Master of Science HES-SO in Engineering
Av. de Provence 6
CH-1007 Lausanne

Master of Science HES-SO in Engineering

Major : Computer Science (CS)

Telemetry for the Formula Student - Optimizations

By

Sylvestre van Kappel

Under the supervision of
Prof. Medard Rieder
HES-SO Valais Wallis

Sion, HES-SO//Valais-Wallis, 31.05.2024

Accepted by the HES SO//Master (Switzerland, Lausanne) on the proposal of

Prof. Medard Rieder, supervisor for the in-depth project

Sion, 31.05.2024

Prof. Medard Rieder	Prof. Nabil Abdennadher
Supervisor	Program director

Contents

List of Figures

List of Tables

List of Listings

1 Introduction

1.1 Formula Student

Formula Student is an international student engineering competition. Teams from all over the world design and build a small-scale Formula-style racing car. During the competitions, the vehicles are judged on static events: Engineering Design, Cost and Manufacturing, Business Presentation, Lap Time Simulation, and Technical Inspection. The vehicle must comply with the rules [1] for the technical inspection. If the car passes the technical assessment, it can participate in the dynamic events: Skidpad, 1 km autocross/sprint, 75 m acceleration, and 22 km endurance.

1.2 Valais-Wallis Racing Team

The Valais-Wallis Racing team [2] is the Formula Student team of HES-SO Valais Wallis. It was founded in the spring of 2022 by a group of students. The team's inaugural car participated in the summer 2023 races, and the telemetry system developed in this thesis will be utilized in the subsequent car for the summer 2024 races. Telemetry was not integrated into the initial car; however, a telemetry system was designed during the summer of 2023.

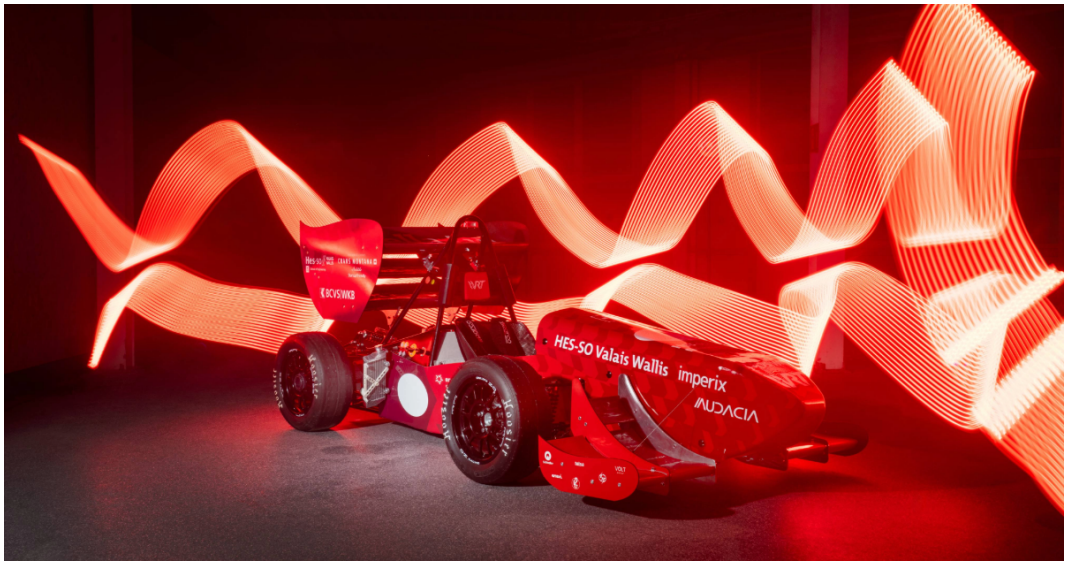

Figure 1.1: Valais-Wallis Racing Team's Car

1.3 Objectives

Telemetry is a technology that enables remote measurement and monitoring. This technology is interesting for a race vehicle as it allows live readings from the car's sensors to be monitored directly from the side of the track. With such a system, the data from all the sensors are easily accessible, and the engineers can adjust the car's parameters during the test sessions to increase the car's performance. A telemetry system is also helpful in improving the driver's skills, providing measurements such as GPS, speed, pedal level, steering angle, etc. Direct visualization of measurements also allows problems to be identified before they cause an accident or damage to the car.

This project aims to enhance the existing system by incorporating additional functionalities and improving its performance. The objectives of the project are as follows :

- **Addition of a Configurable Filter for the CAN Bus:** The objective of this section is to implement a filter on the CAN input to allow the passage of other messages without compromising the performance of the telemetry system.
- **Telemetry System Control via CAN Bus:** Currently, recording is initiated and stopped using a button connected to the system. The objective of this section is to add the capability to control these functions via the CAN bus.
- **Integration of Time and Date:** The objective of this section is to replace the current timestamp (relative time since recording started) with absolute time.
- **System Optimization to Enable the Recording of More Data at a Higher Frequency:** The goal of this section is to push the limits of the number of CAN messages the system can handle and increase the recording frequency on the SD card.

The first two points will be implemented on the existing hardware. This is because the VRT team will need these modifications for the summer 2024 races, and these optimisations do not require new hardware. For the last two points, the same hardware is reused, but it is duplicated in order to have one device for the transmission and the other for the recording.

1.4 Structure of this Report

This report is in 4 parts.The first part is a general overview of the system. The second part describes the small hardware modifications that were made, and the third part describes the software optimisations. The last part describes the various tests that were carried out.

1.5 Git

All the files referring to the project (codes, altium PCB project ...) and the first release of the project are available on a git repository on the following link :

https://github.com/svankappel/Telemetry-for-the-Formula-Student

1.6 Existing Hardware

The following image shows the existing hardware :

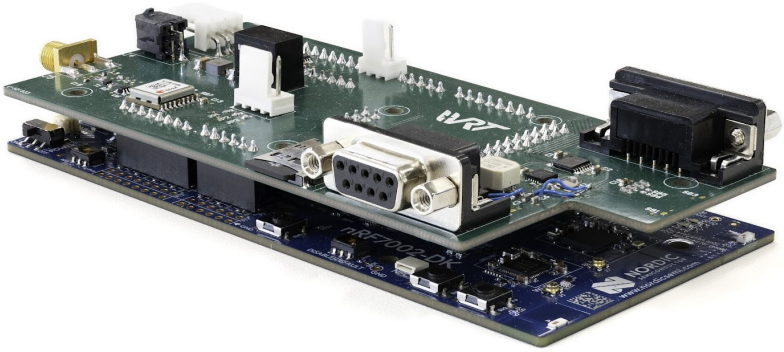

Figure 1.2: Current Hardware

2 System Overview

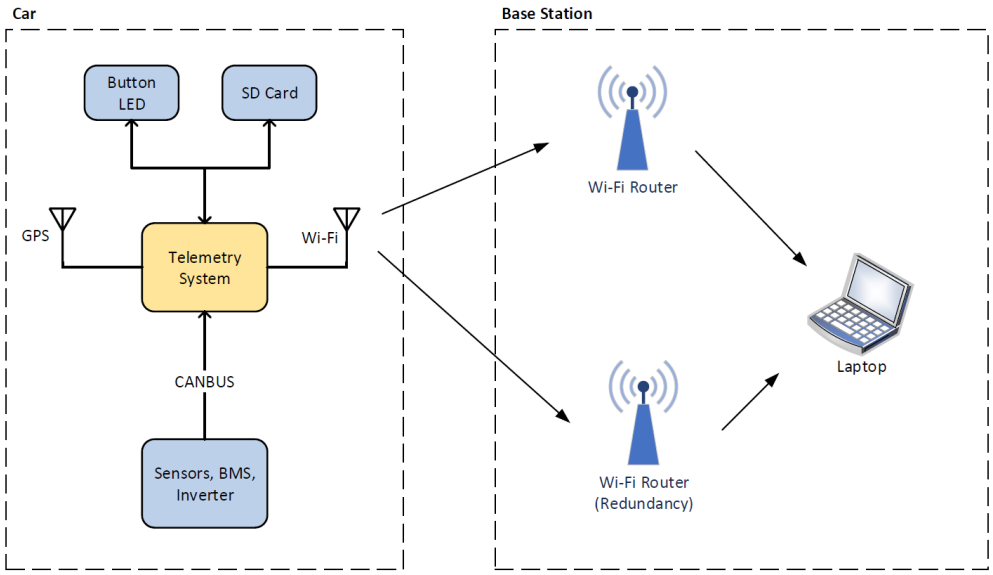

Figure 2.1: System Overview

This diagram shows the general functioning of the telemetry system. The data coming from the sensors are transmitted by Wi-Fi to the base station.

2.1 CAN Bus

The CAN bus is a widely used communication protocol in automotive applications. It was developed in the 1980s by Bosch to allow reliable communication between ECUs in vehicles. The CAN bus protocol is robust and fast. It can handle multiple devices on a single network while maintaining reliability and data integrity.

The CAN bus connects all the different devices of the car that need to communicate. The messages contain various parameters, such as engine speed, pedal status, temperature of the different components, etc. The BMS and the inverter in the actual car include a CAN interface. These two components provide some measurements such as voltage and current of the High Voltage system, power, temperatures, and angular speed. The car will also include more sensors also connected to the CAN bus.

The telemetry system is connected to the CAN bus and retransmits the data from the sensors.

2.2 SD Card

The system includes an SD card. The SD card has two functions.

The first function is to hold a configuration file. It is a JSON file that contains all the information the system needs to work correctly. The configuration file contains the settings for every sensor. A new sensor can be easily added by putting its CAN settings in the configuration file.

The second function is to register the measurements made by the sensors. When the button is pressed, the recording is activated, and the LED on the button blinks. When the button is pressed again, the recording is stopped. The data on the SD card provide a backup in case the transmission is interrupted. The data from all sensors are recorded at a high frequency. These data are saved in CSV files.

2.3 Wi-Fi Transmission

The Wi-Fi transmission includes an optional redundancy. When it is activated, the system tries to connect to the first router. If it cannot connect, it tries with the second router. It then retries with the first if the second is unreachable, etc.

The settings of the Wi-Fi connection (SSID, password) are set in the configuration file. The system can use any standard Wi-Fi router. It employs the WPA2 security protocol but can be easily migrated to WPA3.

The data of only a selected number of sensors shall be live transmitted to the base station. A boolean field in the configuration file indicates whether the data of each sensor shall be transmitted or not. The transmission frequency is also defined in the configuration file. The defined frequency is lower than the recording frequency.

2.4 GPS

The telemetry device includes a GPS. It provides the GPS coordinates and the speed of the car. As for the sensors, the data are saved in the CSV files and can also be sent by the Wi-Fi transmission.

2.5 Front End

The software on the base station's PC displays the data sent by the telemetry device. The data saved on the SD card can also be loaded on the software. This software was developed by a student from the Business Information Technology program.

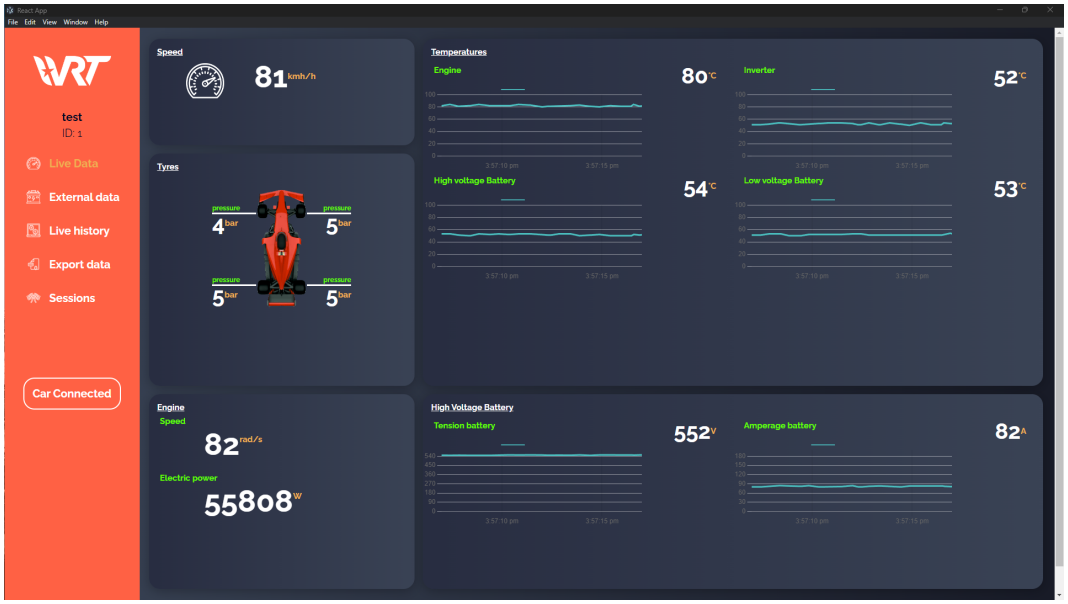

Figure 2.2: Telemetry System Display

The software can be downloaded at the following link :

https://vrt.simonbeaud.ch/

2.6 Optimizations

The optimizations do not alter the general system overview. The system continues to function in the same manner. The CAN filter and CAN button functionalities merely introduce some configuration adjustments in the configuration file. The inclusion of time and date provides additional data in the CSV recording file.

3 Hardware

3.1 Existing System

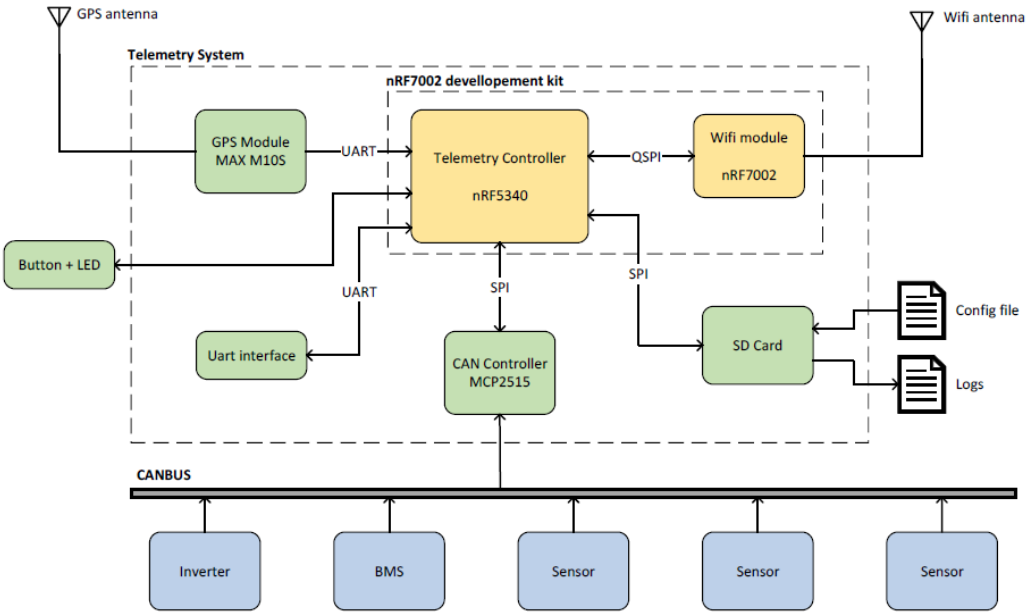

Figure 3.1: Bloc Diagram - Current System

This diagram shows the general architecture of the telemetry system's onboard device. The CAN Filter and the CAN button are implemented on this version on the hardware (and also on the new version).

The main controller (nRF5340) is the central element of the system.

The CAN controller is connected to the nRF5340 via an SPI bus. On the other side, it is connected to the CAN bus to communicate with the BMS, the inverter, and all the sensors.

Another peripheral is the GPS module. It is connected to the nRF5340 via a UART communication. The GPS module uses an external antenna because it needs to be pointed to the sky, and the telemetry system will be placed in a case in the car.

The microSD card is connected to the nRF5340 via an SPI bus. The microSD card serves to store the measurements of the telemetry system. The memory card also holds a configuration file.

The nRF7002 chip handles Wi-Fi communication. It is connected to the nRF5340 via a QSPI bus. As for the GPS module, the Wi-Fi module uses an external antenna to improve the transmission.

The system also includes a button with an intern LED. The button permits to start and stop the recording of the logs, and the LED indicates the system's status. The button will be placed on the dashboard of the car.

The last peripheral is a UART interface. The telemetry system does not use it. It is just there for future system improvements (to connect an onboard computer, for example).

3.2 New System

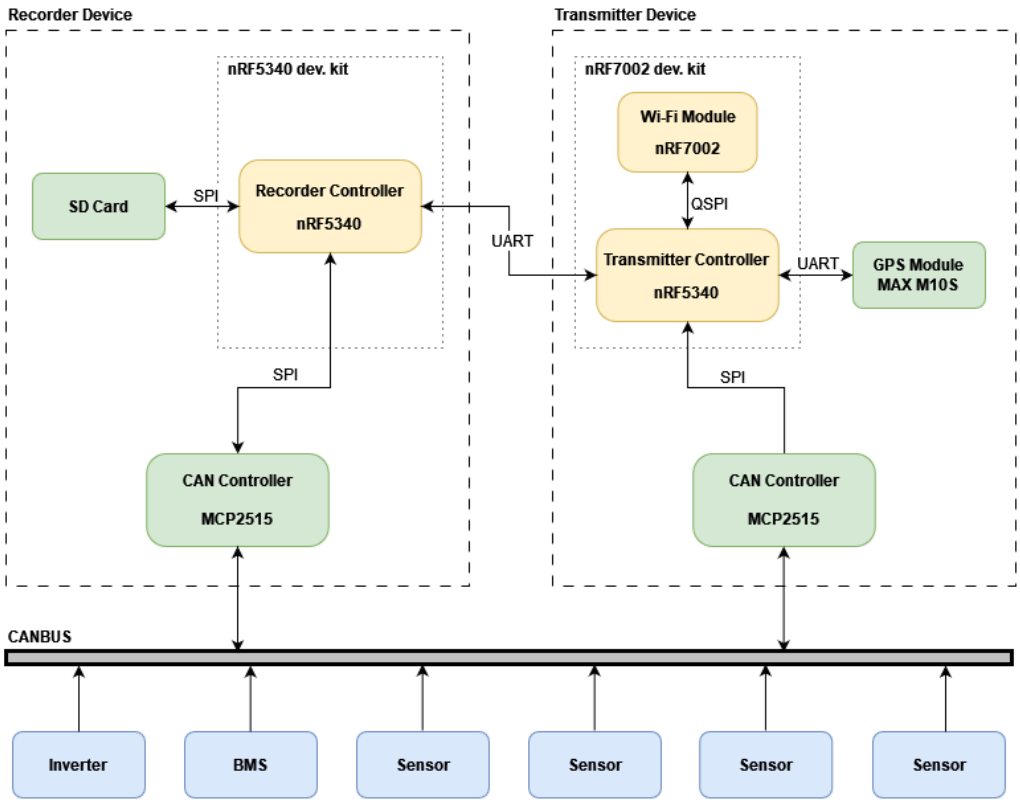

Figure 3.2: Bloc Diagram - New System

The new hardware is identical to the existing version, but the system has been duplicated, with only the components required by each respective device being mounted on the PCB. For the recording device, the nRF7002DK was replaced by a nRF5340DK as it does not require Wi-Fi. The two devices are connected together via a UART bus and they both have their own CAN Controller.

The following image shows the new hardware:

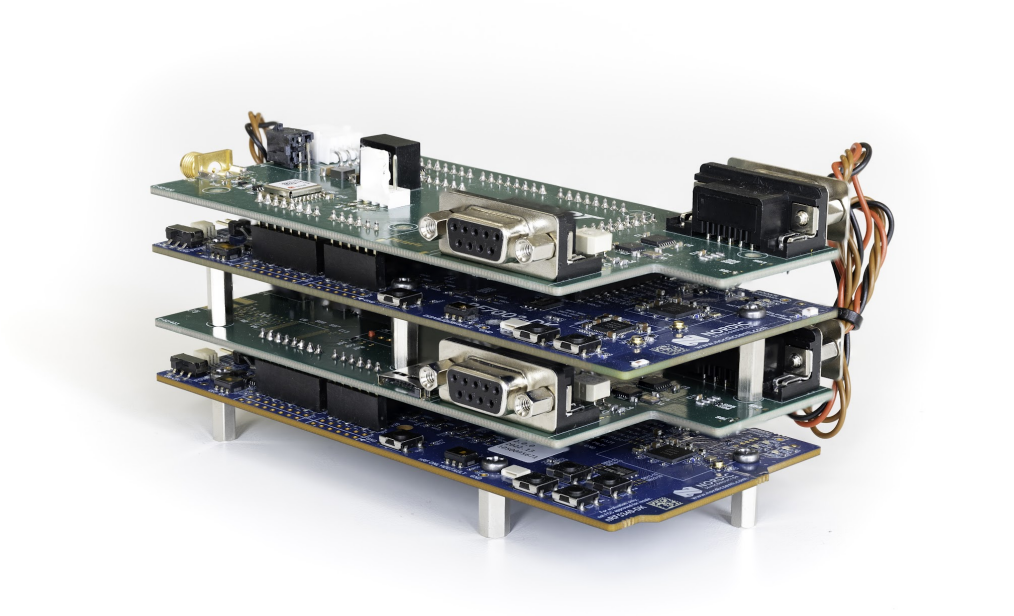

Figure 3.3: New Hardware

4 Software Development

4.1 Existing System Design

The software implemented on the nRF5340 uses the Zephyr RTOS. Zephyr RTOS is a compact operating system designed for devices with limited resources. Nordic's advanced hardware, renowned for its performance and efficiency, fits absolutely with Zephyr's lightweight and modular design.

The software is implemented on the application processor of the nRF5340. It was developed with the nRF Connect SDK v2.3.0.

The software implements the following tasks working in parallel:

- Wi-Fi Station
- UDP Client
- Data Sender
- Data Logger
- GPS Controller
- CAN Controller
- Button Manager
- LED Controller

The *main* function starts the Button Manager and the LED Controller first. Then the configuration file is read, and all the other tasks are started if the reading of the file is successful.

The design includes two buffers. The first is the sensor buffer. It contains the value and the configuration of every sensor present in the configuration file. The CAN Controller writes the values received by the sensors on this buffer. The second buffer is the GPS buffer. It works like the sensor buffer but takes its data from the GPS Controller.

The Data Sender periodically reads the sensor and GPS buffers and creates a JSON message containing the sensor and GPS values configured to be sent on the live transmission. Then it puts it in the UDP queue.

The UDP Client reads the UDP queue and sends the messages to the base station via the Wi-Fi using a UDP socket. The UDP protocol was chosen because it is faster than TCP, and the transmission does not need to be 100% reliable because the messages are periodically re-transmitted. Consequently, it is not a problem if sometimes a transmission contains an error.

When the button is pressed, the Data Logger creates a new CSV file on the SD card. It then periodically reads the sensor and GPS buffers and writes a new line on the CSV file with the read values. When the button is pressed again, the recording is stopped. The LED on the button is blinking when the Data Logger is recording.

The Wi-Fi Station manages the Wi-Fi connection.

The following diagram shows the system's general architecture:

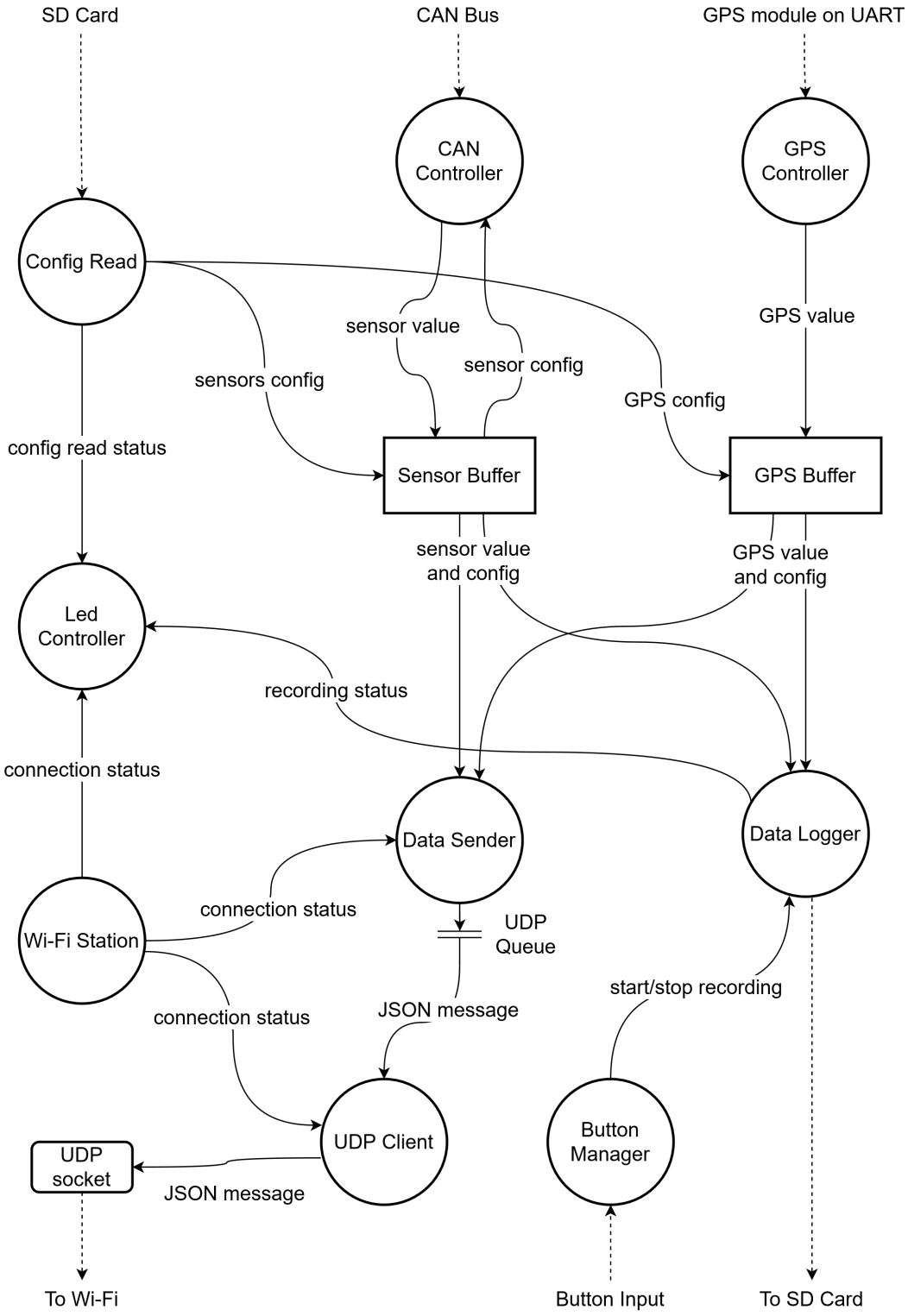

Figure 4.1: Data-Flow Diagram

4.2 CAN Filter

The first optimization added to the system is the CAN filter. The Zephyr CANBUS API [3] allows a CAN filter to be set. Messages not intended for the telemetry system are also sent over the CANBUS. The existing system also processes these messages, reads them, and then ignores them. This takes some time and affects the performance of the system. Tests have shown that the system cannot handle more than 200 messages per second to function reliably. By adding a CAN filter, a range can be defined, and message IDs that are not in the range will be ignored without affecting the performance of the system.

The CAN filter is set during the initialization of the CAN device in the CAN controller component. It takes two arguments: an ID and a mask. The filter works as follows:

A message is accepted if the following condition is true :

(filter ID & mask) == (message CAN ID & mask)

For instance, with the following configuration, only messages with a CAN ID in the specified range will be considered:

ID = 0xAA = 0b10101010

Mask = 0xF0 = 0b11110000

Messages taken into consideration are 0xA0, 0xA1, ..., 0xAF.

This allows creating ranges of CAN IDs that are designated for the telemetry system.

As the system must be configurable, the ID and mask have been added to the configuration file of the system. The configuration file is read when the program starts, and the values are stored in a structure that is then accessed by all the components of the system.

```
"CANFilter":
{
    "id":"0x0",
    "mask":"0x0"
}
```

Listing 4.1: Configuration of the CAN filter

4.3 CAN Button

On the existing system, the only way to start and stop the recording is to push the physical button. An LED on the button indicates if the system is currently recording.

The goal of this part is to start and stop the recording with a message on the CAN bus. The telemetry system must also emit a message on the CANBUS to indicate if it is currently recording or not, to replace the LED. This allows the button and the LED to be connected via the CANBUS, avoiding additional wiring. With this solution, the telemetry system will also be controllable by the on-board computer. The system must also include an option to automatically start the recording on boot.

To achieve this, several configurations have been added to the configuration file. The first is a boolean to enable recording on startup:

```
"RecordOnStart":false,
```

Listing 4.2: Configuration of the Record-on-Start

Next, there are the button configurations. There are two configurations: one to start and one to stop the recording. These configurations require a CAN ID, the length of the message, the index of the byte that contains the information in the message, the match value (the value the message must correspond to in order to start/stop the recording), and finally a mask for this match, which allows using only a single bit so the same byte can transmit other information:

```
"CANButton":
{
    "StartLog":
    {
        "CanID":"0x010",
        "dlc":8,
        "index":7,
        "match":"0x01",
        "mask":"0x01"
    },
    "StopLog":
    {
        "CanID":"0x010",
        "dlc":8,
        "index":7,
        "match":"0x00",
        "mask":"0x01"
    }
},
```

Listing 4.3: Configuration of the CAN Button

There is also a configuration for the "LED". A message indicates if the device is recording. For this, the system will emit a 0x01 when it is recording and a 0x00 when it is not recording, on the CAN ID specified in the configuration:

```
"CANLed":
{
    "CanID":"0x11"
},
```

Listing 4.4: Configuration of the CAN Button

To send the LED message, an event was added to the Data Logger component. When the recording successfully starts, a function is called in the CAN Controller. A callback method with a function pointer is used to pass the functions to the Data Logger. In the CAN Controller, the message is formed with the CAN ID configured in the configuration file, and the message is sent.

For the CAN Button, when a message arrives on the CAN, it is put in a queue that the CAN Controller component reads. If the CAN ID matches the ID of the start or stop message, the same function that is called by the button controller is invoked. This creates a work item [4] that then calls the start or stop function in the Data Logger component.

The following diagram shows the sequence of the function called when a start message arrives on the CANBUS :

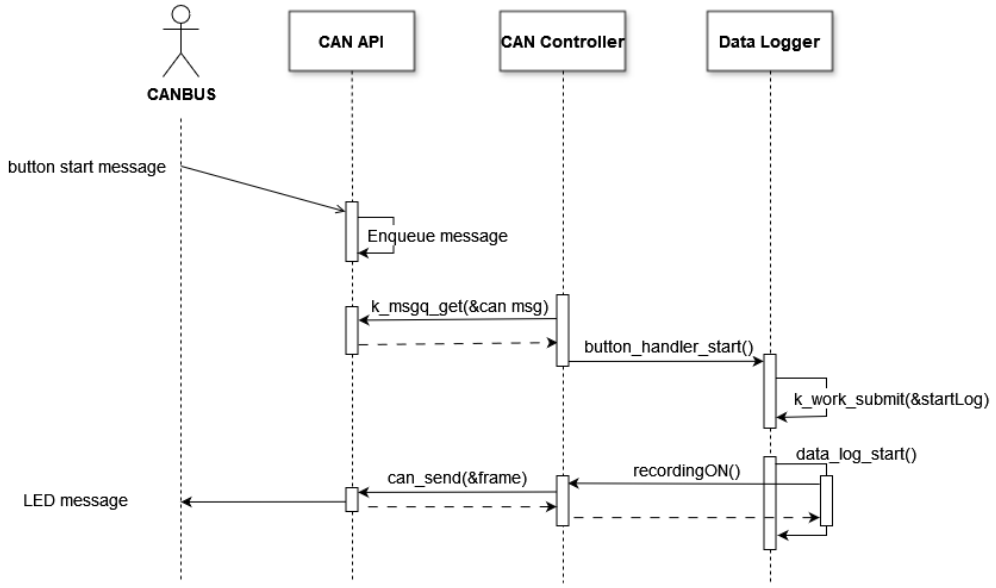

Figure 4.2: CAN Button Sequence

4.4 Time and Date

The aim of this chapter is to add a time and a date to the recordings. For this, the system needs to know the current time. There are three options to achieve this:

- **Remote PC:** The first option is to request the time from the remote PC via Wi-Fi communication. The issue with this approach is that the system may be used without Wi-Fi, in which case it would not be able to provide the time.

- **Real-Time Clock (RTC):** An RTC is an integrated circuit that provides the time. It must be powered by a small battery to retain the time when the system is powered off. The drawback is that hardware modifications would be necessary, and there was not enough time for such modifications in this project.

- **GPS:** The system's GPS module also provides the time and date. The only problem is that the GPS signal can be weak indoors. To ensure this would not cause problems, tests were conducted in the Valais-Wallis Racing Team's workshop. The results indicated that the GPS signal inside the workshop was strong enough to obtain the time and position as quickly as outside.

The GPS module works with the NMEA protocol [5]. Frames are transmitted to the nRF5340 via a UART bus. The frame that contains the date and time is the RMC frame. The GPS module sends data every second. On reception, it triggers an interrupt. During the interrupt routine, the message is pushed to a queue. The GPS Controller component reads from the queue, parses the NMEA frame, and writes the time and date data to the GPS buffer. When a recording is started in the Datalogger component, the time and date are read on the GPS buffer and written to the first line of the CSV file.

The CSV file uses the following structure:

Date:	31-05-2024	Time:	09:45:12	
Timestamp [ms]	**Sensor1**	**Sensor2**	**Sensor3**	**Sensor4**
50	10	34567	156	332
100	12	30678	876	333
150	15	22344	459	335
...	...	...	...	...

Table 4.1: CSV File Structure

The following diagram show the sequence of the GPS data from the GPS module to the CSV file:

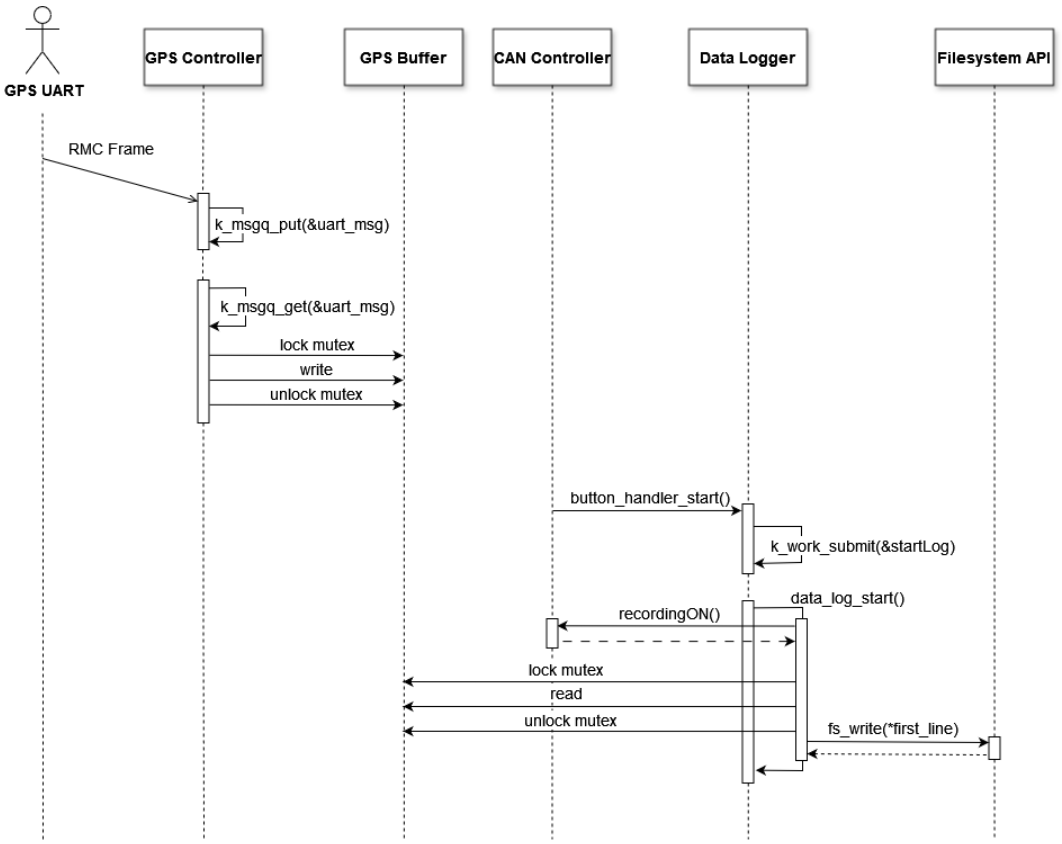

Figure 4.3: GPS Time and Date Data Sequence

4.5 System Optimization

The aim of this chapter is to optimise the performance of the system. Currently the system can process 200 messages per second on the CAN bus. If the message rate exceeds this limit, data loss can occur because the processor does not have enough time to process each message and write the data to the SD card. In addition, the processor must simultaneously handle Wi-Fi connectivity, data transfer and GPS tasks. These parallel operations fully utilise the processor's capacity, resulting in incoming CAN messages arriving faster than they can be processed. To prevent data loss, the CAN controller uses a FIFO buffer to temporarily store messages until they are processed. However, if the buffer fills up faster than it can be emptied, any further incoming data will be lost once the buffer is full.

It's important to note that the 200 messages per second limit may vary depending on other system configurations, such as the frequency of writing to the SD card and the amount of data written to the SD card. These factors have a significant impact on overall system performance and its ability to handle incoming messages without data loss.

There are two ways to optimise the performance of the system. The first is to reduce the time taken by tasks that are frequently called, and the second is to reduce the number of tasks that are executed by the processor by splitting the system between two processors. The split was made so that one processor was dedicated to recording on the SD card and the other to transferring the data.

4.5.1 Optimization on the existing device

The existing system is already relatively straightforward, with limited opportunities for optimization. The only significant non-essential time-consuming process is the periodic synchronisation of the SD card [6]. When the function to write to the SD card is called, the data is not necessarily written directly to the SD card. Instead, the data is initially written to a cache memory, which is then copied to the SD card when a significant amount of data has accumulated. In the current system, the cache is periodically flushed in order to save data in the event that the user fails to stop recording before shutting down the system. However, in the new car, the recording is controlled by the car's embedded computer rather than a human operator. Consequently, this situation will not occur, and the periodic synchronization can be eliminated.

The data is written to the sd card with the following sequence:

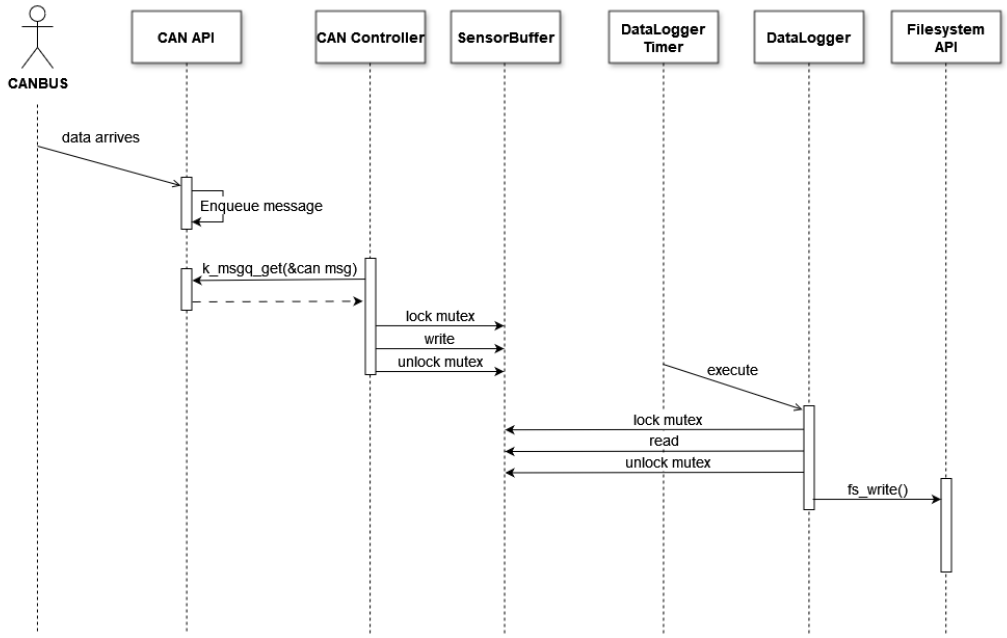

Figure 4.4: CAN Data Sequence

4.5.2 Optimization on the new hardware

To improve the optimization, the device was duplicated, with one dedicated to transmitting and the other to recording. This allows two factors of improvement: first, each processor has fewer tasks to manage, and second, the processor dedicated to recording can run at 128 MHz instead of 64 MHz, since Wi-Fi requires the processor to run at 64 MHz in the SDK version used during development.

Dividing the system into two units makes it possible to eliminate certain tasks on each unit. The GPS is kept only on the transmission device, since the priority is to optimize the recording function. The SD card is kept only on the recording device. Consequently, GPS data must be transferred to the recording device and the configuration file must be transferred to the transmission device.

The following data flow diagram shows the general architecture of the recording device:

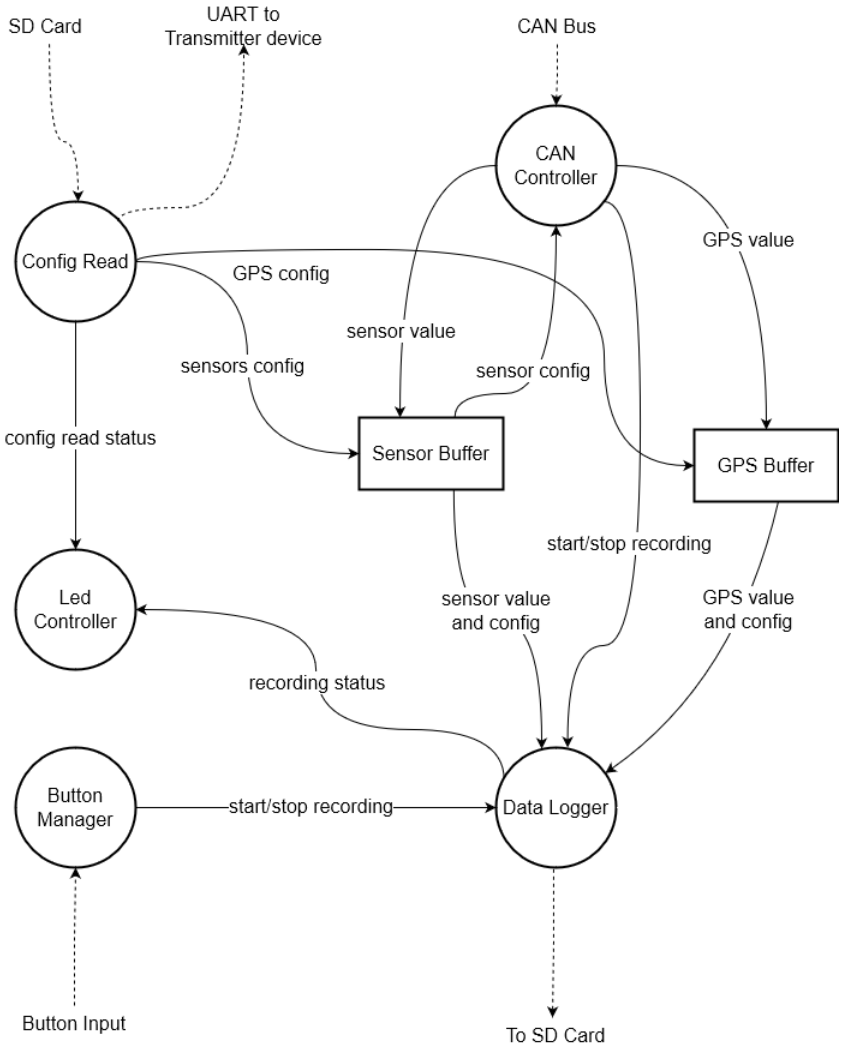

Figure 4.5: Recording Device Flow Diagram

The following data flow diagram shows the general architecture of the transmitting device:

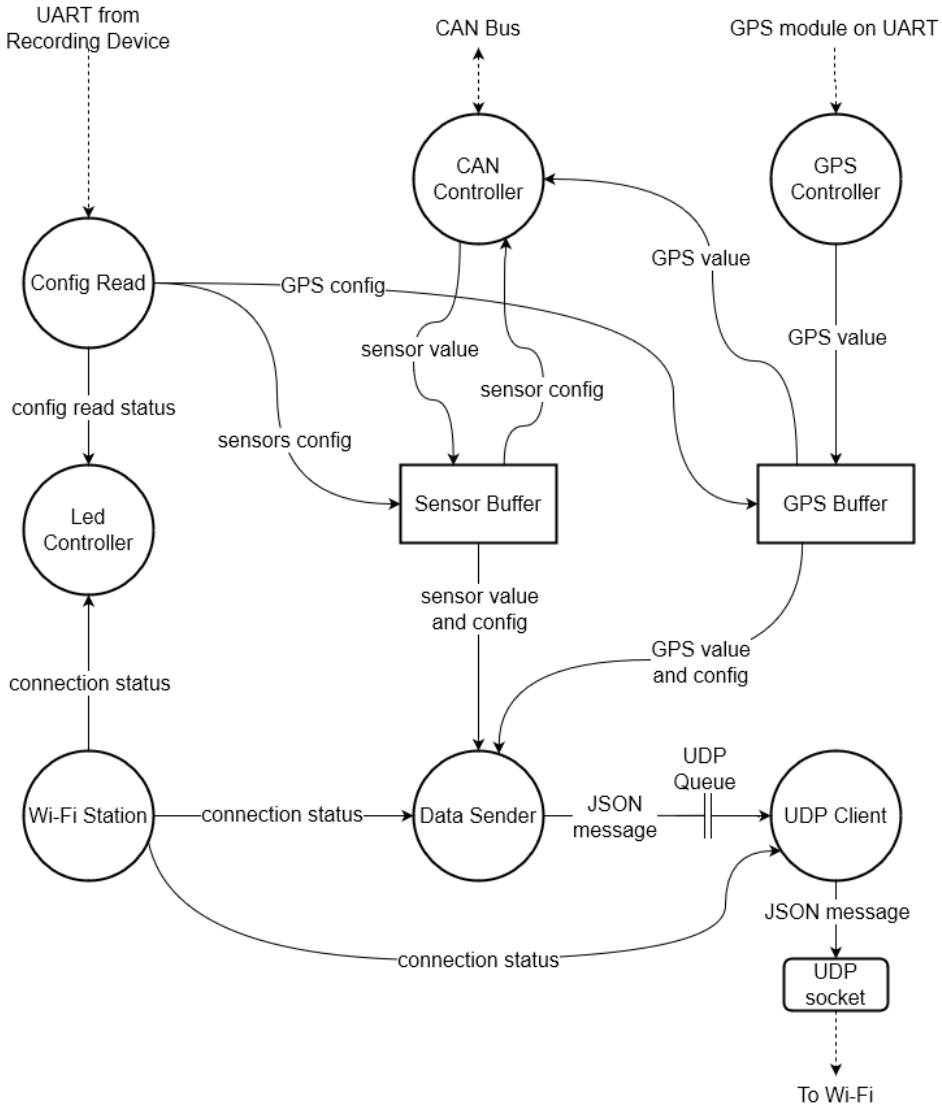

Figure 4.6: Transmitting Device Flow Diagram

4.5.3 Configuration File Transfer

The configuration file is transferred from the recording device, which has the SD card, to the transmitting device using a UART bus. Because the file is 8 KB, this transfer is more complex than the control of the GPS module (e.g., GPS, as described in the original device report), which is only a few bytes. The transmission uses the Zephyr UART [7] with the interrupt-driven API.

The UART device is configured to operate at a baud rate of 115200. At this speed, an 8 KB configuration file is transferred in approximately one second. When the system powers up, the LED and button controllers are initialized first on both devices. Next, the transmitter device pauses and waits for the configuration. On the recording device, the configuration file is read and parsed before being sent via UART. After the configuration is sent, all other components of the recording device are initialized and started. Once the transmitting device receives the configuration, it parses the file and then starts the other components.

The following diagram shows the booting sequence on both devices :

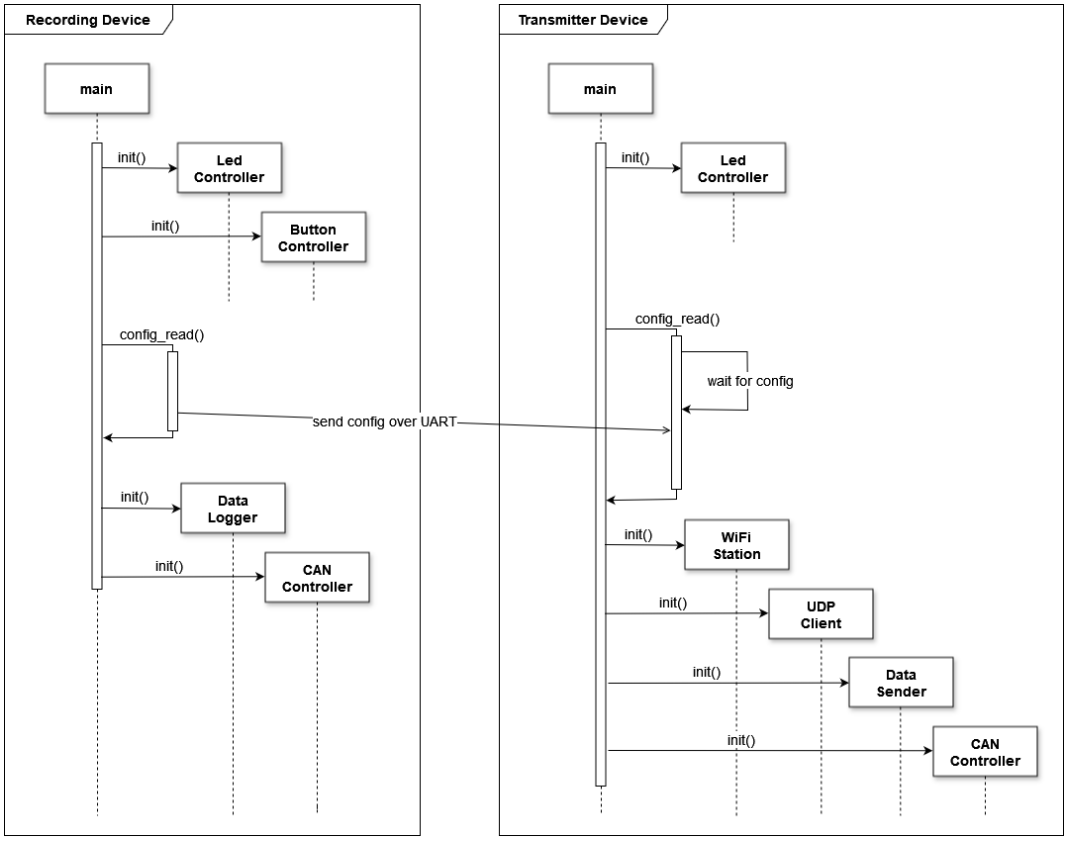

Figure 4.7: Booting Sequence

For the transmission of the configuration file, the code works with small temporary buffers and the bytes are copied using the memcpy operation, which copies multiple bytes simultaneously. This is important because bytes need to be stored quickly as they arrive. Copying bytes one at a time using the polling API is insufficient because the bytes arrive faster than they can be stored, resulting in data loss.

For the send portion, when the UART interrupt is enabled, an interruption is triggered when the device is ready to send data. During the interrupt routine, the system attempts to send a 512-byte chunk. The send function returns the number of bytes actually sent; if less than 512 bytes are sent, the system continues to send the remaining bytes in a loop. When the chunk is completely sent, the system waits for the next interruption to send the next chunk. The buffer containing the configuration file is slightly larger than the file itself and is initialized with "\0" to ensure that a terminating character is automatically added. After the entire file has been sent, the 0x14 symbol indicates that the transfer is complete. Once the file is sent, the UART device interruption is disabled as it is no longer needed.

The following activity diagram shows how the transmission works:

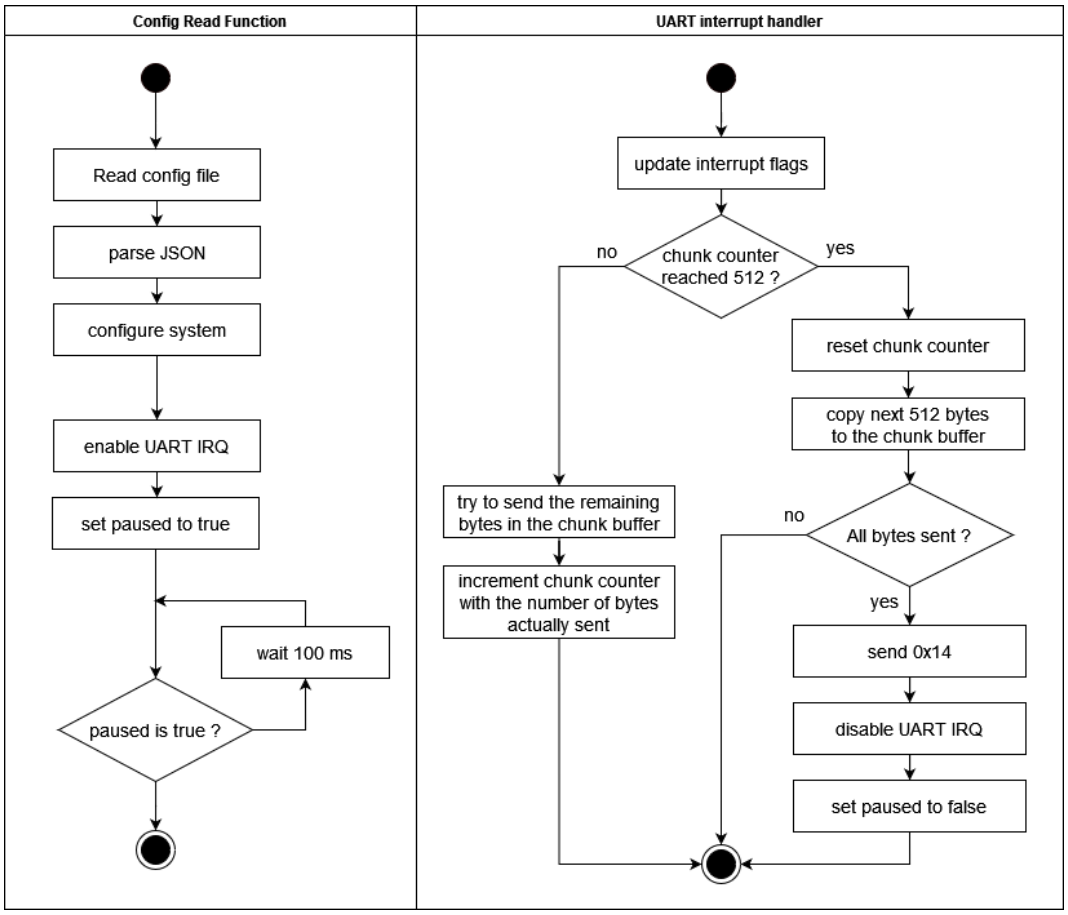

Figure 4.8: Configuration Transmission

A similar process applies to the receiving part. An interrupt is generated when new bytes arrive in the UART receiver. During the interruption, the system attempts to read a 64-byte chunk. The read function returns the number of bytes read, and the system copies those bytes into the buffer containing the complete configuration file. The system then waits for the next interrupt to continue copying bytes from where it last stopped. When the terminator (0x14) is detected, the UART device interrupt is disabled because it is no longer needed.

The following activity diagram shows how the reception works:

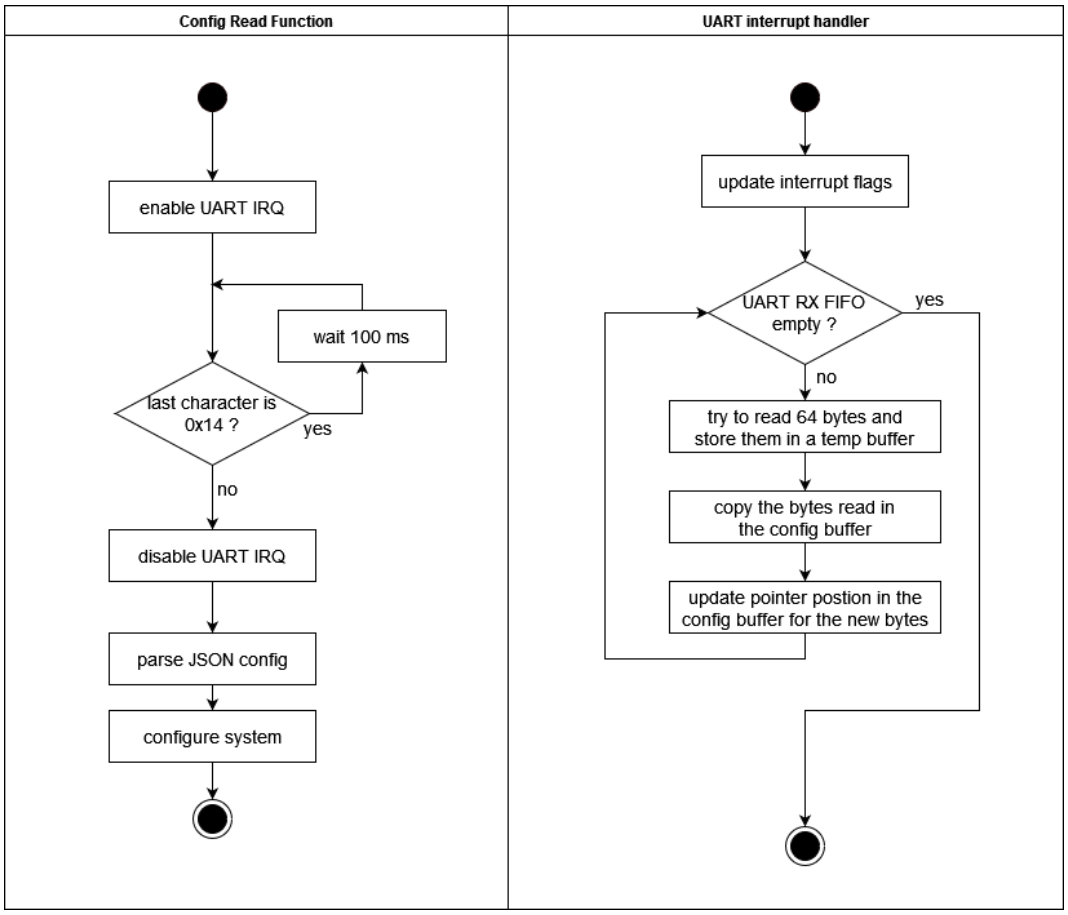

Figure 4.9: Configuration Reception

4.5.4 GPS data transfer

The GPS data is parsed and small calculations (e.g., unit conversions, as detailed in the original device report) are performed on the transmission device. The data is then transmitted over the CAN bus to the recording device. On the recording device, the data is placed in the GPS buffer, mimicking the input from the GPS module. Using the CAN bus allows the same thread to handle receiving data from the GPS and other sensors, improving performance.

As data is transferred over the CAN bus, CAN IDs must be assigned to the system. To ensure the system is configurable, a field was added to the GPS section of the configuration file. Three IDs need to be assigned: the first for a message containing the longitude, the second for the latitude, and the third for a message that includes the fix information, vehicle speed, time, and date.

The configuration file section simply contains three fields for the CAN IDs:

```
"CanIDs":
{
    "Lat":"0x05",
    "Long":"0x06",
    "TimeFixSpeed":"0x07"
}
```

Listing 4.5: Configuration of the CAN Button

The following figure illustrates how the data is stored in the CAN frame:

CAN ID	8 data bytes							
	0	1	2	3	4	5	6	7
Longitude ID	-	sign	characteristic		mantissa			
Latitude ID	-	sign	characteristic		mantissa			
TimeFixSpeed ID	fix	speed	year	month	day	hour	minute	second

Figure 4.10: GPS CAN Frames

On the existing system, GPS data is received by the GPS Controller component. The data is then written to the GPS Buffer. To send the data to the recording device, a new timer was added with a period fixed at 1 second, matching the GPS module's data transmission rate. During the timer interruption, a work item [4] is submitted to avoid blocking other processes during the interruption. In the work handler, CAN frames are created with the values currently present in the GPS buffer and are sent using the Zephyr CAN API [3].

The following sequence diagram shows how the GPS data are sent over the CAN bus:

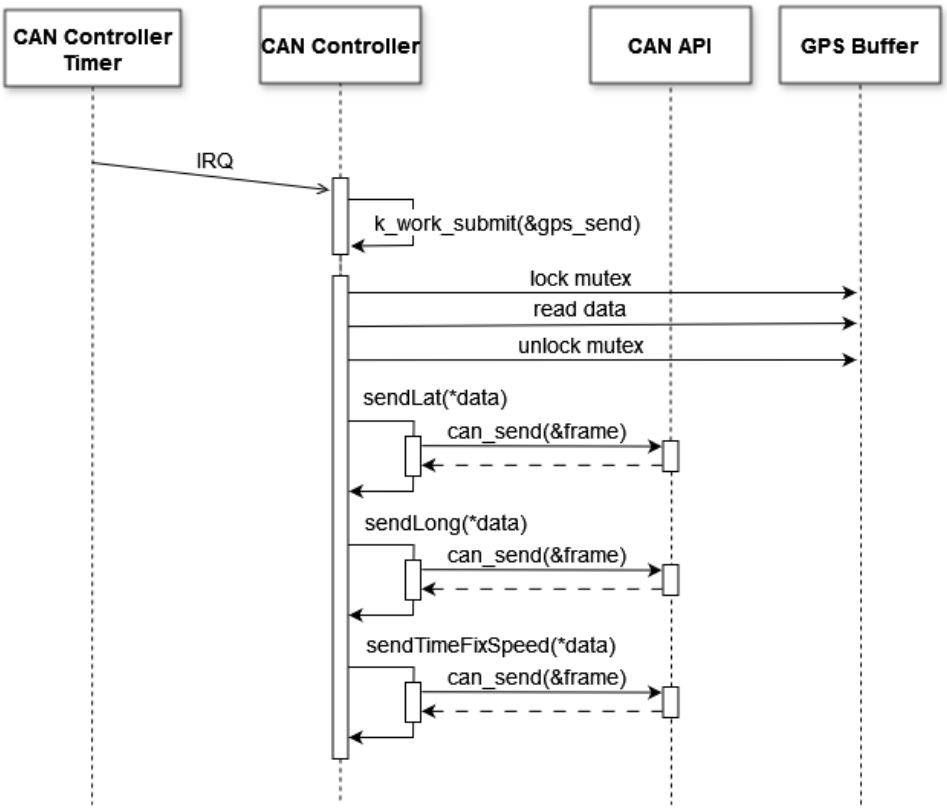

Figure 4.11: GPS CAN Frames Transmission

On the recording device, the reception of GPS data works similarly to the other data transported on the CANBUS. The incoming frames are stored in a queue by the CAN Zephyr API. The CAN Controller component reads the queue and processes the frames. If the CAN ID corresponds to GPS data, the data is extracted from the frame and written to the GPS buffer.

The following sequence diagram shows how the GPS data is received from the CAN bus:

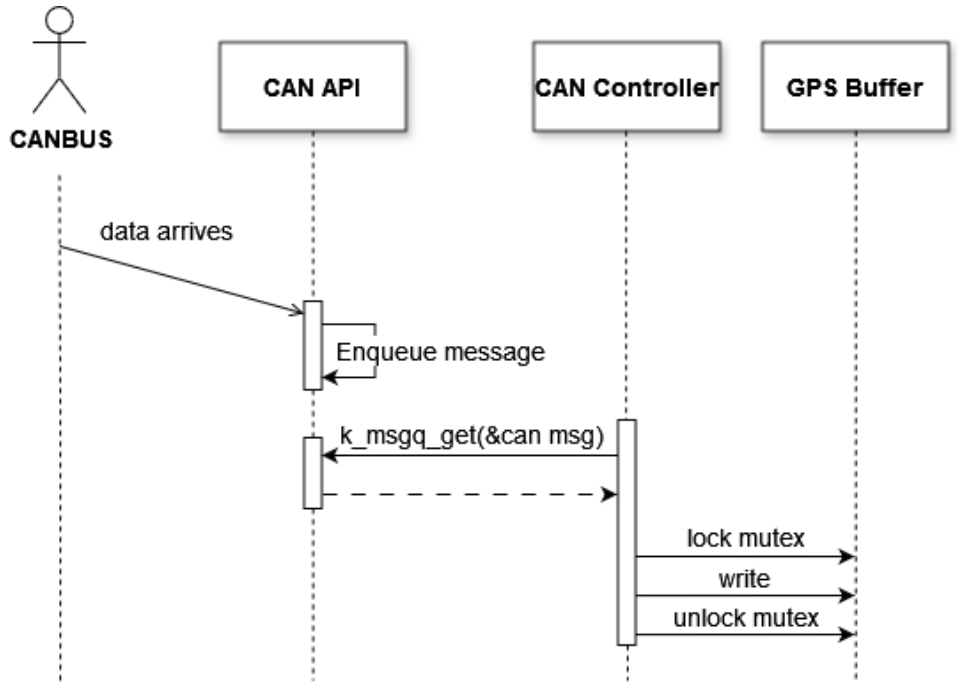

Figure 4.12: GPS CAN Frames Reception

At this point, the GPS buffers of both devices are synchronized. The operations involving the GPS data, such as reading from the module via UART, writing to the SD card, and transmitting to the remote PC, were copied from the existing system.

The results of the optimizations are measured and commented in the Tests chapter.

5 | Tests

The tests aim to measure the impact of the optimizations on performance and to define the system's operating point. Since the system is highly configurable, performance varies significantly depending on the configuration. Adding more sensors, CAN frames, etc., will affect performance. The tests were conducted on a configuration similar to the one that will be used in the car.

The test variables are the message frequency on the CAN bus and the recording frequency on the SD card. The results show the maximum values for these two parameters.

The other parameters are fixed for all the tests :

Wi-Fi transmission : *enabled*
GPS : *Active with fix*
Log recording : *enabled*
Number of sensors configured : *22*
Number of different CAN messages destined to the telemetry system : *8*
Number of different CAN messages destined to the telemetry system : *4*

The CAN bus is working at 500 KBits/sec. A CAN message is 125 bits long on average, which means the CAN bus is saturated at 4000 messages per second. However, these tests targets to characterize the telemetry system's limits and not the CAN bus limit, so to avoid problem with the CAN bus saturation, the CAN message rate was limited to 3400 messages per second, which correspond to an 85% usage of the CAN bus.

The recording frequency was tested up to 1000 records per second because the system is not designed to operate at higher frequencies. Additionally, recording logs at higher frequencies would be impractical, as it would require sending messages on the CAN bus at 1 kHz, quickly saturating the bus.

The tests were performed by gradually increasing the CAN bus message rate and recording rate until the system failed. The last sets of working parameters were retained to characterize the system's limits. The CAN bus message rate included only the messages destined for the telemetry system. During the tests—except for the base system tests that did not include the CAN filter—additional messages not destined for the telemetry system were sent on the CAN bus, totaling 3400 messages per second, to test if the CAN filter was working well. A test was considered passed if the system operated for 30 minutes without the CAN buffer overflowing.

The following chart shows the results for the existing system and for the upgrades made on the existing hardware :

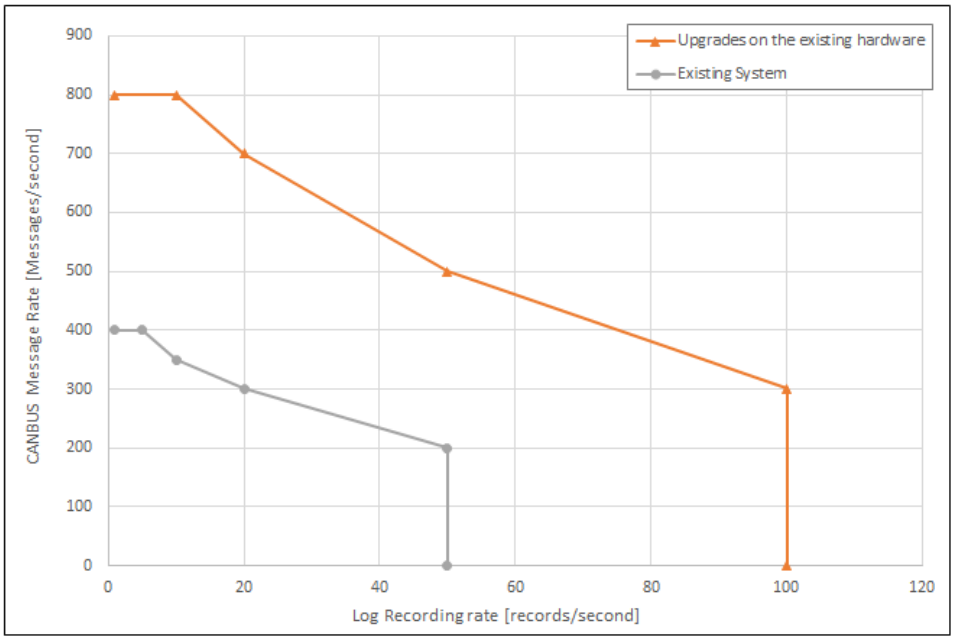

Figure 5.1: Limits of the system - Base system and upgrades on the existing hardware

The results show that the initial software upgrades significantly increase the system's performance. This improvement is primarily due to the elimination of the periodic cache flush. Additionally, the tests demonstrate that messages not destined for the telemetry system do not interfere with its operation.

The following chart adds the limit of the new system working on the new hardware:

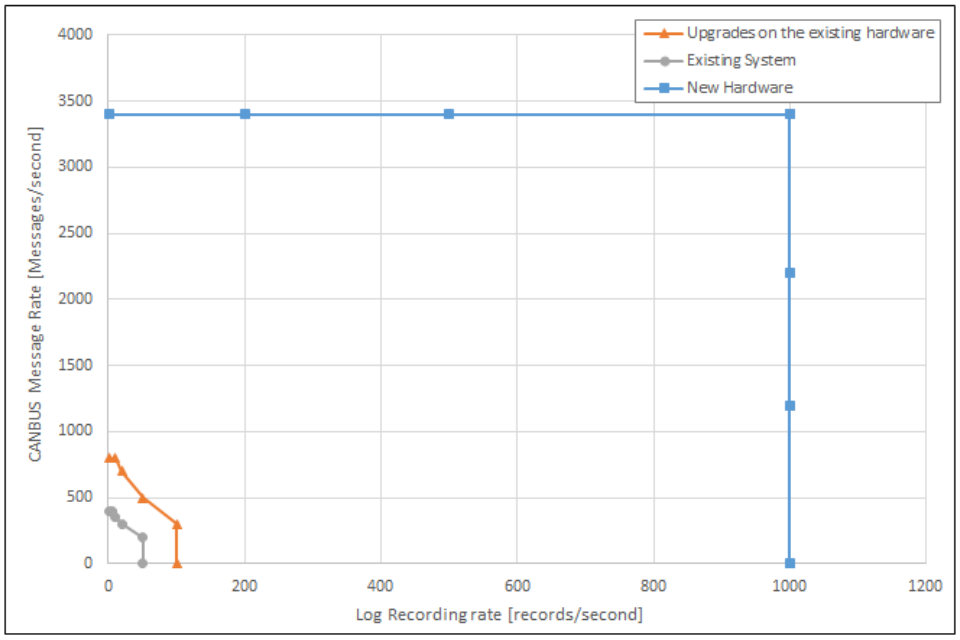

Figure 5.2: Limits of the system - Base system, upgrades on the existing hardware and new hardware

The performance rise is mind blowing compared to the system working on a single device. The performance is not anymore restricted by the telemetry system but by the CAN bus. The performance gain can be explained by multiple factors.

First, splitting the device in two reduced the number of tasks running in parallel on each device, providing more CPU time for the CAN Controller and Data Logger tasks. Additionally, it reduced the number of context switches, and decreased concurrency on shared resources, thus minimizing the time a thread is blocked by a mutex, preventing it from performing its work.

Another benefit of splitting the system was that it allowed the recording device to be clocked at 128 MHz instead of 64 MHz. The existing system operated at 64 MHz because the Wi-Fi functionality did not work at 128 MHz with the version of the SDK and Zephyr used during its development.

It's also important to understand that the system operating on a single device is not incapable of working beyond its limits. However, it becomes unstable, and the CAN Buffer occasionally overflows, leading to data loss. This explains why even a modest increase in CPU time resulted in astonishing improvements.

The last test illustrates the system's performance when the CAN filter is not configured correctly. In this test, messages not intended for the telemetry system are not filtered. Instead, all messages are placed in the CAN buffer and must be analyzed. Similar

to previous tests, the CAN bus message rate only reflects the number of messages intended for the telemetry system. The total message rate is 3400 messages per second.

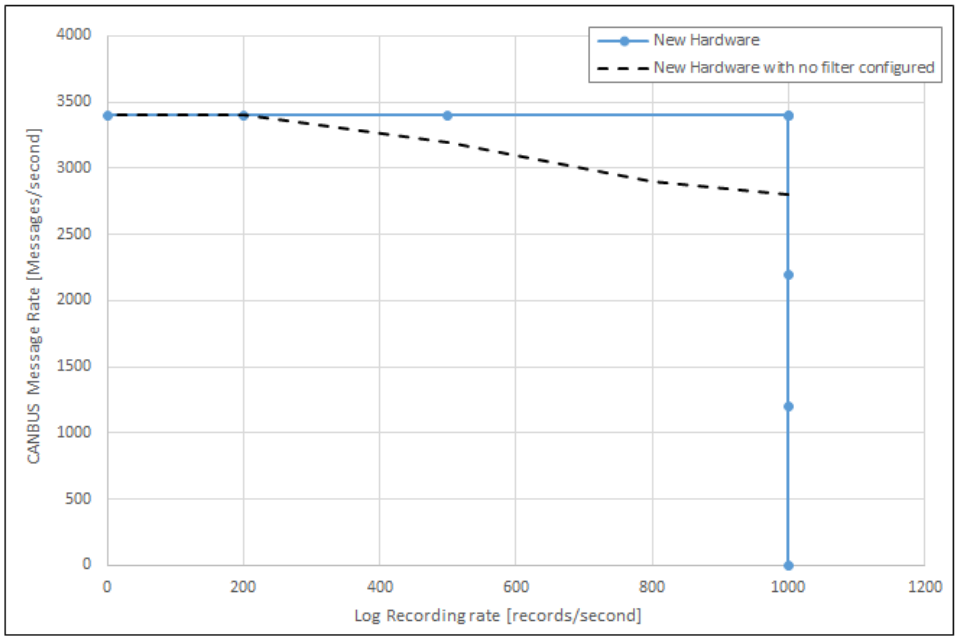

Figure 5.3: Limits of the system - New System

The results indicate that when there are messages not intended for the telemetry system, the system's performance decreases compared to the performance with the same parameters but with messages destined for the telemetry system. This might seem counter-intuitive, but it is explained by how the messages are analyzed in the CAN Controller. When a CAN message is parsed, a loop iterates through the sensor buffer to determine where the data should be placed. Once found, the loop breaks, and the next message can be processed. However, if the message is not intended for the telemetry system, the loop traverses the entire sensor buffer unnecessarily. As a result, a message not intended for the telemetry system takes twice the average time to analyze compared to a message intended for the telemetry system.

In summary, the tests have shown significant improvements in performance. However, it's important to interpret the absolute results with caution, as system configuration has a significant impact on performance. Nevertheless, the tests were carried out in a configuration as close as possible to real-world conditions.

6 Conclusions

6.1 Project summary

The aim of this project was to improve the existing telemetry system used by the Valais-Wallis Racing Team (VRT) for their Formula Student car. The project began by adding a configurable filter to the CAN bus to allow selective message passing without degrading system performance. The second objective was to enable system control via the CAN bus, replacing the manual push-button control. To integrate time and date, the existing relative timestamp system was extended to include absolute time and date. The final objective was to globally optimize the system to handle a higher frequency of data acquisition. This was achieved by splitting the system into two devices: one for data transmission and one for data recording, reducing the processing load on each device and improving overall performance.

6.2 Comparison with the initial objectives

All the initial objectives were successfully met. The addition of the CAN bus filter and CAN-based control improved the flexibility and usability of the telemetry system. The integration of time and date gives more detail to the recordings. The most significant improvement came from system optimization, where splitting the system into two units not only met but exceeded performance expectations. This configuration allowed the system to handle a higher frequency of CAN messages and data logging, demonstrating a significant increase in performance over the original single unit setup.

6.3 Encountered difficulties

One of the biggest challenges during the project was splitting the system into two units. The complex part was transferring the configuration file between the units, which took more time than expected. The configuration file, stored on an SD card on the recording unit, had to be transferred to the transmitting unit via a UART bus. This process, which was finally successfully implemented, required meticulous handling to ensure data integrity and system synchronisation, adding unexpected delays to the project schedule.

6.4 Future perspectives

Looking ahead, there are two primary areas for further enhancement:

- **Improvement of Time and Date Handling**
  To enhance the accuracy and reliability of timekeeping, an RTC (Real-Time Clock) can be added Synchronizing the RTC with the GPS time would ensure accurate timekeeping. This setup would allow the RTC to maintain the correct time independently, and in cases where the RTC battery needs to be replaced, the time can be promptly reset by the GPS. With this improvement, the system would always know the current time, even when the GPS signal is weak.
- **Hardware Optimization**
  Moving from development kits to a custom PCB design would be a significant improvement. Custom PCBs can be tailored to the specific needs of the telemetry system, reducing physical space requirements.

In conclusion, this project has achieved significant advances in the VRT telemetry system, meeting the objectives set and overcoming the technical challenges. The proposed future enhancements will build on this foundation to further improve performance and reliability for future races.

7 | Appendix

- Original project report
- Source code
- Configuration file example
- Test file

All the appendices are available in the git repository at the following link:

https://github.com/svankappel/Telemetry-for-the-Formula-Student

Bibliography

[1] *FS-Rules_2023_v1.1.pdf*. URL: https://www.formulastudent.de/fileadmin/user_upload/all/2023/rules/FS-Rules_2023_v1.1.pdf (visited on 05/26/2023).

[2] *Valais Wallis Racing Team*. URL: https://www.vrt-fs.ch/ (visited on 05/26/2023).

[3] *CAN Controller — Zephyr Project Documentation*. URL: https://docs.zephyrproject.org/latest/hardware/peripherals/can/controller.html (visited on 03/01/2024).

[4] *Workqueue Threads — Zephyr Project Documentation*. URL: https://docs.zephyrproject.org/latest/kernel/services/threads/workqueue.html#work-item-lifecycle (visited on 07/26/2023).

[5] SiRF Technology Inc. "NMEA Reference Manual". In: (2007). URL: https://www.sparkfun.com/datasheets/GPS/NMEA%20Reference%20Manual-Rev2.1-Dec07.pdf.

[6] *File Systems — Zephyr Project Documentation*. URL: https://docs.zephyrproject.org/latest/services/file_system/index.html (visited on 07/28/2023).

[7] *Universal Asynchronous Receiver-Transmitter (UART) — Zephyr Project Documentation*. URL: https://docs.zephyrproject.org/latest/hardware/peripherals/uart.html (visited on 05/25/2024).

Acronyms

BMS Battery Management System. 9

CAN Controller Area Network. 9, 13, 15, 16, 20, 27–29

CSV Comma-Separated Values. 6, 13, 18, 19

JSON JavaScript Object Notation. 6

QSPI Quad SPI. 9

SPI Serial Peripheral Interface. 9, 41

UART Universal Asynchronous Receiver Transmitter. 9, 10, 18, 24, 25, 29