THE UNIVERSITY OF DANANG
UNIVERSITY OF SCIENCE AND TECHNOLOGY
Faculty of Advanced Science and Technology

Developing a Real-Time Bus Tracking System Based on Module GPS, STM32 and ESP32

This report is submitted as the final project of the Embedded System and Microcontroller Communication course in Embedded System and IoT (21ES) of Faculty of Science and Technology

Lecturer : Nguyễn Huỳnh Nhật Thương
Class : 21ES
Group : 3
Group members: Trần Minh Ánh (L) (123210104)
Lê Quang Kiệt (123210117)
Võ Đại Thuận (123210127)
Lê Gia Vinh (123210129)
Phạm Hữu Phước (123210123)
Phan Dương Gia Bảo (123210106)

Da Nang, May 2024

Acknowledgements

We would like to thank our project supervisor Nguyen Huynh Nhat Thuong for his continuous help, guidance and support throughout the course and the final project. We would also like to thank teacher assistant Nguyen Quang Phuong for his valuable assistance about the project. Lastly, We’d like to thank our friends, especially Nguyen Duc Hung, for their support throughout our running project time.

Table of Contents

Table of Acronyms

Acronym	Definition
ES&MC	Embedded System and Microcontroller Communication
GPS	Global Positioning System
NMEA	National Marine Electronics Association
GGA	Global Positioning System Fixed Data
RMC	Recommended Minimum Specific GNSS Data
UART	Universal synchronous receiver transmitter
I2C	Inter-Integrated circuit interface
PWR	Power control
FDD	Function decomposition diagram
USB	Universal serial bus

Table of Tables

1. Introduction

1.1. Background

The GPS is a satellite-based navigation system made up of a network of 24 satellites placed into orbit by the U.S. Department of Defense (Ali). GPS was originally intended for military applications, but in the 1980s, the government made the system available for civilian use. GPS is the first and most commonly used navigation system.

GPS works in any weather conditions, anywhere in the world, 24 hours a day. There are no subscription fees or setup charges to use GPS.

1.2. Aims and objectives

The aim of the project was to apply our knowledge from ES&MC course to develop a system using STM32.

To summarize, the objective of the project were to:

- Develop a real-time bus tracking system (provide bus location and routes to users)
- Using UART to communicate between GPS module-STM32, STM32-ESP32
- Using I2C to control LCD 1602
- Energy saving when designing circuit and mode
- Apply some technique to solve the GPS data (latitude, longitude, date, time)

1.3. Project overview

Real-time bus tracking system is an application that provides essential information about buses in operation. The system uses a GPS module (installed in the bus) to track the bus location and provide real-time data (bus location) to users through a web or mobile application. This helps users to conveniently track bus routes and schedules

Figure 1: The real GPS device.

Figure 2: The real-time web for bus

The hardware designs and source code were published to Github. Link to videos demonstrating the project can be found in the footnotes.

1.4. GPS module (LC76G)

Introduce module here,

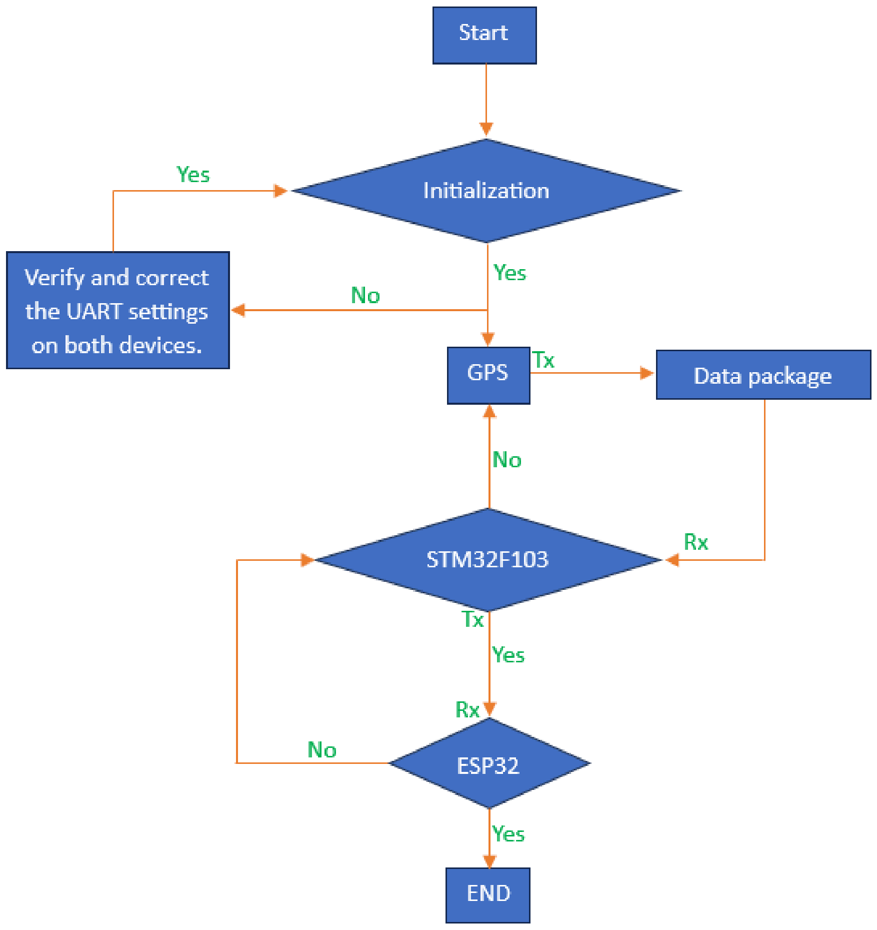
2. Design and Implementation

In the simplest case, GPS is applied to determine the bus's location.

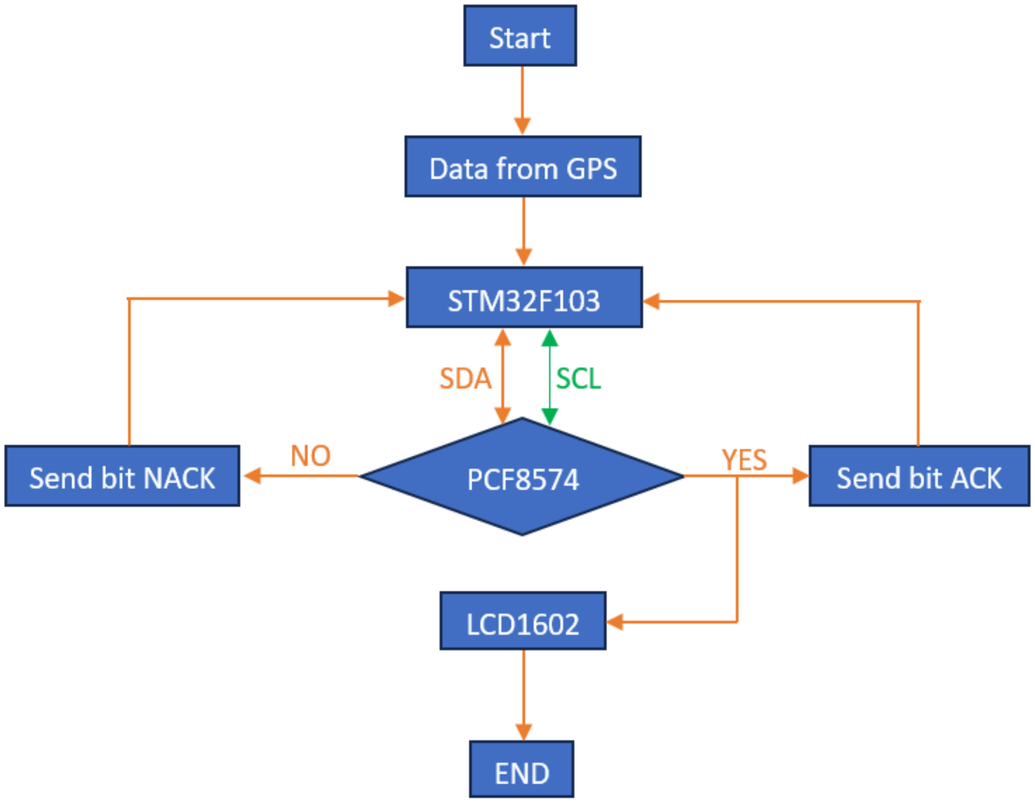
2.1. Function Decomposition
2.2. Flow chart
2.3. Circuit design

2.1. UART and I2C

- UART (Universal Asynchronous Receiver-Transmitter) is a hardware communications protocol that uses asynchronous serial communication and has configurable speeds.
- UART communication pseudocode diagram:
  (GPS → STM32F103 → ESP32)



- I2C (Inter-Integrated Circuit) is a synchronous, multi controller/multi-target, single-ended, half-duplex serial communication bus used for attaching lower-speed peripheral ICs to processors and microcontrollers in short-distance, intra-board communication.
- I2C communication pseudocode diagram:
  (STM32 - LCD1602)



2.2. ESP

ESP is a wifi module that allows processors to connect to internet networks, specifically, stm32 processor is linked with GPS module and ESP32 as a peripherals that GPS send continuous data into processor, and ESP32 delivery data pack into firebase.

Beginning, ESP32 will checks wifi module

```
Serial.begin(115200);
WiFi.begin(WIFI_SSID, WIFI_PASSWORD);
Serial.print("Connecting to Wi-Fi");
while (WiFi.status() != WL_CONNECTED) {
  Serial.print(".");
  delay(300);
}
```

```
 if (Firebase.signUp(&config, &auth, "", "")) {
    Serial.println("Sign-up OK");
    signupOK = true;
  } else {
    Serial.printf("Sign-up Error: %s\n", config.signer.signupError.message.c_str());
  }
```

If ESP32 already connects to the internet network, then it checks firebase access rights.

After two authority confirm complete, the esp will receive the data pack and assign it into variable by ‘scanf ‘

```
if (sscanf(test, "%f %f", &latitude, &longitude) == 2) {
      Serial.print("Latitude: ");
      Serial.println(latitude);
      Serial.print("Longitude: ");
      Serial.println(longitude);
    } else {
      Serial.println("Error parsing string");
    }
```

Finally, esp send data into firebase

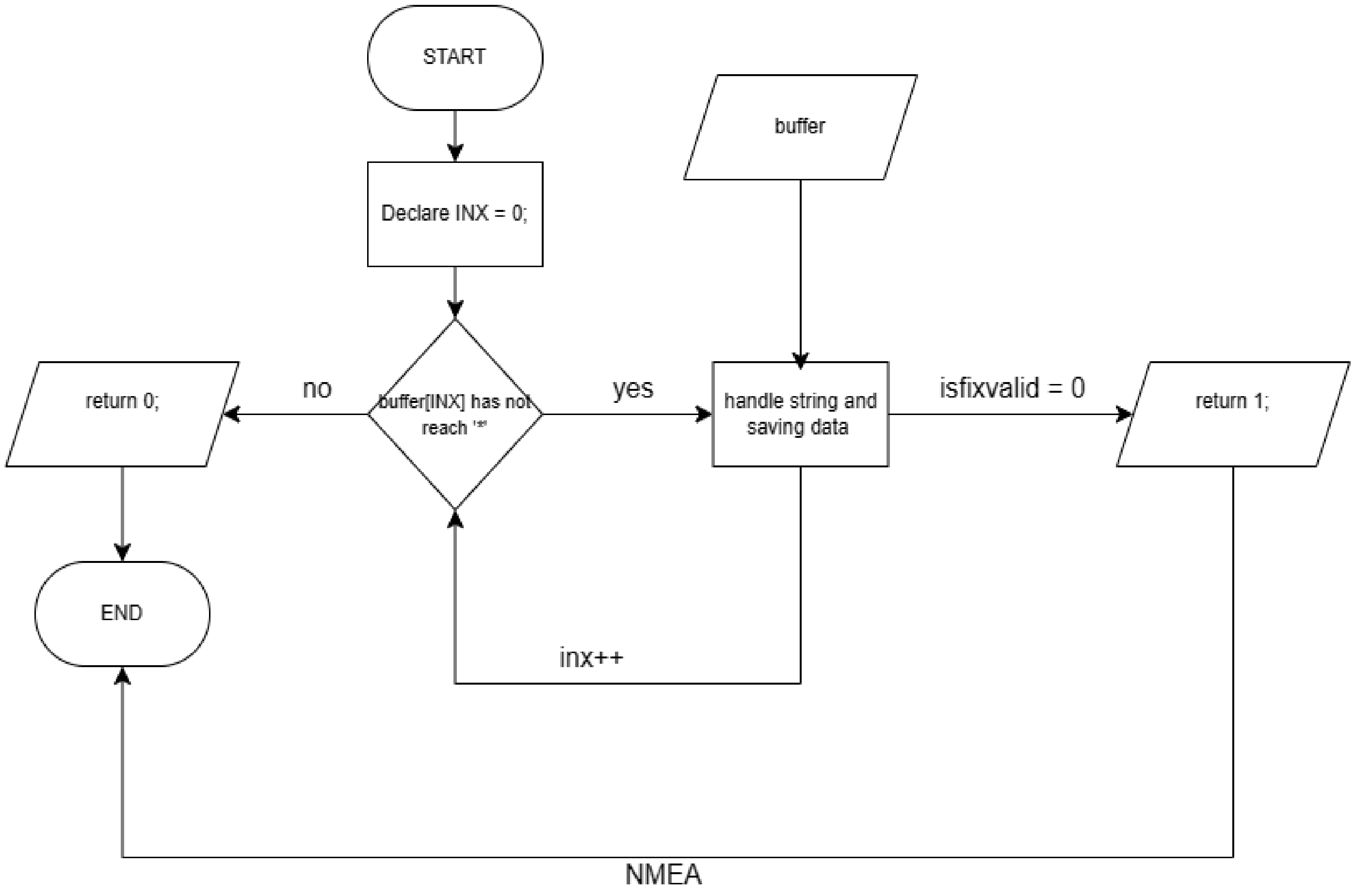
```
if (signupOK) {
      // Send data to Firebase
      if (!Firebase.RTDB.setFloat(&fbdo, "/GPS_Data/Latitude", latitude)) {
        Serial.print("Error sending latitude data: ");
        Serial.println(fbdo.errorReason());
      }

      if (!Firebase.RTDB.setFloat(&fbdo, "/GPS_Data/Longitude", longitude)) {
        Serial.print("Error sending longitude data: ");
        Serial.println(fbdo.errorReason());
      }
    }
  }
```

2.3. NMEA

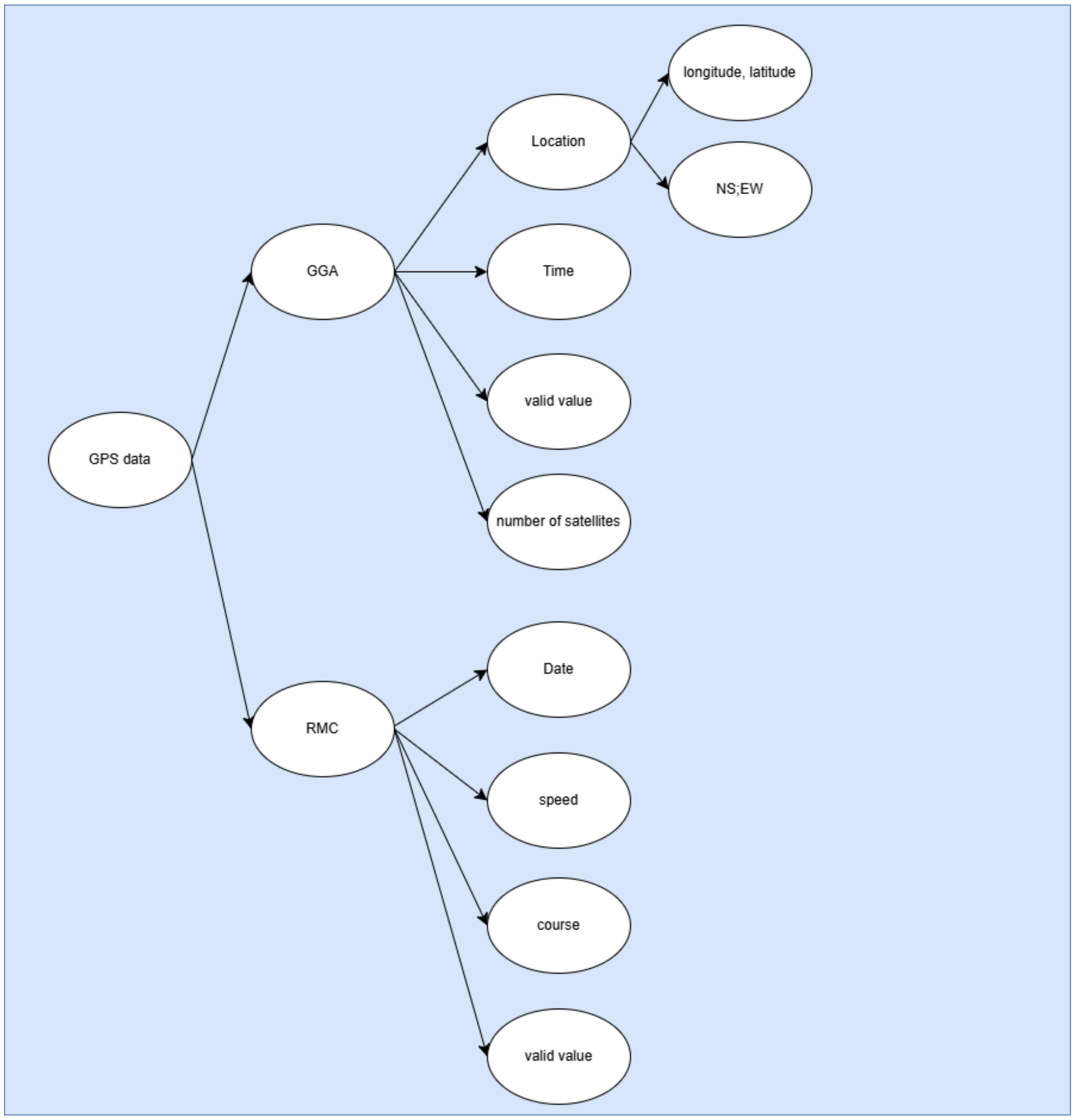
The NMEA is a messaging protocol standard. These messages consist of sentences with data transmitted, with each sentence containing various bits of data that are separated by commas.
GPS spawns many different protocols, and depends on <TalkerID>, purifying necessary strings and working on it. In this report, “RMC” and “GGA” are the main keywords.

Format:
$<TalkerID>GGA,<Timestamp>,<Lat>,<N/S>,<Long>,<E/W>,<GPSQual>,<Sats>,<HDOP>,<Alt>,<AltVal>,<GeoSep>,<GeoVal>,<DGPSAge>,<DGPSRef><checksum><CR><LF>*
$<TalkerID>RMC,<Timestamp>,<Status>,<Lat>,<N/S>,<Long>,<E/W>,<SOG>,<COG>,<Date>,<MagVar>,<MagVarDir>,<mode>,<NavStatus><checksum><CR><LF>*

After extracting GGA and RMC strings, load into stm32 processors which handle data and transmit them to peripherals.
NMEA’s processing diagram:



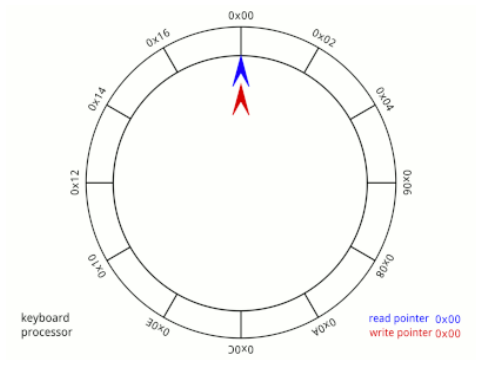
NMEA data type: `GPSdata` is a user-defined data type, which contains two sub-data types, `RMC` and `GGA`, which are also user-defined data types. These sub-data types contain subfields as described in the table below. The purpose is to categorize and store data in appropriate variables.
Also GGA and RMC protocols contain the common data such as longitude, latitude and many others, but RMC string has: speed, course,... GGA is not.



2.4. Circular buffer / Process the input signal

- When we receive data from modules that provide signals continuously such as a GPS module, you must have a large database to store data. Information processing facilities will probably have several facilities to store that large amount of data. However, in our project, it is only designed to be easy to carry when traveling.

- With limited memory requirements, we choose the Ring Buffer / Circular Buffer to process the data sent by the GPS module. Ring Buffer doesn't need infinite amounts of memory, since older entries get overridden automatically. The "challenge" here is that you need to find a suitable size for your use case. However, its necessary needs are only at 256 or 512.
- Circular buffers are popular for serial data streams in embedded systems. Microcontrollers often have a UART to handle a serial byte coming in, these need to be stored in order and dealt with later.
    + Upon the receipt of a byte the UART can generate an interrupt to which the software responds by quickly taking the received byte and shoves it onto the end of the buffer.
    + Background software routines can then regularly check if the butter has anything in it yet and empty it as required.
- As the Circular Buffer size can be defined pre-compilation the size is then limited. This helps improve space efficiency and should eliminate memory corruption as a trade off to how many bytes can be received before data starts to become lost.



3. Testing and results

All the experiments and processes will be presented below.

3.1. Circuit Testing

Typically,

3.2. System testing

Another improvement

3.3. Something

Đâsdas

4. Conclusion

4.1. Results

GPS can determine the location of the person holding the device, but its accuracy is still not very high, with a deviation within a 10-meter radius. The ability to capture the location while the device is moving can provide fairly accurate positioning and can map the device's route on a map.

4.2. Potential

This project demonstrates that integrating GPS into locating the nearest bus and estimating the time for the bus to reach the stop is entirely feasible.

4.3. Something

As explained

4.4. Something

We repeat those assignments

5. Appendix

5.1. Appendix A - Work breakdown structure

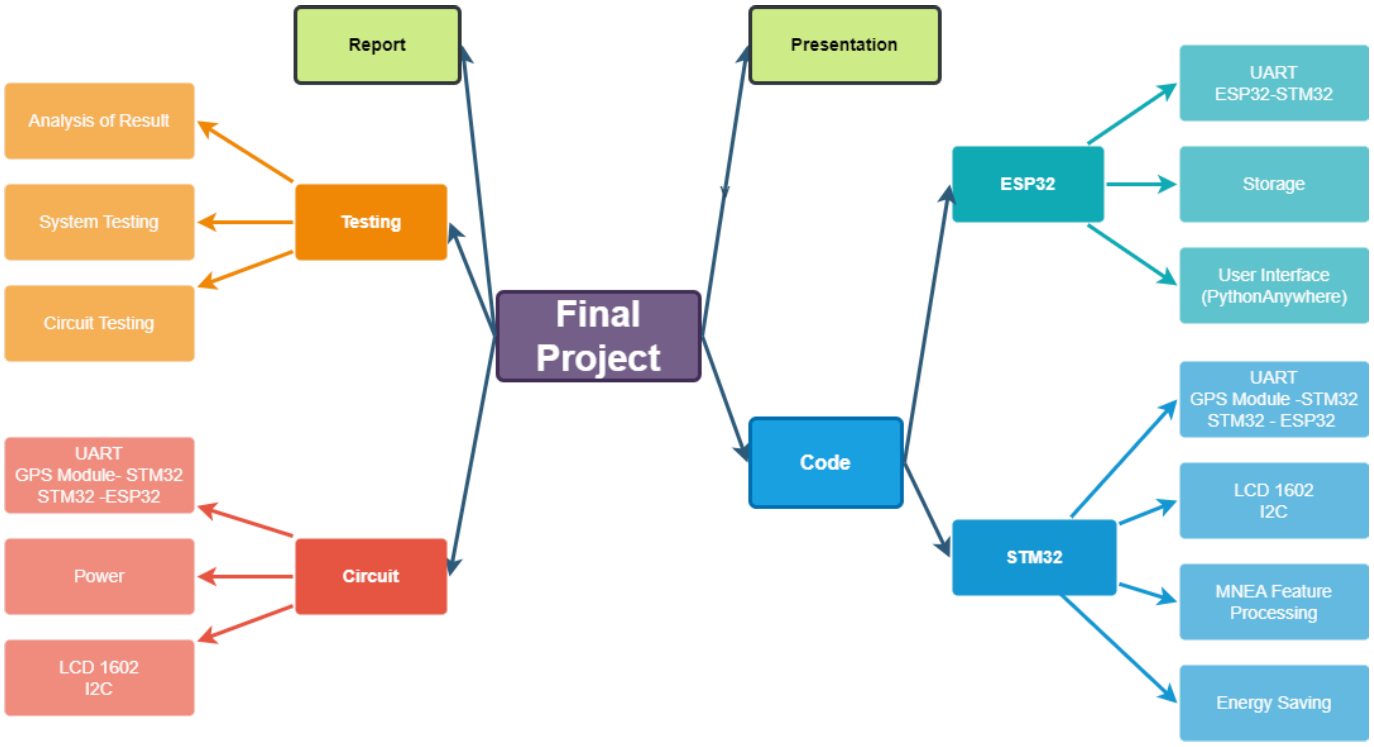


Figure: Work Breakdown Structure

5.2. Appendix B - Bill of materials

Below is the table of resources and equipment that were used during the project.

Table 1: Table of resources and equipment

Resource/Equipment	***Quantity***	***Cost (VND)***	***Notes***
Total circuit cost			
STM32F103C8T6, Blink	*1*	*250.000*	
ESP32	*1*	*120.000*	
USB TTL Converter	*1*	*20.000*	*CH340*
Module GPS	*1*	*430.000*	*LC76G*

LCD 16x2 and I2C	*1*	*45.000*	
Battery	*1*	*150.000*	*7500mAh, 5.2V 1A*
Voltage regulator	*1*		
Breadboard	*1*	*28.000*	*16.5x5.5cm*
Wire	*-*	*-*	*Kinds to be assessed*
Micro USB	*2*	*20.000*	
Software			
PC	*-*	*-*	*Install Python 3.10*
Microcontrollers	*-*	*-*	*STM32Cube, Arduino IDE*

6. Bibliography