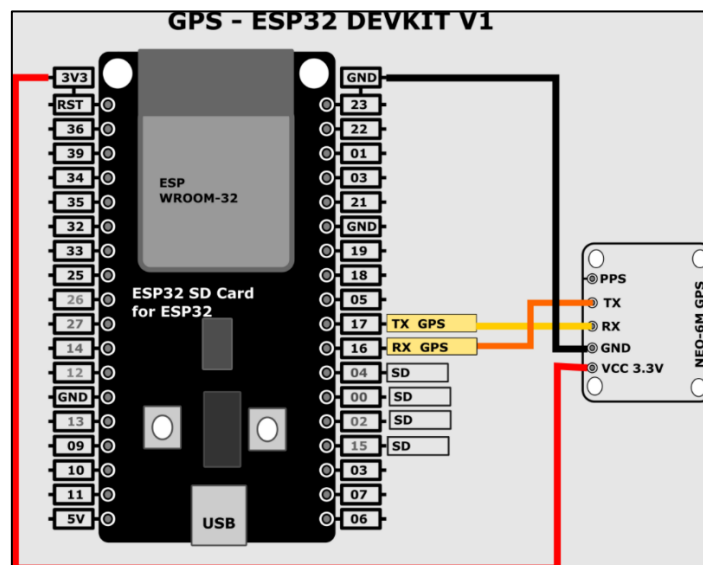


Figure 2: Wire Connections between ESP32 and GPS NEO 6M

## Receiver

LoRa Receiver Pins:

SS = 18

RST = 14

DIO0 = 26

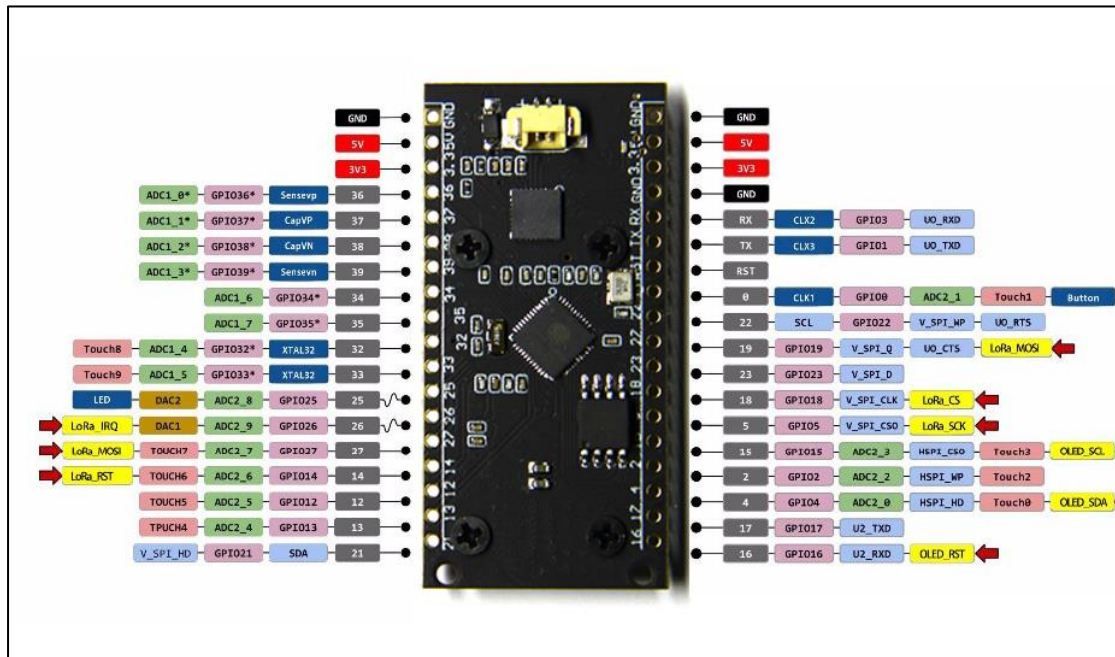


Figure 3: TTGO LoRa SX1278 ESP32 Pins

## **CODING REFERENCES**

### **Tasks in Transmitter Code**

1. gpsReader
2. loraSend

### **Tasks in Receiver Code**

1. loraReceive
2. printAll
3. promptData
4. printPacketNum
5. launchRocket

### **Data taken from GPS NEO 6M**

The TinyGPS++ library is used to parse information from the raw GPS data. The date and time I got from this library is in time zone of Universal Time Coordinated (UTC). To better suit with the local time, I use the following function:

```
static int calcLocalTime(TinyGPSTime &t) {  
    if (t.hour() >= 16)  
        return (t.hour() - 16);  
    else if ((t.hour() >= 0) && (t.hour() <= 15))  
        return (t.hour() + 8);  
}
```

The date is still in the time zone of UTC. Although there might be a fix for the date, the date data is not important as it is not one of the required data needed as stated in the requirements.



## OVERALL SYSTEM FLOWCHART

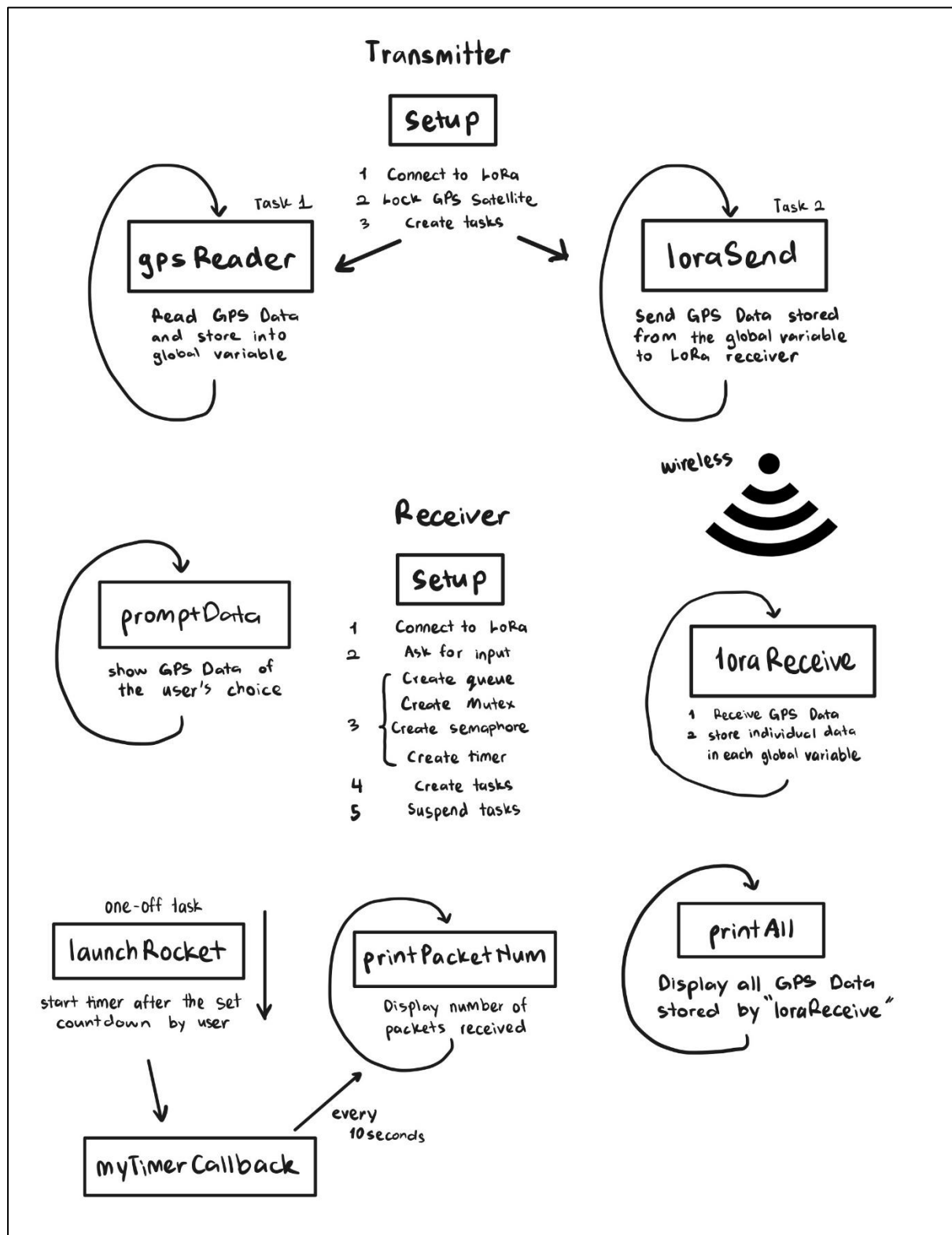


Figure 4: System Flowchart of Transmitter and Receiver

Most of the tasks in the Receiver are suspended as soon as it is created except for the “lorareceive” task. These tasks are handled inside the loop function where it requires some user’s input.

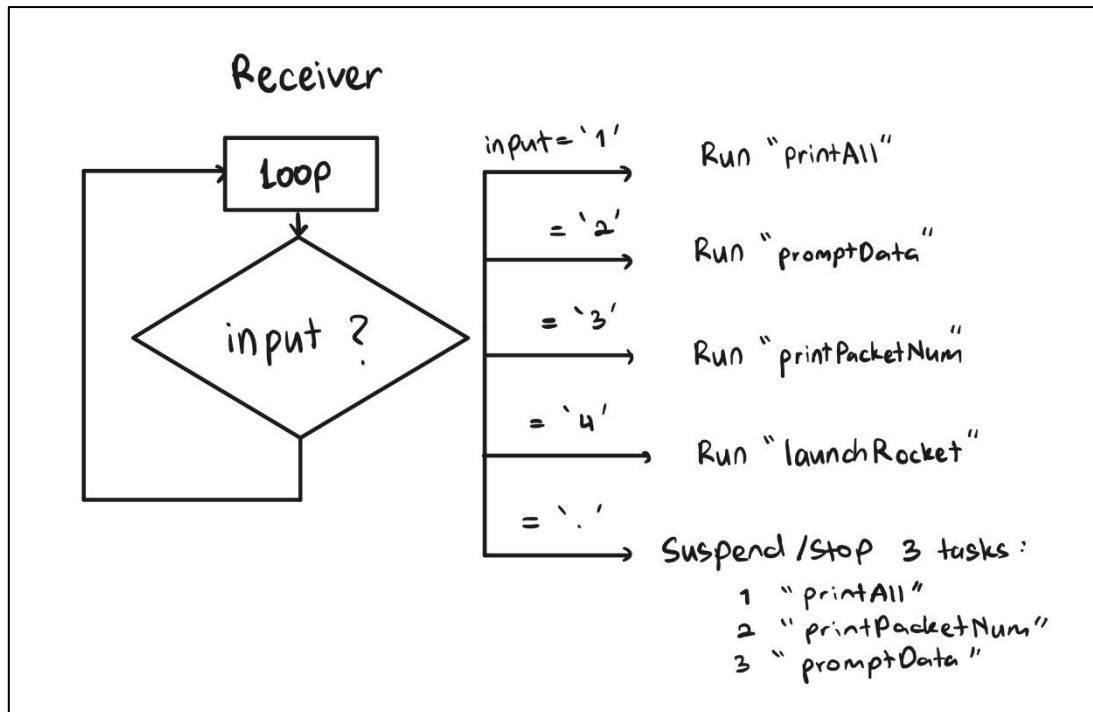


Figure 5: Task handling with user’s input

## IMPLEMENTATION OF RTOS COMPONENTS

### 1. Task Scheduling

The “vTaskSuspend()” and “vTaskResume()” functions are used to schedule and handle the tasks. In the receiver code, tasks are handled according to the user’s input in the loop function as shown in Figure 2 and the following code.

```
void loop() {  
    while (Serial.available()) {  
        char input = Serial.read();  
        xQueueSend(input_queue, (void *)&input, 10);  
        if (input == '1') {  
            vTaskResume(printall);  
        } else if (input == '2') {  
            vTaskResume(promptdata);  
            Serial.println("(5) for DATE");  
            Serial.println("(6) for TIME");  
            Serial.println("(7) for ALTITUDE");  
            Serial.println("(8) for SPEED");  
            Serial.println("(9) for COORDINATE");  
        } else if (input == '3') {  
            vTaskResume(printpacket);  
        } else if (input == '4') {  
            vTaskResume(launch);  
        } else if (input == '.') {  
            Serial.println(F("Type in your command: "));  
            Serial.println("(1) to show all data in realtime");  
            Serial.println("(2) for prompting choice of data");  
            Serial.println("(3) to show # of packets received since");  
            Serial.println("(4) to initiate launching rocket");  
            Serial.println("(.) to EXIT");  
            vTaskSuspend(printall);  
            vTaskSuspend(printpacket);  
            vTaskSuspend(promptdata);  
        }  
    }  
}
```

## 2. Memory Management

Due to the small stack size for the “loraSend” task, the task faces a problem which is a stack overflow. This problem is solved by increasing the stack size or using heap memory and freeing up memory whenever possible.

```
xTaskCreatePinnedToCore(  
    loraSend,  
    "Send to RX",  
    1024,                // small stack size (bytes)  
    NULL,  
    1,  
    NULL,  
    app_cpu  
);
```

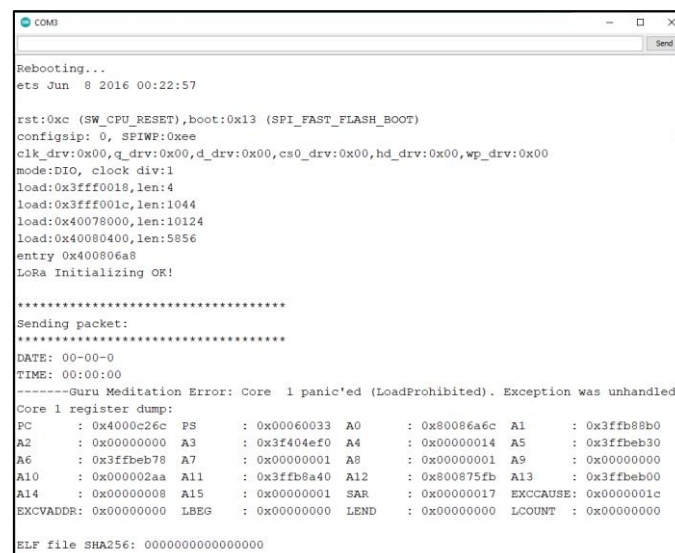


Figure 6: Error using small stack size for a specific task

```
xTaskCreatePinnedToCore(  
    loraSend,  
    "Send to RX",  
    5000,                // Sufficient stack size (bytes)  
    NULL,  
    1,  
    NULL,  
    app_cpu  
);
```

### 3. Queue

The queue used in this project is to send a data or a parameter between tasks. In the loop function of the receiver code, the user's input will be sent to the "promptData" task.

In loop function:

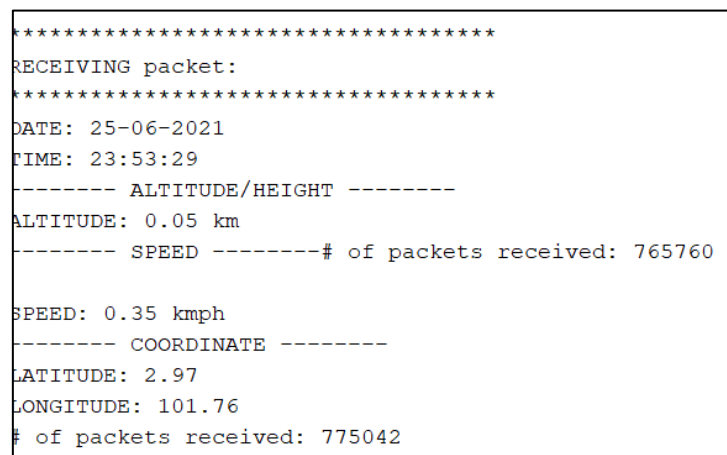
```
xQueueSend(input_queue, (void *)&input, 10);
```

In "promptData" task function:

```
char input;
if (xQueueReceive(input_queue, (void *)&input, 0) == pdTRUE)
{...
```

### 4. Mutex

There are problems when the programs tried to print different information in a short amount of time as shown in the figure below.



```
*****
RECEIVING packet:
*****
DATE: 25-06-2021
TIME: 23:53:29
----- ALTITUDE/HEIGHT -----
ALTITUDE: 0.05 km
----- SPEED -----# of packets received: 765760

SPEED: 0.35 kmph
----- COORDINATE -----
LATITUDE: 2.97
LONGITUDE: 101.76
# of packets received: 775042
```

*Figure 7: Serial output conflict without Mutex*

To solve this issue, mutex is used for the critical section to be done before any other progress.

```
xSemaphoreTake(mutex, portMAX_DELAY);

// Critical Section

xSemaphoreGive(mutex);
```

## 5. Semaphore

The semaphore used in this project is to pass a parameter when creating the “launchRocket” task. The semaphore makes sure the parameter is done copied inside the task for it to be used.

```
xTaskCreate(  
    launchRocket,  
    "Start timer since launch",  
    5000,  
    (void *)&countdown,  
    1,  
    &launch  
);  
// Do nothing until binary semaphore has been returned  
xSemaphoreTake(bin_sem, portMAX_DELAY);
```

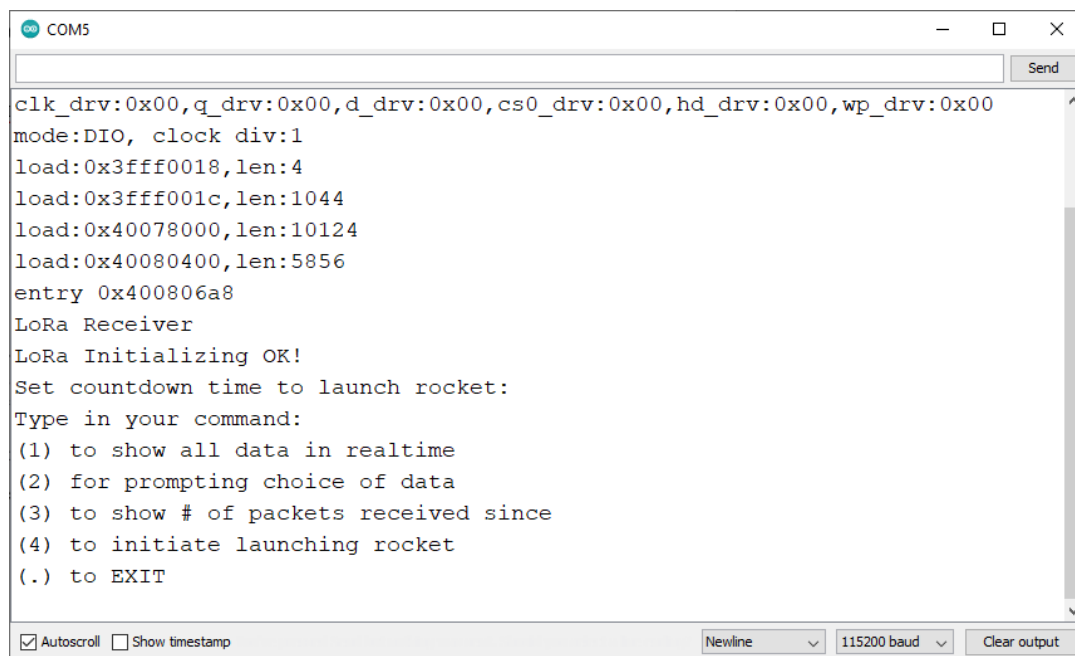
```
void launchRocket(void *parameter) {  
    while (1) {  
        // Copy the parameter into a local variable (from countdown)  
        int cd_num = *(int *)parameter;  
  
        // Release the binary semaphore so that the creating function can finish  
        xSemaphoreGive(bin_sem);  
        ...  
    }  
}
```

## 6. Software Timer

For this project, an auto-reload timer is used and will display some information for every 10 seconds after the timer is started. Assuming the timer will start when the rocket is launch which is when the “launchRocket” task gets to run when the user prompted it to. For every 10 seconds since the rocket was launched, the timer will run the “printPacketNum” task where it will display the number of packets received. Any other tasks could replace the stated task if I wish to change the displayed information.

```
void myTimerCallback(TimerHandle_t xTimer) {  
    // show # of packets received if timer id 1 expired  
    if ((uint32_t)pvTimerGetTimerID(xTimer) == 1) {  
        vTaskResume(printpacket);  
        vTaskDelay(500 / portTICK_PERIOD_MS);  
        vTaskSuspend(printpacket);  
    }  
}
```

## SERIAL MONITOR (RECEIVER)



*Figure 8: User Interface from Serial Monitor of Receiver*

## **SAFETY & PRECAUTIONS**

Some of the safety issues that need to be considered for this project:

- Some of the components are needed to be soldered. During the soldering process, make sure to handle the soldering equipment with extra caution as it might damage the components or harm other people.
- Most of the components used in this project requires only a low voltage, to avoid damaging these components, it is advised not to use a larger voltage than required.
- Cleanliness is also important, not cleaning up after setting up the circuits and anything else might cause an accident.

## **CONCLUSION & FUTURE WORKS**

In conclusion, I have implemented some of the Real Time Operating System (RTOS) programming components in order to design a low-cost real-time rocket telemetry system. It is quite unfortunate that I have not been able to implement all the RTOS components in the programming. Nonetheless, the conducted experiment is done using multiple breadboards with different sizes and numerous male-to-male wires. With that said, the design of printed circuit board is not done due to time constraint. Other than that, I have successfully tested a real time data transmission between two microcontrollers by capturing the time, altitude, speed, and the coordinate information from the GPS and transmit that data to the receiver.

Some of the possible works in continuation of this project are finding the course or trajectory of the rocket, finding the differences between the starting point before launch and the destination after launch of the rocket, connecting to Wi-Fi and displaying data through a web server, and hopefully future works could implement better and more RTOS components in the program of the rocket telemetry system.



## APPENDIX A: TRANSMITTER CODE

```
#if CONFIG_FREERTOS_UNICORE
static const BaseType_t app_cpu = 0;
#else
static const BaseType_t app_cpu = 1;
#endif

#include <SPI.h>
#include <LoRa.h>
#include <HardwareSerial.h>
#include <TinyGPS++.h>

// GPS
static const int RXPin = 16, TXPin = 17;
static const uint32_t GPSBaud = 9600;
TinyGPSPlus gps;

// LoRa Transmitter Pins
#define SS 5
#define RST 4
#define DI00 2

// Globals
String LoRaMessage = "";

// GPS DATA VALUE
int year_val;
int month_val;
int day_val;
int hour_val;
```

```

int minute_val;
int second_val;
double lat_val;
double lng_val;
double alt_km;
double speed_kmph;

// ----- TASKS -----

void gpsReader(void *parameter) {
    while (1) {
        while (Serial1.available() > 0) {
            gps.encode(Serial1.read());
            if (gps.date.isValid()) {
                year_val = gps.date.year();
                month_val = gps.date.month();
                day_val = gps.date.day();
            }
            if (gps.time.isValid()) {
                hour_val = calcLocalTime(gps.time);
                minute_val = gps.time.minute();
                second_val = gps.time.second();
            }
            if (gps.location.isValid()) {
                lat_val = gps.location.lat();
            }
            if (gps.location.isValid()) {
                lng_val = gps.location.lng();
            }
            if (gps.altitude.isValid()) {

```

```

        alt_km = gps.altitude.kilometers();
    }
    if (gps.speed.isValid()) {
        speed_kmph = gps.speed.kmph();
    }
}

// Wait for a while
vTaskDelay(100 / portTICK_PERIOD_MS);
}

void loraSend(void *parameter) {
    while (1) {

        Serial.println();
        Serial.println("*****");
        Serial.println("Sending packet:");
        Serial.println("*****");

        char arr1[32];
        sprintf(arr1, "DATE: %02d-%02d-%02d ", day_val, month_val, year_val);
        Serial.println(arr1);

        char arr2[32];
        sprintf(arr2, "TIME: %02d:%02d:%02d ", hour_val, minute_val, second_val);
        Serial.println(arr2);

        Serial.println("----- ALTITUDE/HEIGHT -----");
        Serial.print("ALTITUDE (in km): ");
    }
}

```

```

Serial.println(alt_km);

Serial.println("----- SPEED -----");
Serial.print("SPEED (kilometer/hour): ");
Serial.println(speed_kmph);

Serial.println("----- COORDINATE -----");
Serial.print("LATITUDE: ");
Serial.println(lat_val, 6);
Serial.print("LONGITUDE: ");
Serial.println(lng_val, 6);

LoRaMessage = String(day_val) + "!" + String(month_val) + "@" + S
tring(year_val) + "#"
              + String(hour_val) + "$" + String(minute_val) + "%"
+ String(second_val) + "^"
              + String(alt_km) + "&" + String(speed_kmph) + "*" +
String(lat_val) + "/" + String(lng_val);

LoRa.beginPacket();

LoRa.print(LoRaMessage);

LoRa.endPacket();

vTaskDelay(1000 / portTICK_PERIOD_MS);
}
}

void setup() {
  Serial.begin(115200);

```

```

Serial1.begin(GPSBaud, SERIAL_8N1, RXPin, TXPin);
// vTaskDelay(5000 / portTICK_PERIOD_MS);

// setup LoRa
LoRa.setPins(SS, RST, DIO0);
while (!LoRa.begin(433E6)) {
    Serial.println("LoRa connecting...");
    delay(500);
}
// Change sync word (0xF3) to match the receiver
LoRa.setSyncWord(0xF3);
Serial.println("LoRa Initializing OK!");

xTaskCreatePinnedToCore(
    gpsReader,
    "GPS Reader",
    1024,
    NULL,
    1,
    NULL,
    app_cpu
);

xTaskCreatePinnedToCore(
    loraSend,
    "Send to RX",
    5000,
    NULL,
    1,
    NULL,
    app_cpu

```

```
    );  
}  
void loop() {  
  
}  
static int calcLocalTime(TinyGPSTime &t) {  
    if (t.hour() >= 16)  
        return (t.hour() - 16);  
    else if ((t.hour() >= 0) && (t.hour() <= 15))  
        return (t.hour() + 8);  
}
```

## APPENDIX B: RECEIVER CODE

```
#if CONFIG_FREERTOS_UNICORE
static const BaseType_t app_cpu = 0;
#else
static const BaseType_t app_cpu = 1;
#endif

#include <SPI.h>
#include <LoRa.h>

// LoRa Receiver Pins
#define SS 18
#define RST 14
#define DIO0 26

// Task Handles for Task Scheduling
static TaskHandle_t printall;
static TaskHandle_t printpacket;
static TaskHandle_t promptdata;
static TaskHandle_t launch;

// Queue for passing data between tasks
static QueueHandle_t input_queue;
static const uint8_t msg_queue_len = 5;

// Timer
static TimerHandle_t one_shot_timer = NULL;
static TimerHandle_t auto_reload_timer = NULL;
```

```

// Mutex & Semaphore
static SemaphoreHandle_t mutex;
static SemaphoreHandle_t bin_sem;

// Globals
String day_val;
String month_val;
String year_val;
String hour_val;
String minute_val;
String second_val;
String alt_km;
String speed_kmph;
String lat_val;
String lng_val;
int packetnum = 0;

void myTimerCallback(TimerHandle_t xTimer) {

    // show # of packets received if timer id 1 expired
    if ((uint32_t)pvTimerGetTimerID(xTimer) == 1) {
        vTaskResume(printpacket);
        vTaskDelay(500 / portTICK_PERIOD_MS);
        vTaskSuspend(printpacket);
    }
}

void loraReceive(void *parameter) {
    while (1) {
        packetnum++;
        int pos1, pos2, pos3, pos4, pos5, pos6, pos7, pos8, pos9;
    }
}

```



```

// try to parse packet
int packetSize = LoRa.parsePacket();
if (packetSize) {
    // received a packet
    String LoRaData = LoRa.readString();

    pos1 = LoRaData.indexOf('!');
    pos2 = LoRaData.indexOf('@');
    pos3 = LoRaData.indexOf('#');
    pos4 = LoRaData.indexOf('$');
    pos5 = LoRaData.indexOf('%');
    pos6 = LoRaData.indexOf('^');
    pos7 = LoRaData.indexOf('&');
    pos8 = LoRaData.indexOf('*');
    pos9 = LoRaData.indexOf('/');

    // Extracting part of the whole string into each data value
    day_val = LoRaData.substring(0, pos1);
    month_val = LoRaData.substring(pos1 + 1, pos2);
    year_val = LoRaData.substring(pos2 + 1, pos3);
    hour_val = LoRaData.substring(pos3 + 1, pos4);
    minute_val = LoRaData.substring(pos4 + 1, pos5);
    second_val = LoRaData.substring(pos5 + 1, pos6);
    alt_km = LoRaData.substring(pos6 + 1, pos7);
    speed_kmph = LoRaData.substring(pos7 + 1, pos8);
    lat_val = LoRaData.substring(pos8 + 1, pos9);
    lng_val = LoRaData.substring(pos9 + 1, LoRaData.length());
}
}
}

```

```

void printAll(void *parameter) {
    while (1) {
        xSemaphoreTake(mutex, portMAX_DELAY);
        Serial.println();
        Serial.println("*****");
        Serial.println("RECEIVING packet:");
        Serial.println("*****");

        char arr1[32];
        sprintf(arr1, "DATE: %02d-%02d-%02d ", day_val.toInt(), month_val.toInt(), year_val.toInt());
        Serial.println(arr1);

        char arr2[32];
        sprintf(arr2, "TIME: %02d:%02d:%02d ", hour_val.toInt(), minute_val.toInt(), second_val.toInt());
        Serial.println(arr2);

        Serial.println("----- ALTITUDE/HEIGHT -----");
        Serial.print("ALTITUDE: ");
        Serial.print(alt_km);
        Serial.println(" km");
        Serial.println("----- SPEED -----");
        Serial.print("SPEED: ");
        Serial.print(speed_kmph);
        Serial.println(" kmph");
        Serial.println("----- COORDINATE -----");
        Serial.print("LATITUDE: ");
        Serial.println(lat_val);
        Serial.print("LONGITUDE: ");
        Serial.println(lng_val);
    }
}

```

```

    xSemaphoreGive(mutex);
    vTaskDelay(1000 / portTICK_PERIOD_MS);
}
}

void printPacketNum(void *parameters)
{
    while (1)
    {
        xSemaphoreTake(mutex, portMAX_DELAY);
        Serial.print("# of packets received: ");
        Serial.println(packetnum);
        xSemaphoreGive(mutex);
        vTaskDelay(1000 / portTICK_PERIOD_MS);
    }
}

void promptData(void *parameter) {
    while (1) {
        char input;
        if (xQueueReceive(input_queue, (void *)&input, 0) == pdTRUE)
        {
            if (input == '5') {
                char arr1[32];
                sprintf(arr1, "DATE: %02d-%02d-%02d ", day_val.toInt(), month_val.toInt(), year_val.toInt());
                xSemaphoreTake(mutex, portMAX_DELAY);
                Serial.println(arr1);
                xSemaphoreGive(mutex);
            }
            else if (input == '6') {

```

```

        char arr2[32];
        sprintf(arr2, "TIME: %02d:%02d:%02d ", hour_val.toInt(), minu
te_val.toInt(), second_val.toInt());
        xSemaphoreTake(mutex, portMAX_DELAY);
        Serial.println(arr2);
        xSemaphoreGive(mutex);
    }
    else if (input == '7') {
        xSemaphoreTake(mutex, portMAX_DELAY);
        Serial.println("----- ALTITUDE/HEIGHT -----");
        Serial.print("ALTITUDE: ");
        Serial.print(alt_km);
        Serial.println(" km");
        xSemaphoreGive(mutex);
    }
    else if (input == '8') {
        xSemaphoreTake(mutex, portMAX_DELAY);
        Serial.println("----- SPEED -----");
        Serial.print("SPEED: ");
        Serial.print(speed_kmph);
        Serial.println(" kmph");
        xSemaphoreGive(mutex);
    }
    else if (input == '9') {
        xSemaphoreTake(mutex, portMAX_DELAY);
        Serial.println("----- COORDINATE -----");
        Serial.print("LATITUDE: ");
        Serial.println(lat_val);
        Serial.print("LONGITUDE: ");
        Serial.println(lng_val);
        xSemaphoreGive(mutex);
    }

```

```

    }

    }

    vTaskDelay(500 / portTICK_PERIOD_MS);
}
}

void launchRocket(void *parameter) {
    while (1) {
        // Copy the parameter into a local variable (from countdown)
        int cd_num = *(int *)parameter;

        // Release the binary semaphore so that the creating function can
        finish

        xSemaphoreGive(bin_sem);

        if (auto_reload_timer == NULL) {
            Serial.println("Could not create the timer");
        } else {
            vTaskDelay(1000 / portTICK_PERIOD_MS);
            // Wait for the countdown and then starting the timers
            for (int i = cd_num; i > 0; i-- ) {
                printf("%d...\n", i);
                vTaskDelay(1000 / portTICK_PERIOD_MS);
            }
            Serial.println("Launched!!");
            // Start timer (max block time if command queue is full)
            xTimerStart(auto_reload_timer, portMAX_DELAY);
        }

        // This task is to only setup the timer and will not run endlessl
y

        vTaskDelete(NULL);
    }
}

```

```

    }
}

void setup() {
    long int countdown;

    Serial.begin(115200);
    while (!Serial);
    Serial.println("LoRa Receiver");

    //setup LoRa transceiver module
    LoRa.setPins(SS, RST, DIO0);
    while (!LoRa.begin(433E6)) {
        Serial.println(".");
        delay(500);
    }

    // Change sync word (0xF3) to match the receiver
    LoRa.setSyncWord(0xF3);
    Serial.println("LoRa Initializing OK!");

    // vTaskDelay(3000 / portTICK_PERIOD_MS);

    Serial.println("Set countdown time to launch rocket:");
    // Wait for input from Serial
    while (Serial.available() <= 0);
    // Read integer value
    countdown = Serial.parseInt();

    // Create Queue
    input_queue = xQueueCreate(msg_queue_len, sizeof(int));
    // Create binary semaphore before starting tasks

```

```

bin_sem = xSemaphoreCreateBinary();
// Create mutex
mutex = xSemaphoreCreateMutex();
// Create timer
auto_reload_timer = xTimerCreate(
    "Auto show # packet received", // Name of t
imer
    10000 / portTICK_PERIOD_MS, // Period of
timer (in ticks)
    pdTRUE, // Auto-
reload
    (void *)1, // Timer ID
    myTimerCallback); // Callback
function

xTaskCreatePinnedToCore(
    loraReceive,
    "Receive from TX",
    5000,
    NULL,
    1,
    NULL,
    app_cpu
);

xTaskCreate(
    printPacketNum,
    "Print # of packets received",
    1024,
    NULL,
    1,

```

```

    &printpacket
);
vTaskSuspend(printpacket);

xTaskCreate(
    launchRocket,
    "Start timer since launch",
    5000,
    (void *)&countdown,
    1,
    &launch
);
// Do nothing until binary semaphore has been returned
xSemaphoreTake(bin_sem, portMAX_DELAY);
vTaskSuspend(launch);

xTaskCreate(
    printAll,
    "Print All Data from TX",
    5000,
    NULL,
    1,
    &printall
);
vTaskSuspend(printall);

xTaskCreate(
    promptData,
    "Print Individual Data",
    1024,
    NULL,

```



```

    1,
    &promptdata
);
vTaskSuspend(promptdata);

Serial.println(F("Type in your command: "));
Serial.println("(1) to show all data in realtime");
Serial.println("(2) for prompting choice of data");
Serial.println("(3) to show # of packets received since");
Serial.println("(4) to initiate launching rocket");
Serial.println("(.) to EXIT");
}

void loop() {
    while (Serial.available()) {
        char input = Serial.read();
        xQueueSend(input_queue, (void *)&input, 10);
        if (input == '1') {
            vTaskResume(printall);
        }
        else if (input == '2') {
            vTaskResume(promptdata);
            Serial.println("(5) for DATE");
            Serial.println("(6) for TIME");
            Serial.println("(7) for ALTITUDE");
            Serial.println("(8) for SPEED");
            Serial.println("(9) for COORDINATE");
        }
        else if (input == '3') {
            vTaskResume(printpacket);
        }
    }
}

```

```

else if (input == '4') {
    vTaskResume(launch);
}
else if (input == '.') {
    Serial.println(F("Type in your command: "));
    Serial.println("(1) to show all data in realtime");
    Serial.println("(2) for prompting choice of data");
    Serial.println("(3) to show # of packets received since");
    Serial.println("(4) to initiate launching rocket");
    Serial.println("(.) to EXIT");
    vTaskSuspend(printall);
    vTaskSuspend(printpacket);
    vTaskSuspend(promptdata);
}
}
}

```