TUHH

Master's Thesis

Hardware Upgrade and Firmware Development of a Prototype for the Long and Short Range Communication of a Multi-Sensor-Plattform

by

Juan Carlos Reyes Andrade

February 28, 2022

1. Examiner:	Prof. Dr.-Ing. Thorsten A. Kern
2. Examiner:	Prof. Dr.-Ing. Bernd-Christian Renner
Advisor:	M. Sc Julius Harms

at the

INSTITUTE
FOR MECHATRONICS
IN MECHANICS

Declaration of Autonomy

I, Juan Carlos Reyes Andrade (Student of Master of Science - Information and Communication Systems at Hamburg University of Technonogy, Mat.-Nr. 21863765) declare, that the present Master's Thesis, examined by Prof. Dr.-Ing. Thorsten A. Kern, titled

Hardware Upgrade and Firmware Development of a Prototype for the Long and Short Range Communication of a Multi-Sensor-Plattform

was composed by myself and without using any resources other than those stated. I assure that when using external publications and sources, either by direct or indirect quotation, I explicitly marked them as such.

Neither this work nor any part of it has been turned in, in this or any similar form, to any examination office. I assure that the print version of this work matches the one included in the attached digital media.

February 28, 2022

Abstract

In-situ measurements play a primordial role in maritime research, therefore drifter buoys are deployed to study different ambient conditions, while transmitting their data regularly over satellite networks. The AMuSeD project at iMEK-TUHH aims to develop a low-cost and reliable multi-sensor platform, for which a current electronic prototype already collects data from I2C sensors. The latter, however, is not robust enough to face harsh physical conditions derived from the different locations embedded sensors could be mounted on or exposed to. Therefore, this Master project upgraded the current prototype to include CAN bus, in order to substitute the original communication bus on the AMuSeD's drifter buoy. Moreover, the developed hardware was equipped with an interface with two different wireless modules (LoRa and an Iridium Subscriber Unit). The integrated system was brought from concept into implementation and validation, creating know-how within the development team about the power constraints and optimizations that take place while working with this technologies. With the results of this project, it is possible then to send bursts of collected data over the Iridium's satellite network. The whole system was integrated at SW level using a RTOS, which made it flexible enough to be portable and keep the door open to explore further power saving features.

Acknowledgment

On this outset of the report, I would really like to show my gratitude to the Institute for Mechatronics in Mechanics (iMEK) at the Hamburg University of Technology and its Director Prof. Dr.-Ing. Thorsten A. Kern, for the accessibility and trust in me while developing this Master Project, for the most part to my supervisor M. Sc. Julius Harms. Additionally, I also take this opportunity to symbolically express thanks to all those people that develop software and hardware by vocation, thus adding up to the open source projects—specially those who value the importance of good documentation. Moreover, thanks to my parents for their constant support throughout my life, to my sister whose example has always motivated me, and specially to Isidro-I will always keep you in my memories. Last but not least, thanks to all people close to me (physically or not) that supported me, specially Denisse for her constructive recommendations and time. Finally, thanks to all those fair journalists that keep delivering objective information about geopolitics during these agitated days. Without them, I could not have kept myself up overnights while writing this report. Let us all hope that better and more peaceful days will come.

Contents

Acronyms

AMuSeD	Autonomous Multi-Sensor Drifter
ASCII	American Standard Code for Information Interchange
AT	AT command
CAN	Control Area Network
I^2C	Inter-Integrated Circuit
IC	Integrated Circuit
IMEI	International Mobile Equipment Identity
ISO	International Organization for Standardization
ISU	Iridium Subscriber Unit
LIN	Local Interconnect Network
LPWAN	Low-Power Wide-Area Network
MIME	Multipurpose Internet Mail Extensions
MO	Mobil Originated
MOMSN	MO Message Sequence Number
MT	Mobil terminated
OSI	Open Systems Interconnection
PS	Power Supply
RA	Ring Alert (ISU's feature)
SBD	Short Burst Data
USB	Universal Serial Bus

List of Figures

List of Tables

1 Introduction

The maritime research employs ocean surface drifting buoys to facilitate in-situ environmental monitoring, observation and analysis of oceanographic conditions. These *drifters* follow the ocean current, collecting different types of data and transmitting it along with their position. At the Institute for Mechatronics in Mechanics (iMEK), part of the Hamburg University of Technology, the Autonomous Multi Sensor Drifter (AMuSeD) is a project that integrates development of cost-effective modular hardware and software, research on wave energy harvesting and new sensor technologies, to deploy a drifter for multi sensor applications [1] (Figure 1).

Within the scope of AMuSeD, the different functional modules interact and complement each other to meet the goals of the mentioned three areas. The *Transmission* functional module has the potential to leverage data processing and data acquisition, as its features complements the system. This data-transmission functionality, in its turn, contains the wireless (or global) and the **on-board** wired communication features. Therefore, these two transmission mechanisms are of high importance, as it is the reliable link between all data collected and the final application that requires the in-situ data.

Figure 1: *Autonomous Multi-Sensor Drifter (AMuSeD)*

In this thesis, a concept for the global and on-board communication is developed to facilitate a reliable data transmission. It is then implemented in a proof of concept prototype and afterwards validated.

The following sections detail the different stages of this Master Project. Section 2 guides the reader through information about parallel projects within the AMuSeD's scope, and the available hardware this Master project started with. It centers on references that helped to understand the implications of non-commercial LPWAN

applications, the theoretical and practical transmission performance of satellite modules, and even the new tools available for quick development with robust communication interfaces. A thorough breakdown of the developed solution is presented in Section 3. Focusing mainly on the robust bus selection (both from concept to transceivers), the power supply design as important part for correct satellite data transmission, including the proposed *switching* feature as core concept for off-board communication.

Section 4 moves towards the working prototype (*commboard*) as the result of design iterations (V-cycle model). It included information like a simple but valuable power performance of the board and how it is influenced by the intrinsic features of the satellite module. A brief but interesting discussion about findings, limitations, and optimization proposals is explored in Section 5. Again, focusing on the satellite module advantages and disadvantages.

Conclusions and future work suggestions can be found in Section 6 and Section 7 correspondingly. The latter, for instance, presents how a RTOS-based firmware could be advantageous in the search for better power performance. Finally, a set of brief and helpful technical resources are available in Section A, adding great value for further references on optimization of the concepts here presented.

2 State of the art

This section summarizes the main prior works that are related to this project at different stages. For instance, thesis projects that took place at iMEK-TUHH, and some others consulted to elaborate the project's concept.

As mentioned in the introduction, the transmission of data is of great importance for one of the main purposes of AMuSeD, that is, reliable collection of data (Figure 2). There is already a board that aimed to start with on-board interaction with sensors. However, this current main controller board (main-board) does not include any working wireless interface. And its on-board interface does not allow it to communicate in harsh environments, neither within relatively large distances between communication nodes. In consequence, the present thesis focuses mainly on the upgrade of the main-board, by developing the mechanisms for **data transmission** in the drifters. The board's concept emerging from this upgrade is referred as *commboard* in the scope of this document.

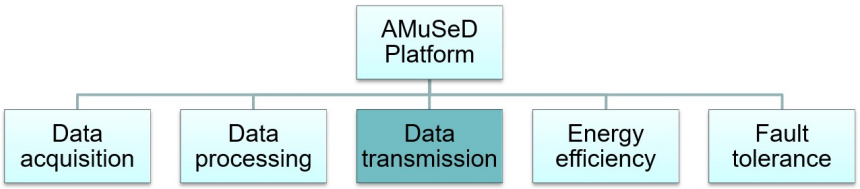


Figure 2: *Master project's goal is a working framework for Data Transmission for both on- and off-board communication in the drifter buoys.*

The on-board communication refers to a main communication bus which makes it possible to exchange data from sensor units (or small development boards) with a main controller, the former group being likely mounted at a relatively long distance from the latter. Regarding the wireless or global communication, it is composed firstly by a satellite modem and, optionally, a module compatible with a low-power wide-area network (LPWAN). These two wireless technologies, referred as **off-board** communication, aim to reliably send any relevant data from the main controller over large distances (Figure 3).

Iridium's service called Short Burst Data (SBD) is considered for data transmission over the satellite network. Furthermore, LoRa is low-power wide-area network modulation technique, that is based on spread-spectrum modulation techniques, and it is foreseen as an appealing technology for future applications in iMEK.

Regarding on-board communication, the current *main-board*[2] uses I2C as the main bus among the numerous external sensors. Even though a Kinéis module is inte-

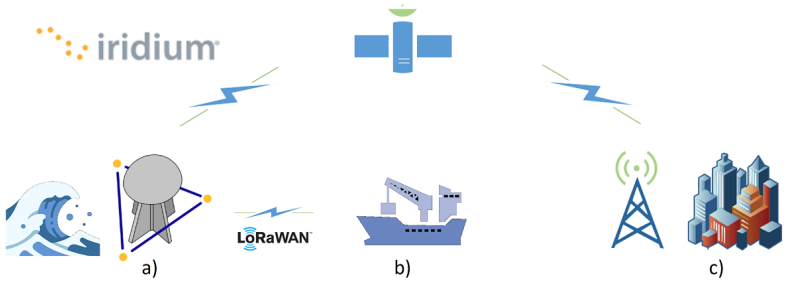


Figure 3: *Off-board communication elements*
a) a drifter buoy, b) a vessel or another buoy that communicates in long-range and c) on-land infrastructure to receive data over satellite.

grated in the main-board, it is not currently working for issues came up, possibly due to power supply's power output and demanding commissioning for satellite acquisition. Summarized, that board's design was taken as a base for the new communication board (*commboard*).

2.1 Monitoring platforms

Regarding modular systems that allow collection and transmission of data over Satellite network, being similar to AMuSeD approach. An interesting full development kit is Arribada Horizon ARTIC R2, introducing a set of rich tools for high-level applications aimed to IoT. The capability to interface two different satellite modules with a modular approach relates directly to this project, nevertheless, that project deals with modules SARA-U270-03S from U-Blox and ARTIC RS certified by Kinéis. Both uses their own modulation techniques for up-and downlink, deviating from Iridium's SBD and relying on satellite trajectory calculators to meet their transmission or reception requirements[3][4]. Both are also part of open-source PCB-shield projects[5], making them interesting for reviewing; nevertheless, the shields or their transceivers appeared to be out-of-stock. Moreover, the mentioned intermission of line-of-sight to the satellite represents a problem. This however is solved by Iridium's technology due to its larger Satellite Network. Condition that benefits the current prototyping phase of AMuSeD.

Additionally, the above mentioned Artemis devices were implemented in the OpenMetBuoy project [6], a contemporary project with similar goals to those of AMuSeD. It is an easy-to-use drifter and waves monitor based mainly on open-source components, aiming to improve the in-situ instrumentation in oceanography and polar science.

Besides drifter buoys, projects focused specifically on satellite modules provided great help for the development of the *commboard*'s concept. For instance, modules like Rock7 RockBLOCK, RockBLOCK 9603, Qwiic Iridium 9603N and the Artemis Global Tracker boards,along with the generic Arduino-Library[7]. It is also relevant to highlight that all the former products were out of stock during the main stages of the project's development. Hence, this situation led to the understanding of the extra value of the developed prototype. That is, reducing one critical link within the supply-chain of the development board including a satellite module.

Interestingly, there is no in-depth characterization of power consumption nor throughput in the Iridium Modem's official documentation. Hence, the need of searching for literature with SBD performance benchmarks. For example, the project [8] reports valuable insights about Iridium modules in small satellites, whereas this performance characterization[9] played an important role while trying to localize on the map a theoretical power performance of SBD-capable modules.

There is a plethora of examples for LoRA networks that were relevant while getting to know the surface of this technology. Specific references are mentioned during the concept. However, highlighted is this LoRa application[10], that shows an example of an underground mesh network of end-nodes, and this research[11], that simulates a Medium Access Layer improved with satellite positioning data.

As conclusion for this section, there are indeed some projects aiming the project's focus areas individually, showing with great advancements and optimizations on their own. However, integration of all these parts is a demanding task, providing the opportunity for this project to bridge an important gap between concept and implementation. Hence contributing to building know-how at iMEK.

3 Requirements and concept

This section presents the requirements outlining the project's scope and also detailing both the solution concept and the structure of its parts. All formulated requirements are summarized and enumerated in a table located in Section A.1.

Throughout this thesis, specially within concept and implementation sections, requirements will be mentioned according to their numbers, for instance, "requirement 14" would refer to "SW means of data transmission over a robust bus to show benefits against I2C". With this information, it is possible to better locate the concept's relevance within the AMuSeD project.

3.1 Requirements

The scope of this Master Project entails the design and manufacturing of a PCB that allows the hardware to exchange data over three different transmission means. To have the tasks cleared, let us clarify each of them. That is, firstly, a main robust wired medium through which the board exchanges data with possible multiple nodes (sensors o small development boards). In consequence, improving the current implementation of I2C protocol in areas like data integrity, management of nodes in the bus and distance among communication parties. The characteristics of the current implementation may lead to loss of data or poor reliability of transmissions. For the nodes do not locate in the same PCB, instead they may be 10 or more centimeters away from the main-board, while exposed to different physical conditions (Figure 4).

Second way of transmission is via a wireless medium, allowing access to a satellite network, in consequence, being capable of transmitting data bursts over long distances. The data, in fact, may be generated or collected through the first wired bus. The referred data bursts, should be capable of covering at least the transmission of 40 bytes each 5 minutes. Iridium's service called Short Burst Data (SBD) is considered for data transmission over the satellite network. The third and optional method for wireless transmission would allow the hardware access to a Low-Power Wide-Area Network. Even though, the full integration of this feature is a long-term plan, the scope requires a concept for an eventual integration of these communication technologies. In addition, all the previous challenges should consider development of power saving features or strategies to optimize them.

The results of the implementation would be integrated into the existing board design and software. Hence, the generated driver libraries should be implemented with a

modular approach and should be suitable to extend the current software (running on an Atmel SAMG55). After the implementation of the prototype, the system needs to be validated for fail-safety.

A table enumerating the above requirements can be found in Section A.1, in this way providing a way to relate the concept's parts to their validation.

3.2 Structure of the solution concept

As outlined in the requirements, off-board (or wireless) and on-board transmission of data is the main topic of this project. There has been previous work that now needs to be upgraded. Moreover, different technologies need to be explored to deliver robust and reliable means of communication in and out of the buoy itself. All this together creates a hardware and software framework to be implemented, representing then a firm base with which the collected data can be reliably transmitted within and out the drifter. Thus, keeping the researched quality of the involved in-situ measurements.

The different parts of the solution interact among each other. Their purpose can be better understood in a layered representation (Figure 5). This is, for instance, the different hardware upgrades (lower layers) allow the user, through middleware libraries, to make use of the final prototype.

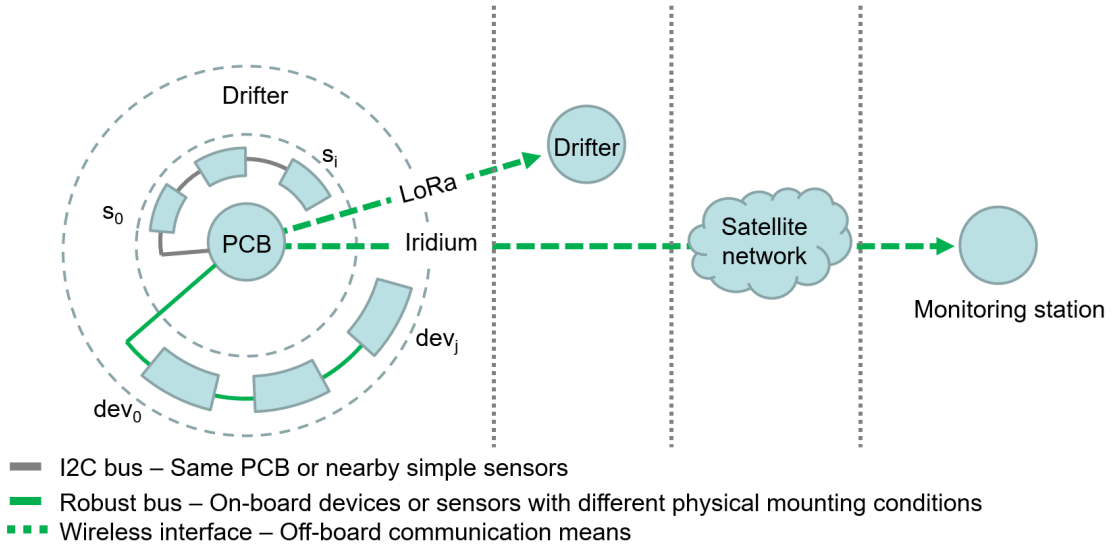


Figure 4: *Devices interacting at different levels in- and out of the drifter buoy. This proof of concept's prototype delivers mechanisms to communicate through interfaces highlighted in green.*

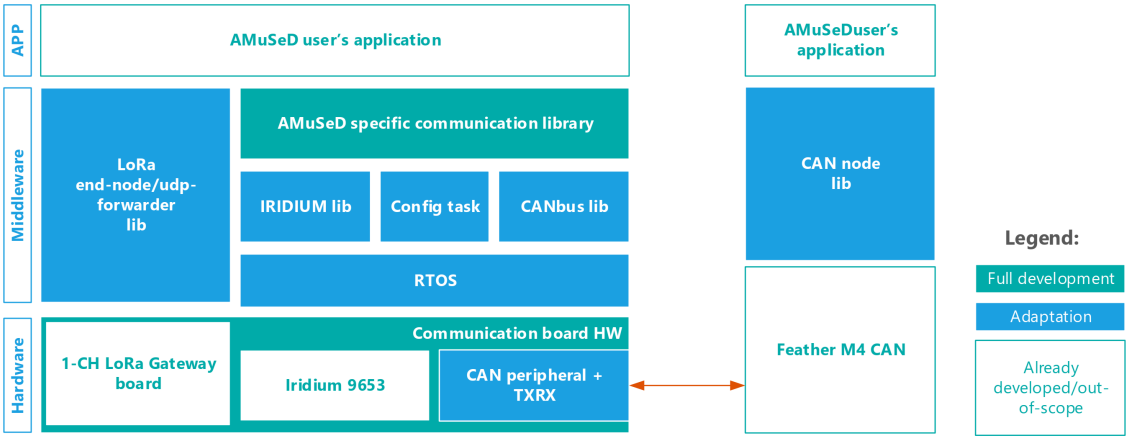


Figure 5: *Sub-modules of the prototype's concept structured as layers. There is hardware and software to be integrated. Green parts are full development, whereas blue ones are adaptations.*

The solution concept can be categorized in the following items, each of them will be detailed in the following subsections:

- PCB upgrade: Constitutes the hardware upgrade that allows the communication features to work at physical level.
- On-board communication: Focuses on the upgrade of the I2C bus to enhance the collection of data that eventually is sent *off-board*.
- Off-board communication
  - LoRa: As pointed out in requirement 10 and 11, this technology is explored to create a long-term integration concept.
  - Iridium's SBD: Service for data transmission over a satellite network.
- Base Firmware: Base libraries to interact with all the hardware upgrade.

3.3 PCB upgrade

In order to fulfill requirements 2, 3, 4 and 5, a flexible and modular concept (Figure 6) was proposed and integrated. This hardware is referred throughout this documentation as communication board or *commboard*.

Besides the formal requirements, the following list summarizes advantages from this PCB manufacturing.

- Safer in the sense of aiming first for a feasible working prototype with the new technologies.
- Practical for parallel development, letting other developers carry out tests with the original boards, while this project focuses on its own modular prototype.
- Generation of hardware know-how, derived from the shortage of Iridium-base development boards (as stated in Section 2.1).
- Flexibility for the developer.

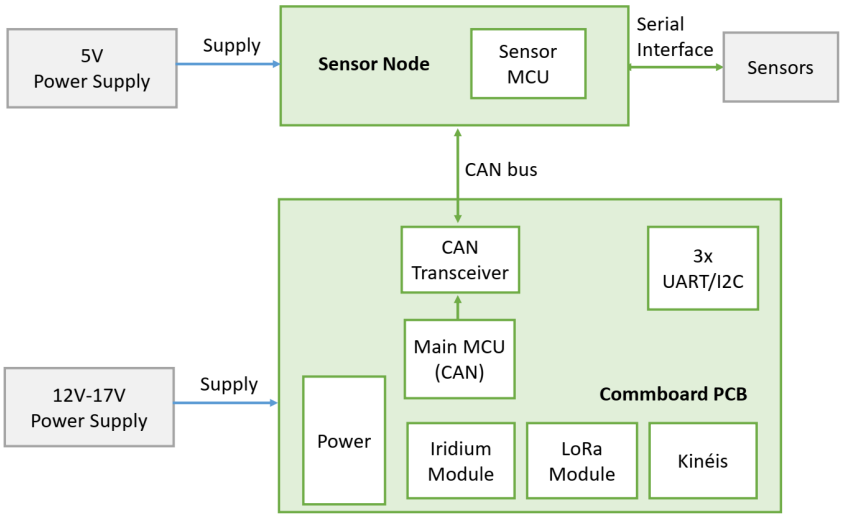


Figure 6: *Communication board's concept or commboard's prototype. This is the crucial intermediate step to the full-integrated board.*

In regard to the quality of supplied voltage, high quality power output is of high importance to ensure a correct setup of a satellite transmission session. The latter is a consequence from the increment in electrical current consumption during SBD transmissions, the latter being the satellite communication service. In addition, the power supply's (PS) concept includes a power charger and in-rush current protection (Figure 7). Further details about specific ISU's power requirements considered during the PS design can be seen in Section A.1.1.

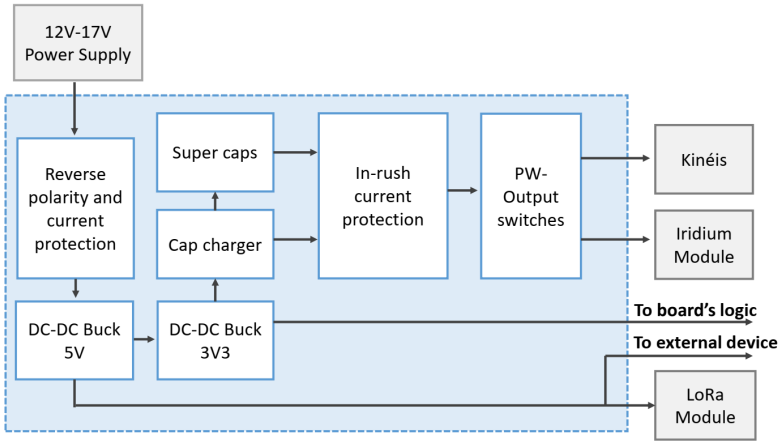


Figure 7: *PS concept to cover ISU's requirements.*

3.4 On-board communication

This part of the concept fulfills mainly requirements 13 and 14, being related to a reliable communication, its robust physical interface and its software framework. Before going into details, important information about the communication protocol selection are presented.

The topology built up by the devices in the drifter is shown in Figure 8. Currently, the I2C bus supports the topology in the same PCB with a multiple-master or master-slave approach, carrying however its own limitations when extended out of the PCB. The basic I2C protocol provides a functional and well known messaging scheme, but does not include data integrity, arbitration of packets, transmission- or reception-error detection features. There are, nonetheless, ways to enhance its native features by extending its data-link-equivalent features or even replacing its standard physical layer[12].

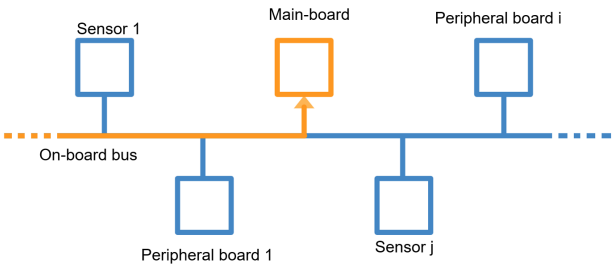


Figure 8: *On-board communication elements*
Main controller, sensor nodes and other peripheral boards coexist sharing a common communication bus. Main board predominantly receives data from these peripheral nodes.

3.4.1 Protocol selection

This section presents the different considered protocols during selection process, then the criteria used to filter them out, until the selection reduced itself to the comparison of transceivers' power-saving potential. Thus, leading to the final selection of CAN bus protocol as the main-bus for on-board exchange of data among sensor devices (Figure 8), fulfilling with requirement 13. A criteria-based matrix technique was used as tool for selection of the on-board communication bus, for there were various options that complied with one or more of the requirements (see Section 3.1). Moreover, the considered buses are shown in Tab. 9 with their weighted criteria along with the selection result. The interpretation of the weight factors (Tab. 2), along with the decision matrix can be found in Section A.6.1. This weighting factors are used to calculate the relative importance of the criteria in the decision matrix.

Table 2: *Selection criteria considered in the decision-matrix.*

Designator	Description	Weight
A	Robust transmission medium	5
B	Multinode with basic QoS	4
C	Overall power consumption	5
D	Driver/Library ease of usage	2
E	Market availability	3
F	Active technology	1

Especially in relation to I2C, more robust transmission mediums based on differential signals (PCA9615 [13], similar to RS-485) or bus "boosters" (LTC4311 [14]) could be used to transmit the I2C data packets. This, however, could be seen as tailor-made developments at hardware and software level, adding new components to the design without leveraging the communication protocol. It complements only isolated features like long-distance-transmission (the former) or more connected nodes in the same PCB (the latter). Nevertheless, it comes with the risk of incompatibility with other standard I2C sensors that would not have integrated the commented alternatives. The global IC shortage [15] affected largely the availability of most of these handy solutions.

In general, among other factors, the availability of semiconductors[15] needed for a physical implementation played again a decisive role during decision making. For instance, the LIN protocol was found to match already the requirements, in contrast to others (RS-485 or Ethernet based ones), due to its similarities with CAN in traffic management. Nonetheless, the transceivers needed for its implementation were not available at the time of developing. Hence, the results were weighted to adjust the relative importance of availability, providing better selection scope. Moreover,

LIN in contrast to CAN is an outdated bus, nowadays facing a lost of support for applications.

Figure 9: *Results before and after relative importance of the criteria. Before weighting robustness and multi-node QoS, I2C still represents a good option.*

Selection	
Protocol	**Points**
USB 2.0	13
CAN bus	25
LIN	20
I2C	23
UART	20
Ethernet 10BaseT	18
Flexray	9
Serial Modbus	13
RS-485	23

(a) *Raw results*

Selection	
Protocol	**Points**
USB 2.0	36
CAN bus	83
LIN	72
I2C	67
UART	55
Ethernet 10BaseT	58
Flexray	42
Serial Modbus	53
RS-485	72

(b) *Weighted results*

Focusing only on technical features, The main bus "competitor" would have been again LIN or others based on RS-485 (like the rather outdated MODBUS).

Furthermore, in some–but not all– ways CAN bus' physical transmission capabilities can be matched with those of RS-485. Nevertheless, CAN's intrinsic features surpass those of RS-485 in the area of multi-node addressability, bus arbitration, collision control, error detection and error handling. Mainly because RS-485's definition corresponds only to the Physical Layer, leading to require at leas another protocol. For instance, typical UART or MODBUS for compatibility with outdated industrial applications.

An important example of how CAN outweighs multi-node RS-485 occurs during message collisions. Since RS-485 does not count with any arbitration feature, this condition likely leads to invalid or indeterminate bus states. These conditions cause significant current draw, leading even possibly to thermal shutdowns[16].

It is also understood, that the physical layer of robust communication buses normally leads to relatively high power consumption. Hence, another important decision factor reduces itself to the power performance of the available transceivers. For instance, it is worth mentioning, that both TIA/EIA-485A[17] (official standard of RS-485) and ISO 11898-2/3[18] (lower-layer standards of CAN bus) have made great improvements in the area of low-power performance. For instance, the transceiver's supply current (or quiescent current) of both in equal conditions, typically of $370\mu A$[19] and $700\mu A$[20], CAN and TIA respectively. Furthermore, typically low current of $40nA$ in CAN's transceiver can be seen in sleep modes[19].

More technical details related to two *top* transceivers' performance can be found in Section A.6.4.

The lower layers of CAN are already known to be still used in these days, even out of its original automotive industry. In this way, its potential of usage in embedded applications spreads over other fields like robotics; for instance, in embedded small tactile sensors[21]. Moreover, there are at hand many useful CAN-oriented open-source tools[22].

The selection of this bus has other side-advantages like being a base for further high-layer-based communication Protocols, such as CANopen, DeviceNet or newer revisions of the original protocol, namely CAN FD. These, however, are not needed in the scope of this project due to the added-complexity of the upper layers. Furthermore, the current concept has already access to robust features in its data-link layer. Nevertheless, it still represents an advantage if this concept is considered for other applications beside AMuSeD.

3.4.2 Communication strategy and the common data packet

In order to implement a simple message exchange strategy and thus, saving overhead from the software side, an unidirectional communication is carried out (from node to *commboard*). However, for more complex strategies, the basic handling of CAN messages is available in all levels, namely physical and software level in both host and node (requirement 13). This defined strategy, regardless its simplicity, keeps data consistency, since the definition of simple time-windows together with the message's ID ensures the identification of full data packets. Moreover, the concept also discards those that-by any reason- are not complete (requirement 14).

This protocol is proposed and implemented as shown in Figure 10. The nature of the data segments is detected by defining time windows, namely t_a and t_b, the former being time between segments and, the latter, the time between full data packets. These asynchronous events should be handled by firmware.

The common data packet is the chunk of memory that will be transmitted from one CAN node to the *commboard*. Its definition takes into account the following points:

- Maximum length of a message's payload over standard CAN bus (not to be confused with CAN FD, whose Physical layer is common)
- Maximum size of the messages transmitted over the satellite module (detailed in Section 3.5.4)

- Scalability of the communication concept (refer to Section A.6.3 and Tab. 3)

Table 3: *Scalability concept for the data packet.*
This is the structure shared from nodes to host (commboard).

CAN common data packet between nodes and commboard		
Initial size	32 Bytes	Size is a multiple of the CAN bus payload and lower than the first segment of a Mobile Originated Message (70 bytes). It still allows for extra data to be appended (38 bytes) adding up to the current maximum size of a message .
Scalation capability	Up to 64 or 128 bytes	Not implemented. However, the structure allows the number of integers and floats to be expanded to fit into the future new sizes, if required.

The proposed CAN data packet structure (Section A.6.2) represents the first implemented version (v1). It is then foreseen that each new version would use the expandable characteristics of the data packet, thus, being possible to add a data section for bilateral communication (*commboard*-node). See Section A.6.3.

3.5 Off-board communication

This section outlines with detail the scope of the *off-board* communication. As mentioned in the introduction and requirements, there are two wireless transmission methods, namely a LoRa transceiver, which had not been previously selected at the beginning of the project, and the Iridium modem (Figure 3). Besides the satellite communication, LoRa technology is explored to build up an implementation concept in the long term (in this or other projects within iMEK, see requirement 12). This,

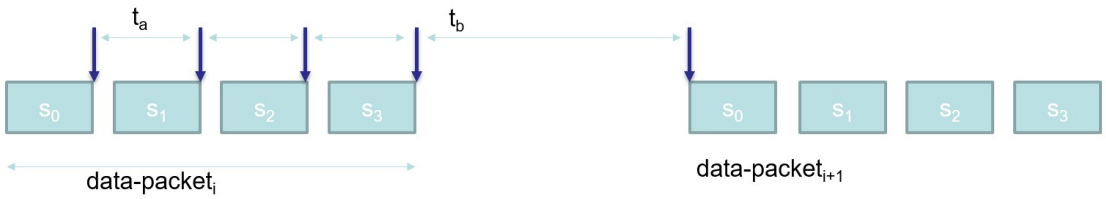


Figure 10: *A full data packet of 32-B is defined for CAN message exchange. These messages are divided into 4 8-byte segments according to standard. Parametrized time windows in firmware are used to identify between full-packets and segments, thus, ensuring data consistency.*

however, does not have a priority as high as that from Iridium's modem, since the latter provides better results in the short- and mid-term within the overall scope of AMuSeD project. Nonetheless, a base concept was developed for LoRa transceivers to be integrated in the drifter buoys.

This concept identifies scenarios that should be explored and implemented as conditions for each drifter in a network. Furthermore, each device would integrate at least a LoRa transceivers to communicate with another one, which would include a satellite modem. It is important to highlight, that the practical implementation is not part of the scope of this project. Nevertheless, some independent tests were carried out at some stages, whose observations are summarized in Section 4.3 and documented in Section A.4.1.

These two concepts are presented in the following subsections, as well as brief technical introductions.

3.5.1 LoRa technology

A LoRa transceiver coincides with the physical layer of the OSI model, whereas LoRa WAN is one of many protocols that defines the upper layers of a LoRa network.

For sake of clarification, let be understood that a device that can generate any kind of data, encapsulate it and send it to another device of the same nature over the same LoRa WAN will be referred as end-node. This end-node corresponds to ABP devices type A and B according to standard[23]. Whereas a device that is capable of collecting data from several end-nodes, forwarding it afterwards over internet to a server for further usage is called a (LoRa) gateway. Moreover, there are some devices that can collect data and forward it without fully complying all the other features of a standard gateway. An example of these other features is listening to parallel channels on the LoRa WAN. These devices are out of standard and are not recommended by commercial IoT frameworks, namely The Things Network (TTN)[24][25]. Nonetheless, they represent an interesting option to work with, since open-source application servers are still compatible; these devices are referred in this document as UDP-forwarders.

3.5.2 LoRa communication concept

The following is the list of scenarios that originated the idea of a possible implementation of LoRa transceivers in the drifter buoys.

- Two or more drifters could be inside similar areas on the ocean, hence, there is no need of sending similar data over satellite.
- Save power by keeping some buoys from using their satellite transceivers, using instead less-power-consuming LoRa ones.
- LoRa is a known Long Range technology used frequently in IoT applications. These can have small and even large networks based on intelligent sensors.
- A possible sharing of data among buoys, allowing a distributed and collaborative control of their behavior.
- There is no need of using satellite network if the buoys are deployed near a on-land LoRa gateway, for example, on coastline or in maritime transport.

Table 4: *General scenarios identified for drifter buoys thoroughly implementing both technologies, namely SBD and LoRa.*

Example scenarios as transition events		
Scenario	Device perspective	Description
I	End-node	No end-nodes nor gateway-like devices on sight: sends its data through SBD.
II	End-node	Detects a gateway-like device on sight: sends its data over LoRa.
III	End-node	Detects other nodes, while no gateway on sight: starts deciding whether change to UDP-Forwarder or not.
IV	UDP-forwarder	End-nodes and no gateway-like devices on sight: continues collecting data from as many as possible end-points.
V	UDP-forwarder	Detects a gateway-like device: starts deciding whether change to end-node mode or not.
VI	UDP-forwarder	No end-nodes on sight: finishes UDP-forwarder mode.
VII	Both	Detects low power

Some systems have been pinned out implementing sensor networks based on LoRa with open-source or private service providers, as introduced briefly in the state of the art. However, at the current stage of AMuSeD, the implementation and management of these networks are a few steps ahead. It is, nevertheless, worthy to have thoughts and information collected about the topic, adding value to this project. Hence, this part of the concept focuses mainly on allowing the prototype to use the LoRa transceiver and the identification of base features needed to start interacting in a real LoRa network. Some further hints about future steps will be presented in Section 5.

Recalling the brief information presented at the beginning of this section, there are end-nodes and UDP-forwarders in a LoRa LpowerAN. This concept works with the ESP-1ch-Gateway and its compatible libraries, for it was researched its hardware can act either like an end-node or an UDP-forwarder.

The different communication scenarios that the system should react to can be seen in Tab. 4. The implementation of these strategies is also proposed to be divided in phases (Tab. 5). Note that it is possible that not all scenarios are considered. However, this will help visualize the scope of future implementation stages, as each scenario demands that certain features work flawless. These scenarios and features still need to be assessed in terms of feasibility and utility in the project. For instance, the devices' interaction assumes LoRa-based devices that are assumed to work as both end-nodes and gateways or, at least, UDP-forwarders. Moreover, they are capable of **switching** between the two modes and can individually **detect** the presence of other parties. In addition, these scenarios also assume that some of the LoRa devices count with an Iridium's SBD service working error-free overseas. Consequently, the scope of this Master project covers specifically Phase 1. Graphics of the scenarios can be found in Section A.4.2.

Table 5: *LoRa communication features implementation proposed as phases. This project delivers valuable information for phase 1.*

Phase	Name	Comments
1	Exploration	Getting to know the technology, acquiring some devices for testing and technical proposals as base plan. Base scenarios: I and IV
2	Basic features	In-depth developing of software to ensure the feature end-node\UDP-forwarder (*swtching* feature). Suggested scenarios: I, II, IV, VII
3	Interaction	Integration of a *detection* feature for end-nodes and on-land gateways. Suggested scenarios: III, V, VI
4	Optimization	Optimizing for saving power and improving the interaction algorithms (optimized MAC strategies). Suggested scenarios: III, V, VII

A *switching* feature of the communication board between end-node and UDP-forwarder capabilities is proposed to be seen as a state machine (Figure 11). This logic concept behind the *switching* feature is the base for future advancements in the above-mentioned phases (requirement 12). A further detailed version of this state machine using super-states can be found in Section A.4.2.

Finally, it is important to stress that the depicted state machine is the proposal to run on the future fully integrated board, hence, its implementation does not take place in the scope of this thesis. However, sufficient details are provided for the abstraction to be both, formally verified and afterwards implemented.

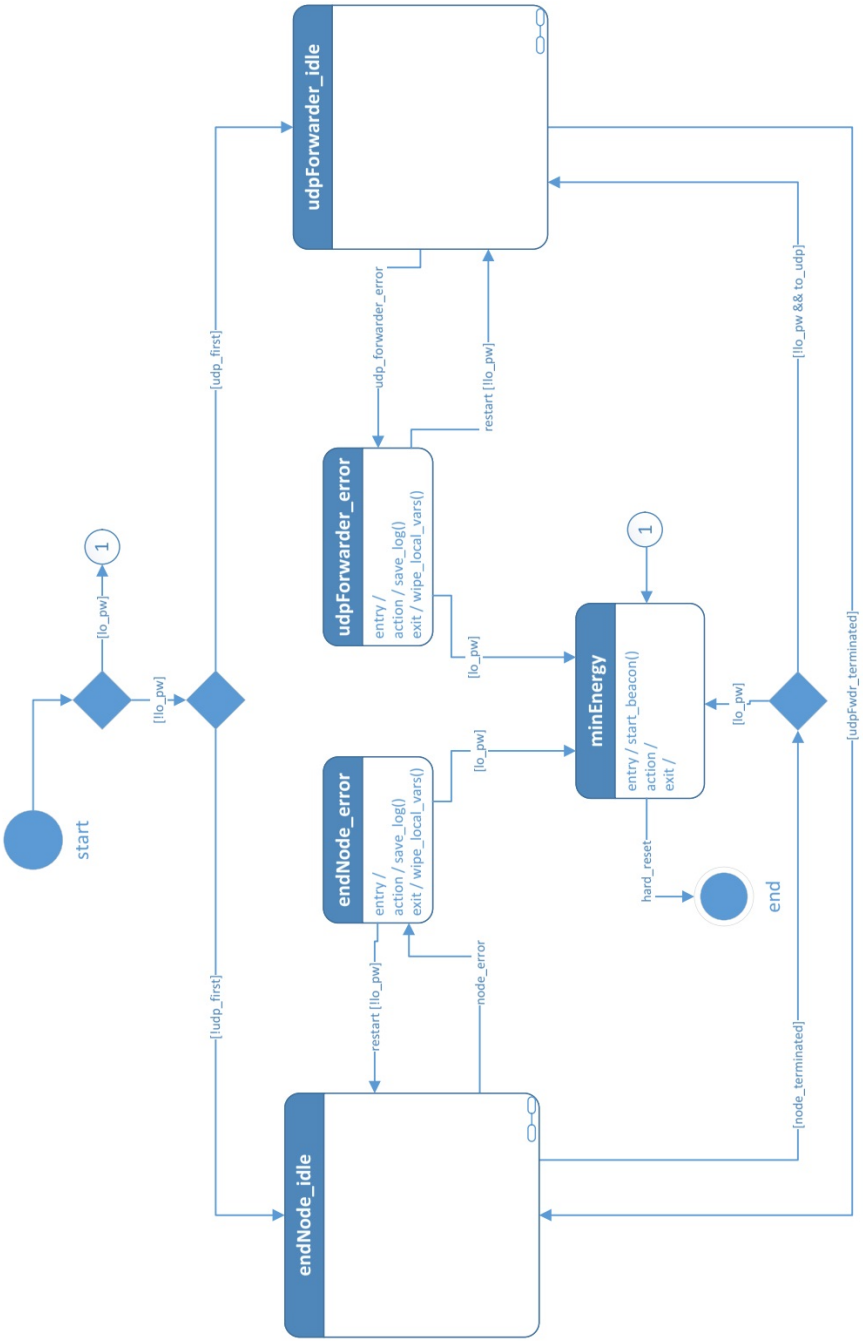


Figure 11: *Concept for LoRa and Iridium off-board communication.*
This state machine represents the switching feature that would allow the communication board to change among wireless technologies.
A detail of the super-states is available in Section A.4.2.

3.5.3 Iridium technology

In the scope of AMuSeD, Iridium technology came into sight as a replacement for Kinéis satellite modules, as this technology delivers a reliable service and established solution for this first proof of concept implementation. However, in the long term, AMuSeD will allow communication over different nano-satellite providers, like Kinéis or SWARM[26] modems, being less costly and providing more power efficiency. This concept and implementation stays however modular, as it foresees alternatives to add other modules.

In this project, the main component of Iridium's SBD is the modem 9603N. It is referred within this document as "Iridium 9603" "SBD Transceiver" "Satellite module" and "ISU." All of these terms refer to the same product.

Iridium's SBD is a simple and efficient satellite network transport capability to transmit short data messages between field equipment and a centralized host computing system[27]. In Figure 12 the elements of the service architecture can be located. Moreover, the modem implements a serial interface through which AT commands are sent from Host to the Iridium Subscriber Unit (ISU).

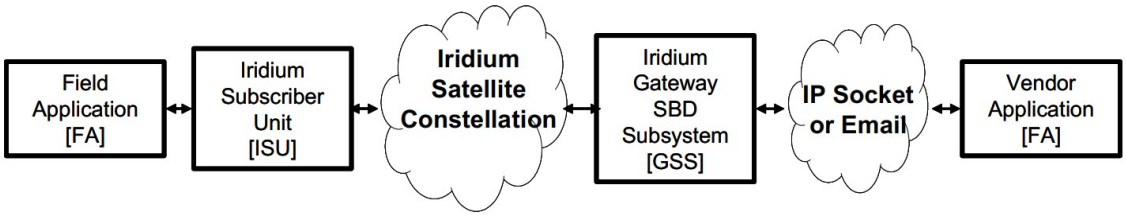


Figure 12: *SBD architecture.*
In this prototype FA (left) is the C library running on the commboard, whereas ISU is the Iridium 9603N and the messages are received thorugh Email.

Messages sent from the ISU to the Iridium Gateway are processed at the Iridium Gateway. These are immediately formatted and sent to a destination email address provisioned with the ISU IMEI. Afterwards, they are sent over the satellite network and will be carried as a binary attachment to an email from the Iridium Gateway to the Vendor Application. The binary attachment is encoded using standard MIME Base64 encoding as defined in RFC 2045. These messages are typically interpreted as ASCII strings[27]. Notice that the theoretically average power of a SBD message transfer is already $790mW$, as denoted in the modem's power requirements[27].

3.5.4 Satellite communication concept

This section deals with the details of the integration concept of the ISU modem into the *commboard* (requirements from 5 to 9). The maximum Mobile Originated (MO) SBD and Mobile Terminated (MT) SBD message sizes are transceiver specific, this is shown in Tab. 6.

This concept uses only the first data segment size of a Mobile Originated (MO) Message, meeting and surpassing the requirement 9. Additionally, this decision takes into consideration the power consumption, since *if optimizing for minimal latency or power consumption, then the minimal number of message segments should be used*[28]. This is because each type of message, both MO and MT, is broken into segments for transmission. The length of the segments is relative to the absolute message length and depends on whether it is a MO or MT message (Tab. 7). Between each segment additional signaling and network overhead occurs[28]. The size used in this concept also coincides with the results shown in [8] and [9], where higher effective data rates were achieved with short messages.

Table 6: *Maximum sizes of messages of ISU.*
In this prototype only Mobile Originated Messages are used.

Transceiver	MAximum Mobile Originated Message Size	Maximum Mobile Terminated Message Size
9603N	340 Bytes	270 Bytes

Table 7: *This prototype uses the first data segment of MO messages.*
Larger messages are divided into several segments but may affect throughput.

First user data segment size	70 bytes
Subsequent user data segment size	135 bytes

The prototype integrates one out of the two methods that the ISU provides for reception of the transmitted data, namely Email MO.

In concordance with the minimum data rate (requirement 9), from which a minimum data size can be deduced, a generic data chunk of 70 Bytes (Figure 13) is proposed as the data unit to be transmitted. At the same time, an equivalent sum of other two data chunks is introduced to ease the tests with ISU and CAN bus. More details about the structure of the CAN bus data structure can be found in Section 3.4.2.

This prototype's concept also considers that the Iridium Modem is a highly customizable communication device. It does not only support the SBD communication, but many other features that complement and leverage the quality of service (QoS)

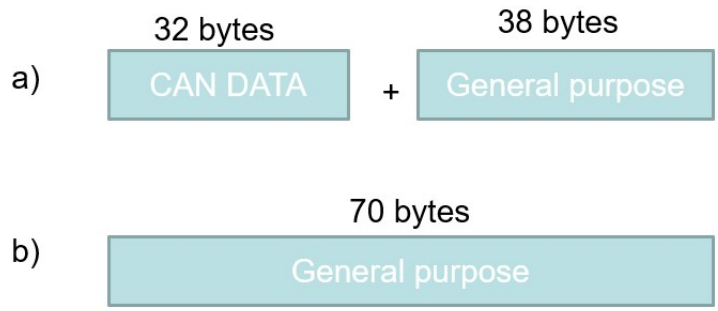


Figure 13: *The prototype has the mechanism to transmit a 70-B data packet over the satellite network.*
For testing purposes, this data chunk is subdivided to meet in one part the data packet received over CAN bus.

of the system it is located in. However, all these features have basically two consequences that may affect negatively a *thin* implementation. These disadvantages could be, for instance, larger libraries to handle all features (like [7]) and longer times where the ISU would be turned on, thus, demanding power even for idle states. For this reason, the solution considers a library as thin as possible, that still allows the basic communication capabilities, leaving aside features that are not necessary for this project. For example, the library avoids "Automatic MT Ring Alert", its notification features and the consequent idle times, since the communication over the satellite network is unidirectional (from ISU to GSS and never the other way around). Details on these two features can be found in Section 4.4.

3.6 Base firmware

Before presenting the firmware concept, some basic concepts concerning RTOS are reviewed, along with details about the generic software interface introduced in the state of the art.

A relevant project in the scope of AMuSeD is the parallel development of the generic communication protocol at software level. This aimed to bridge a gap between the above mentioned main board and generic slave units. The mentioned project is crucial, for it also implemented a task-scheduler to start the integration of features into the firmware, that would eventually ensure determinism of the measurements[29]. However, this project was developing almost simultaneously to this thesis; therefore, its results could not be known at early stages of the project. Keeping that in mind, this thesis' concept explored alternatives, delivering rather positive impact, as it broadens the possible solutions for requirements like asynchronous events handling,

tasks scheduling, determinism and prioritization (similar to those of the generic software interface). Consequently, this proof of concept uses well-known and reliable software development tools like FreeRTOS[30] as framework for the software tests. A technical introductory subsection will is added next.

3.6.1 Real-Time Operating Systems

The following lines aim to introduce a base understanding of a Real-Time Operating System (RTOS). It is brief and serves as a common understanding for the firmware concept (mainly requirements 18,19,20 and 22).

An RTOS is designed to serve where the response time is primordial in order to prevent error, misrepresentation, or disaster, such as, motion planners, machine tool control or monitoring of nuclear power stations. The most important significance of RTOS is managing the resource of a CPU and peripherals, so that a particular operation executes in precisely the same amount of time every time it occurs. This is, well-defined fixed-time constraints. The kernel of an OS provides the most basic level of control over all hardware. It monitors, depending on its complexity, time management, task scheduling and prioritization, memory management, file system, etc.

Regarding task scheduling, multitasking refers to the ability of a system to execute more than one task at the same time. However, in reality, multitasking just creates the appearance of many tasks running concurrently[31]. Moreover, in relation to time management and scheduling, in soft real-time systems occasional misses are permitted and the average response time for an event should be within a specified time. There are other many RTOSes depending on the applications, for instance FreeRTOS, RT-Thread and RIOT. Moreover, some are functional safety compliant like ThreadX or Zephyr OS[32].

Finally, having summarized the advantages of RTOS during prototyping and integrating different technologies, the firmware concept is presented in the following section.

3.6.2 Firmware concept

All the communication features are considered to be state machines that would run in individual tasks (threads following FreeRTOS' naming). The independence of each state machine would allow the system to react to communication events and schedule data transmission accordingly to each communication mechanism, without affecting

the main control task. The libraries concept aims to be portable into other firmware (requirement 18), namely the one being developed under the scope of AMuSeD. The integration proposal considers that the parallelly developed firmware[29] is based on a scheduler. Hence, the whole logic behind this project's state machines (SM) can run in independent tasks, in consequence, they could be run on any other scheduler-based system or RTOS that meets the base features of task scheduling, prioritization and preemption.

The scope considers task notifications for signaling among tasks when necessary, and identification and measures against possible race conditions or buffer overflows are basic part throughout the implementation. Moreover, static memory allocation is selected in the concept, fundamentally as there is no dynamic creation of tasks, the base firmware is desired to be as thin as possible, among other advantages.

Besides the *commboard*'s firmware, there is another relevant piece of code, namely the one that an external user would interact with while sending information to the *commboard*. That is basically a CAN node. Currently, users at Uni-Oldenburg employ I2C nodes based on Arduino. These nodes should be upgraded to meet the new prototype's hardware. Therefore, and in accordance with requirement 22, the implementation of the node's concept considers a user-friendly programming environment.

3.7 Integration concept

All the previous parts from this prototype in this Master Project represent an intermediate step towards a full-integrated concept, merging in the future the features of *commboard* with those of the *main-board*.

This ensures a modular transition from the original prototype to the full-integrated concept (Figure 14), Hence, by having new technologies available in an intermediate prototype, full redesign of the main board was avoided, and left room for allocating crucial resources to the highest priority tasks (off- and on-board communication-related ones). Moreover, this sensible decision prevents the rather *risky* situation of completing a redesign of the whole main board without having even tried out the selected new technologies.

In the following sections, more details about the implementation, results, limitations, and discussions are described.

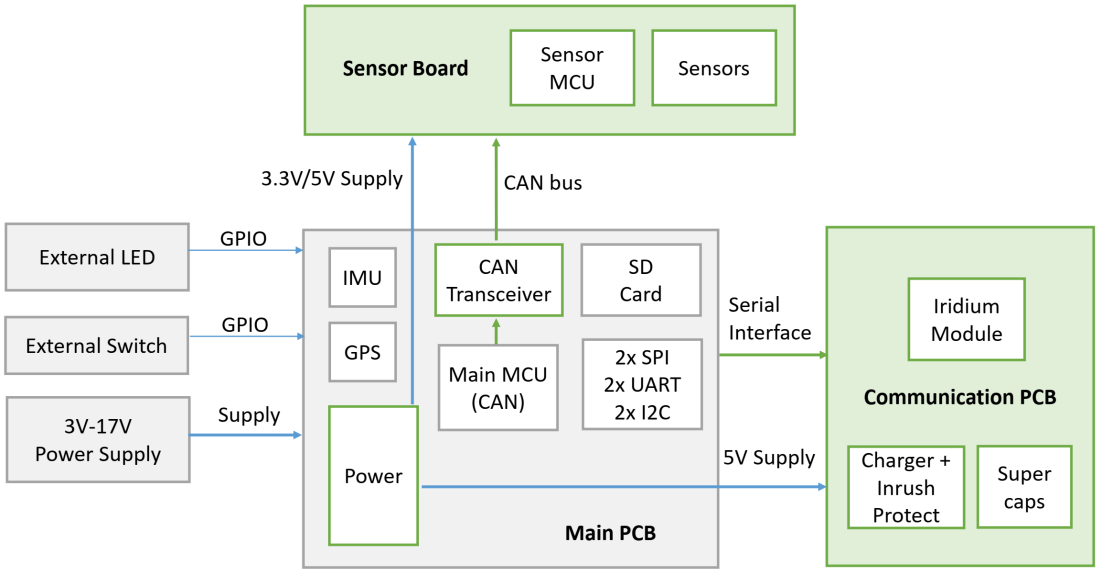


Figure 14: *Future full-integrated board concept.*
This would result from merging the original board with the working commboard (light-green).

4 Implementation

This section summarizes the implementation and structures it as presented in the concept.

4.1 PCB upgrade

This subsection depicts the results after the manufacturing of the PCB design (Figure 15). This PCB now integrates all necessary hardware to interface the on-and off-board communication technologies. All hardware features were validated thoroughly in different stages of the tests (see tables in Section A.3). Even though, the overall tests were successful, in consequence providing a working prototype, some misadventures can be highlighted. For instance, a copper trace linked to one pin that at CAN transceiver was found to be faulty; hence, the slope-control and external reset of the IC was not possible to implement (Tab. 17). However, external wiring was used instead to continue the development of the prototype. In the same line, a temporal stand-alone adapter was also designed and manufactured thus allowing the functional integration of the ISU into the *commboard*.

In general, the main hardware features fully allowed continuing development.

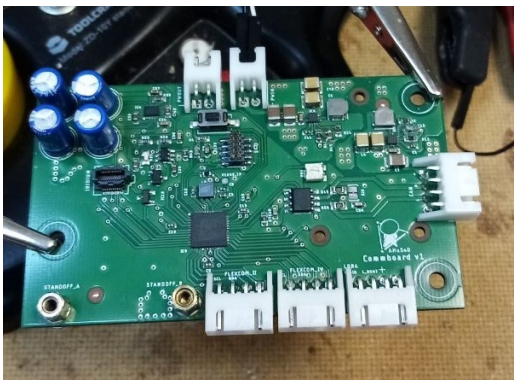
4.2 On-board communication: CAN bus

This section presents the results of the implementation of the selected CAN bus and its performance, in this manner completing the outline presented in Section 3.4. Fundamentally, the physical layer of this protocol is covered by the selected transceiver, whereas the data-link layer is covered by the peripheral of ATSAME51G18A. The following subsections deepen into relevant information throughout its commissioning and tests.

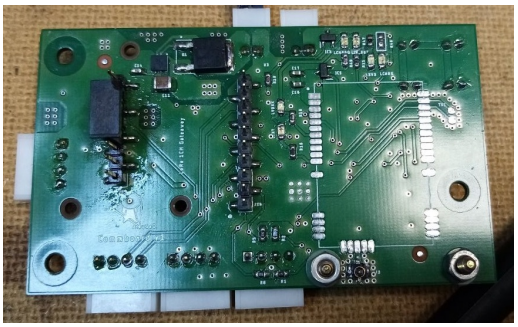
4.2.1 Communication strategy implementation

The implementation of the concept used the tools listed in Tab. 9. That is, a compilation of open-hardware and open-software resources, along with the *commboard*, that built up a bodywork for the proof of concept prototype.

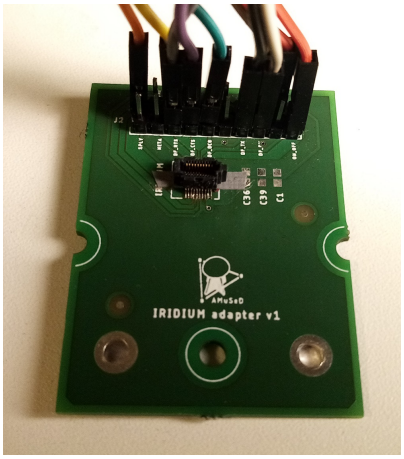
In relation to the message exchange between the *commboard* and the test node, the CAN standard was taken as a base. According to CAN 2.0A specification, the bus



(a) *Top side without ISU.*



(b) *Bottom side without LoRa nor Kinéis modules.*



(c) *Stand-alone Iridium PCB adapter*

Figure 15: *Final manufactured PCBs.*
c) is the result after finding out that ISU had a hardware incompatibility in its serial interface. This adapter was designed to carry out the one-time default configuration.

messages have a 11-bit identifier that can be used to sort the type of messages, while providing a prioritization scheme.

The prototype implements a simple arrangement of IDs allocating up to 110 nodes (Tab. 8), fulfilling already the necessary multi-node requirement 16. This quantity of nodes could be, however, optimized in the future by using a unique RX and TX message-filter for all nodes connected, requiring from the host to implement a multiple access technique to sort correctly messages from different nodes.

Table 8: *Message IDs arrangement can allocate up to 110 nodes. The least significant nibble of the 11-bit message id corresponds to either RX or TX message. Hence, the node IDs extents from hex values 0x11 to 0x7F, corresponding to the 7 most significant bits of the message id.*

ID	Comment
0x100	Host Rx ID
0x101	Host Tx ID
0x110	Node 1 RX ID
0x111	Node 1 TX ID
0x120	Node 2 RX ID
0x121	Node 2 TX ID
...	...
0x1x0	Node X RX ID
0x1x1	Node X TX ID

In this prototype only node 1 is served (Tab. 8). The host listens to the bus with a message filter used for transmission at that node, namely 0x111 (Tab. 8) and stores the data consistently (Figure 17). In the case of more nodes, then an Aloha-type network would take place. In this type of network, nodes are allowed to transmit arbitrarily. This won't represent any collision condition, as the nodes would communicate with different messages ID's, as stated before. Hence, a prioritization of the messages, and consequently of the nodes, is inherited from CAN specification.

Note that the transmission mean was selected to be over a twisted-pair wire inside a standard category-5 cable. This ensures low degradation of the differential signal. This is important in many ways, even indirectly for power consumption. For instance, a simple wiring from high- and low-pin into a CAN bus could also work in some conditions; however, it would lead to rapidly increasing receiving or transmitting errors, in turn, forcing the transceiver to increase messages as they are retransmitted. The data transmission is carried out by default at $125kbps$, anyhow, tests were also made at $250kbps$. Hence, requirement 17 is fulfilled.

Regarding the nodes, and meeting requirement number 22, MU editor was selected to match the available hardware, and so easing the usage of robust features-like

CAN protocol itself, with a friendly user approach[33].

```
main.py output:
Hallo Grom
CAN was on STANDBY
Bus state changed to canio.BusState.ERROR_PASSIVE
Sending message: count=0 now_ms=22324
Sending message: count=1 now_ms=23333
Sending message: count=2 now_ms=24342
Sending message: count=3 now_ms=25352
Sending message: count=4 now_ms=26362
Bus statecanio.BusState.ERROR_PASSIVE
Sending message: count=5 now_ms=27373
Sending message: count=6 now_ms=28382
Sending message: count=7 now_ms=29392
Sending message: count=8 now_ms=30401
Sending message: count=9 now_ms=31410
```

Figure 16: *Node sending data over CAN bus.*
The output from the terminal is part of MU editor. The Python code will log the messages and the bus state.

In relation to the state printed out in the terminal, showing *ERROR_ACTIVE* (Figure 16), it means that errors may or may have not occurred. However, the transmitter can still interact with the bus and its nodes. On the contrary, *ERROR_PASSIVE* means that the bus is in the passive state due to the number of errors that have recently occurred. The device will then acknowledge arriving data packets, but it cannot transmit messages. If additional errors occur, this device may change to *BUS_OFF*. However, if it successfully acknowledges other packets on the bus, it can return to *ERROR_WARNING* or back to *ERROR_ACTIVE* and thus would transmit packets[34].

Table 9: *Hardware and firmware used for on-board communication tests.*

Item	Description	Comment
1	Communication board	Used as main node
2	FreeRTOS v10.0.0	firmware framework for task handling
3	CAN bus protocol	CAN 2.0A as specified in ISO 11898-1:2015
4	Adafruit Feather M4 CAN	Used as the user node
5	Feather M4 image v 6.3.0	UF2 image for circuit python compatibility
6	Library bundle v 6.x	Library bundle for base features of Feather M4
7	canio	CAN library included in the bundle
8	Transmission mean	Twisted-pair cable (0.15 m,0.5 m,1.5 m)
9	Mu Editor 1.1.0b7	Node programming environment

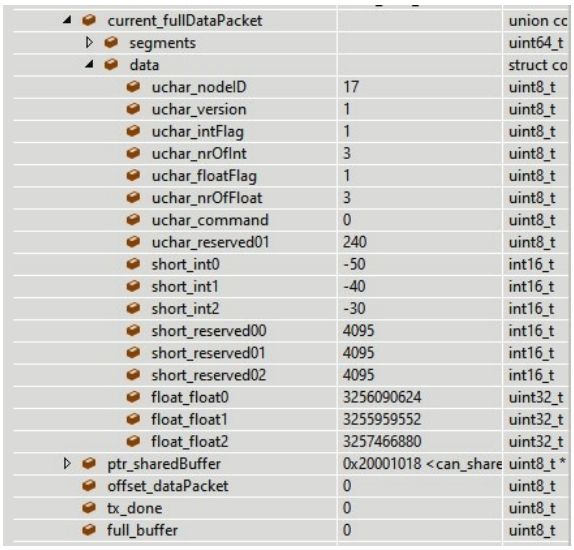
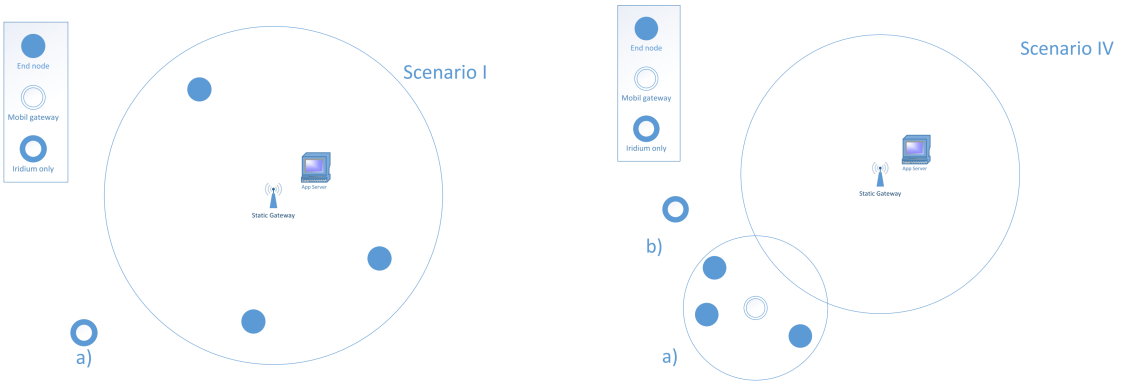


Figure 17: *firmware at commboard stores the received data keeping the order and thus taking care of data integrity.*

4.3 Off-board communication: LoRa

The following section summarizes information corresponding to the proposed phase "Exploration" (Tab. 5). This phase focused on researching and understanding the technology, selecting the transceiver, testing the basic communication between end-node and UDP-forwarder and researching types of application servers. Consequently, the activities related to scenarios I and IV (Tab. 4 and Figure 18).



(a) *Nodes transmitting to a fixed gateway.*

(b) *An UDP-forwarder receiving data from end-nodes.*

Figure 18: *Communication tests with LoRa transceivers focus on scenario I and IV. Large version of these graphics and the complement ones can be found in Section A.4.2*

To carry out the tests as proposed in the concept, firstly a transceiver matching the hardware requirements had to be bought. To locate a suitable option with the prototype's purposes, a general research about the different IC's providing access to LoRa WAN was done. Since LoRa is a proprietary modulation technique, two major manufacturers were found, namely the original IC's manufacter Semtech[35] and a second and more flexible HOPERF's[36].

Afterwards, the advantages and disadvantages of carrying out a self integration of the IC were weighted, resulting on the decision of acquiring a development board that had already integrated the mentioned chip. This avoided long PCB-design-related times, for the stand-alone chip demanded large amount of effort for implementation at both, hardware and software level.

Finally, a practical comparison among the available development boards took place, being size constraints, price, and open-source development tools the main decision criteria. The selection of the LoRa compatible module was affected by the global IC shortage[15]. Finally, two different modules were acquired to avoid possible

incompatibilities with European frequency plan. The selected and explored module corresponds to a UDP-forwarder (Tab. 20) and the development of the concept took place parallel to the tests. The Tab. 21 shows a brief description of the main tasks that took place, being crucial for elaborating the *switching* feature concept (details on Section A.4.2).

The *commboard* described in this documentation is able to interface through hardware with an end-node/UDP-forwarder (Figure 19). Details on the hardware selection process, hints, assumptions for the *switching* feature are presented on Section A.4.1.

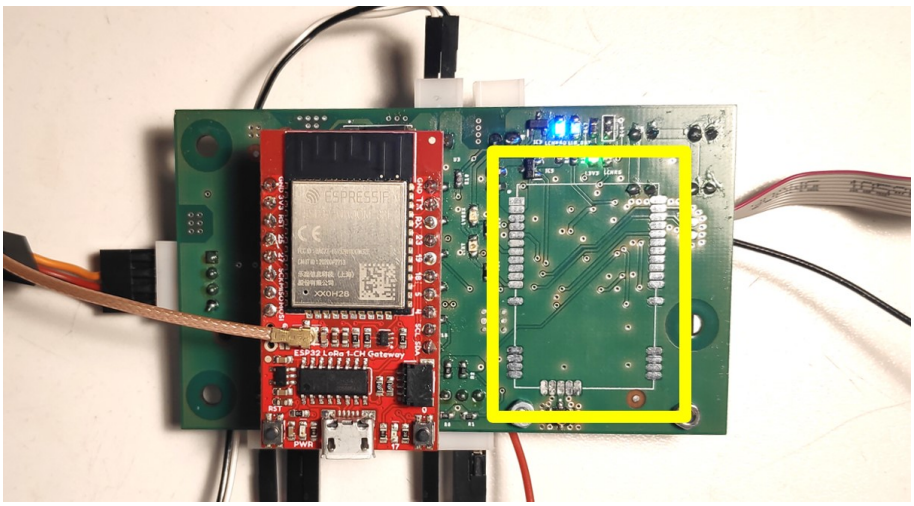


Figure 19: *Bottom view of commboard with the LoRa board attached. Kinéis' footprint is marked in yellow.*

4.4 Off-board communication: Iridium's SBD

This section presents the main results of the integration of the ISU modem into the *commboard*, thus providing the working prototype as main achievement under this project's scope.

Fundamentally, after the hardware integration of the Iridium Modem, the data transmission concept is implemented through the firmware. In this regard, by knowing the final size of the data to be transmitted and the segment that corresponds to the data bits arriving over CAN, a transmission scheme that fills up the data buffers was implemented and tested. Furthermore, this implementation has the flexibility of being possibly adjusted according to the developer's necessity(Figure 20). That is, the data can have origin on an external CAN device or on the communication board itself (self generated or even received by other mechanisms). Representing

advantages for further integration with the main-board or any other device that could be interfaced with the *commboard*.

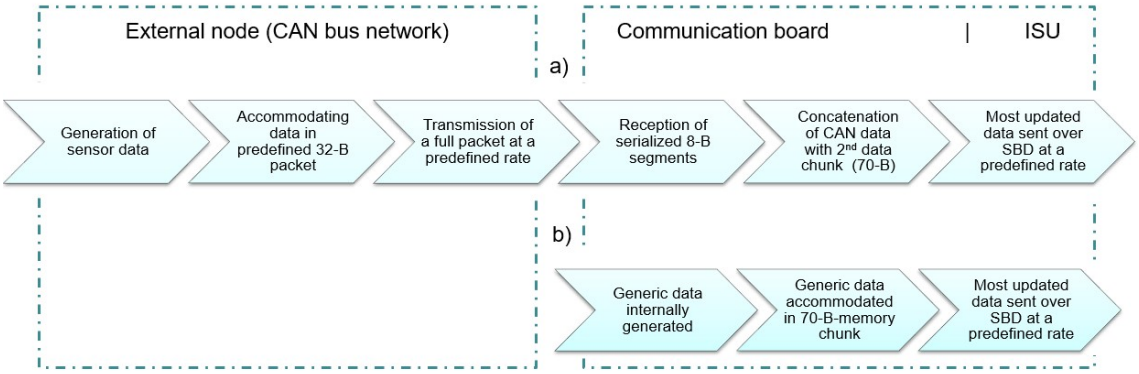


Figure 20: *Satellite transmission concept offers two alternatives either it contains the received CAN data + general purpose data or sends a user defined data chunk.*

4.4.1 Achieved data rate

The data rate capabilities of the Iridium modem is affected by many factors. This became clearer during the implementation, as there is no characterization of it, not even in the official documentation. However, as mentioned in Section 2 there is some literature that focused on the estimation and characterization of the practical transmission capabilities of modems implementing Iridium's SBD service. For example, [9] provides a theoretical value of 100 bytes in 1 second for certain conditions. Then, several experiments are conducted to reach that transmission values. The length of the messages and the number of them, in both receiving and transmitting buffers, affected the performance of the modem. The overall performance is also modified by the SBD service's grade, namely military or not. That is, better results are achieved with the former-no surprise.

It is important to have in mind that the numbers here presented show **effective data rates**. Meaning that, it is based upon the original data packet sent and later on received at the end-point, it does not refer to the baud rate the transceiver can dynamically achieve while implementing modulation and coding techniques. In similar line, this can be seen in LoRa transceiver's data transfer rate. For a message is a packet consisting of a preamble, a header, and payload, the size of which depends on the expansion coefficient (SF) and varies from 51 to 256 bits (22 bps-27 kbps). Hence, the final value depends on range, bandwidth and spreading factor.

To broaden the data rates understanding, another more recent resource was taken into consideration. This project[8] presents an interesting summary of learned lessons after integration Iridium modules using SBD communication. From it, there is another and more realistic approximation of the transmission capabilities using the service. In all cases, the shorter the message the higher the transmission rate. The values coming from both sources, at conditions as similar as possible to our application, are reported in Tab. 10. The mentioned conditions are basically the usage of short messages, around **100 Bytes**, and only MO messages taking place. Thus no MT messages are expected to saturate the receiving buffer of the modem.

Table 10: *SBD data rates summary.*
Two values reported from other projects, the minimum required in our application and the achieved in this prototype.

Data rate	Description
800 bps	Theoretical data rate after a successful satellite acquisition [9]
112.04 bps	Practical value with MO size of 100 bytes (Full duplex mode)[9]
440 bps	Practical value with only MO size of 1960 bytes, minimum achieved[8].
1.066 bps	Calculated minimum data rate to be covered (requirement 9)
46.64 bps	Achieved data rate

An average modem processing time is also provided in [9], along with a standard deviation. It is worth mentioning that the mean value is **7.14 s** and the usual maximum usual value is **15.46 s**. The experimental delay in this prototype was set to be **12 s**, interestingly coinciding within that range.

4.4.2 Handling the Email MO

Regarding the generated mails, unlike MT messages sent to the Iridium Gateway, MO messages sent to the VA (refer to Figure 12) carry additional information in the email message body. This information includes the MO Message Sequence Number (MOMSN), the time of the session, the session status, the message size, and ISU specific geo-location information. A MO-SBD message may be sent to up to five email destinations. The five destinations are programmed into the Iridium Gateway by using the SPNet provisioning tool available to Value Added Resellers[28]. Figure 21 shows the Email MOs received at the configured inbox. The ASCII characters within the *sbd* file can also be seen as raw data, being possible to be unpacked onto different and more efficient data types. An example of how to do that is also provided using python (Figure 22).

The emails are configured as recipients in SPNet and the credentials needed to access to the latter provisioning tool are handed over as part of the final project data bundle. Moreover, as recommended by the technical documentation, library functions are included to check for good signal quality (Section A.5.2) and network availability. This is one of the firmware outcomes that could optimize the transmission intervals.

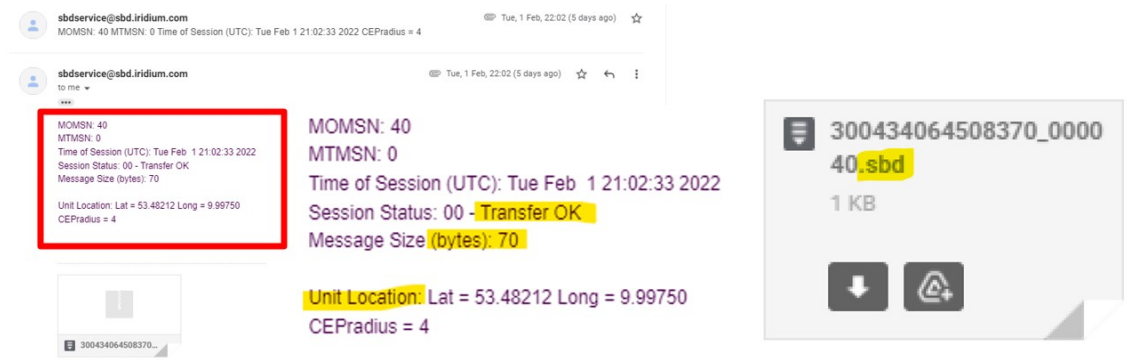


Figure 21: *SBD messages received as emails.*
The ASCII contents can also be seen as RAW data.

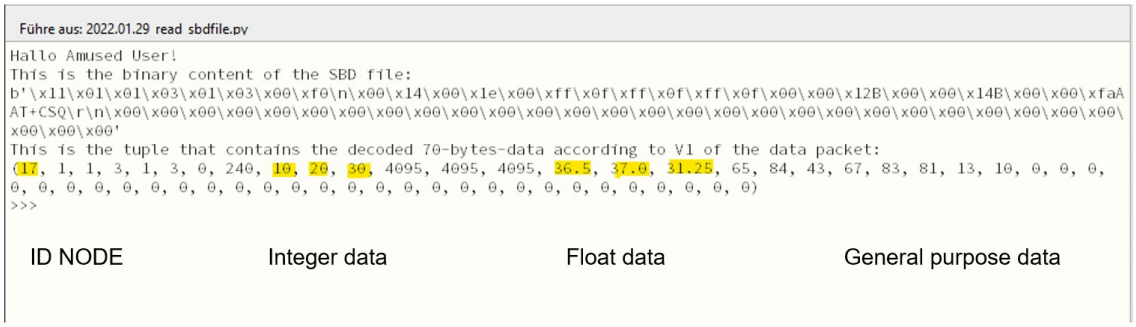


Figure 22: *Reinterpretation of the RAW data*
with a Pyhton code.

4.5 Power management related topics

This subsection presents pieces of information originated during different stages of the implementation; however, since they relate to power management they are located here together.

4.5.1 Power supply implementation

The Iridium 9603N modem is very specific with the power supply requirements, this subsection will show the main points of its implementation. The overall requirement 4 is fulfilled, as the final prototype is able to transmit messages without affecting the function of the control device. For instance, no abrupt shutting downs or power cyclings were detected during the tests. This stability is achieved with the usage of in-rush current protection and super-capacitors. In addition, the more demanding requirement number 4, related to the low noise profile, is in practice covered (Tab. 11); nevertheless, there is room for tests fully focused on measure the prototype's PS noise rejection performance-if needed. Refer to Section **??** for detailed information about the hardware configuration of the DC-DC Buck converter.

Table 11: *DC-DC buck's low noise activated feature.*
The calculations were aimed to have a low-noise output to not affect the satellite module's function.

Configuration	Details
switching frequency	2.2 MHz
Spread spectrum modulation	Triangular
Rated max current	4.1 A
Input voltages	Calculation of saturation current for maximum input and minimum t_{ON} are covered up to 17 V[37]

Finally, another important feature of this PCB is the reverse polarity and current protection (requirement 2). Refer to Section A.2.2 for a brief summary of the information.

4.5.2 Prototype's power profile

This section presents a brief summary of the *commboard*'s power profile. There is currently no power consumption profile based on the working prototype or original

main-board. For instance, [2] focuses mainly on the design values for power consumption, and presents few basic voltage-presence test cases or general-features-test with no power current consumption involved. It also includes a list of requirements that mentions a power consumption goal, namely $1mW$. This places itself far off from the real implementation-at least at this stage. Hence, the power profile can be largely optimized in the future. However, this power optimization process is rather a larger topic.

Therefore, the following measurements are registered, hoping they'll become valuable.

All the measurements were taken at the main external power supply, being set up to fixed 12 V. Hence, the measurements, even though related to the tested features, integrate already the power loses in DC-DC Buck converters, coils, and all the other subcomponents within the communication board system.

Table 12: *These values were measured while running the basic communication library.*

Item	Short description	I min at ext PS	I max at ext PS	power min	power max
1	Base power consumption: MCU bare + RTOS + No tasks	25.579 mA	25.605 mA	306.948 mW	307.26 mW
2	Base power consumption: MCU + CAN TX @8Bps	26.904 mA	26.940 mA	322.848 mW	323.28 mW
3	Base power consumption: MCU + CAN RX @8Bps	26.901 mA	26.921 mA	322.812 mW	323.052 mW
4	Base power consumption: MCU + Iridium TX	23.343 mA	127.9 mA	280.116 mW	1534.8 mW
5	Base power consumption: MCU + Iridium TX + CAN RX	25.652 mA	130.34 mA	307.824 mW	1564.08 mW

Results in Tab. 12 led to the following conclusions.

- Lower consumption of CAN transceiver can be achieved by turning it into the sleep mode; however, further tests need to be taken into account to verify that the RX messages are received. Theoretically, the sleep mode keeps listening on the bus, while not being able to transmit. Fortunately, the on-board concept keeps the *commboard* as only listener. So this test could be done. But due to a possible PCB manufacturing defect, this test had to be stopped.

- The power consumption of the prototype while the iridium module is turned-on is high. However, it's important to take into account the average expected values already foreseen in the concept (Section 3.3), namely SBD message transfer power consumption of $790mW$. More about this topic will be commented in Section 5.1.

4.5.3 Limiting of modem's features for power saving

As mentioned in the concept, some features were left aside to keep the implementation as *thin* as possible, avoiding at the same features-related delays that may require longer idle states of the modem. However, this section presents their description, for these services may result interesting to a future developer. The following points are taken directly from [27] and [38]. The method these features were bypassed is shown in Section A.5.1.

- Automatic MT Ring Alert Implementation: *The SBD service is a fully acknowledged protocol that confirms the delivery of the messages between the ISU and the gateway. To insure this, the gateway does not send unsolicited MT-SBD messages to an ISU that may not be acknowledged. The MT-SBD messages are queued at the gateway until the ISU requests delivery of them. The application, if it is configured to handle the RA, can then initiate a SBD session and receive the queued message.*

- Automatic MT Ring Alert Notification: *When the gateway receives a MT-SBD message, and the destination device is configured for the RA and attached to the network, the gateway sends a RA signal to the device. If the device does not respond in approximately 20 seconds, a second RA is sent. If the ISU does not respond to the second RA, no further RAs will be sent until another MT-SBD is queued for this device.*

- Retrieving MT-SBD Message by the RA *The gateway queues a MT-SBD message for the ISU and send the RA signal. The ISU receives the RA and sends an unsolicited response to the application. This response is either SBDRING in Verbose Mode (mode where ISU's responses are ASCII coded) or "124" in Numeric Mode. The Ring Indicator pin is also asserted. The application interprets the response and initiates a SBD session with the +SBDIXA command. The "A" suffix indicates this is a response to the RA signal and cancels the second RA to prevent a possible race condition. If the application has a MO message to send, it moves the data into the transmit buffer prior to issuing the AT command. If the application has nothing to send and just wants to*

retrieve the queued message, it clears the MO transmit buffer before initiating the session, for instance, a "mailbox check".

4.6 Firmware structure

In order to integrate all the previously described features into the working prototype, source codes are grouped as shown in Tab. 13.

Table 13: *software parts with brief descriptions.*

Item	Name	Description
1	Project libraries	Source and header files for Iridium, CAN and task state machines.
2	Test project	Configuration files in Atmel Studio. Hardware- and FreeRTOS-related.
3	LoRa end-node	Source and header files for end-node tests.
4	LoRa UDP-forwarder	Source and header files for UDP-forwarder tests.
5	SBD message interpreter	Python code to extract data types according to the prototype's definition

Regarding the project libraries implementation (item 1 and 2), there are several peripherals that work asynchronously, either because the host starts communication and then waits for responses, or because the peripherals would create signals that depend on external triggers. For them to be handled without affecting other time-based parts of the software, the scheduler included in FreeRTOS was employed. In this way, different tasks can handle recurrent events in a deterministic manner. As mentioned in Section 3.6, generic tools common among RTOS were used to handle the asynchronous events generated by peripherals and other libraries. In this way, this implementation is kept flexible for other scheduler-based frameworks, for instance, the AMuSeD's generic software interface[29].

To keep the integration simple, only one event makes use of the FreeRTOS notification scheme. This property needed to be adjusted according to the OS' priority scheme, as a result, the peripheral interrupt priority (CAN bus peripheral included) was located below the Kernel's interruption handler (Figure 23). To complement this information, the *CAN_event* used for CAN bus interruptions is emulated with a binary semaphore for simplicity (using *vTaskNotifyGiveFromISR(tHandler_canSM,x)*). Refer to Section A.7 for notes about the library implementation with RTOS.

As introduced in Section 3.6, a user-friendly approach is requested for implementation of a CAN node. Hence, it is worth mentioning the existance of open-source

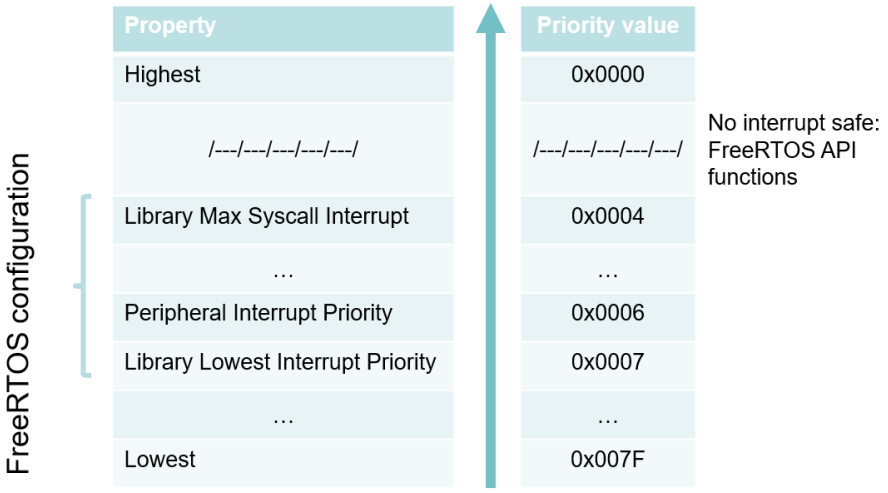


Figure 23: *The above prioritization complies FreeRTOS' recommendations*
Note that the timer interruptions are handled as timer daemon task withing the API region (Section A.7.2).

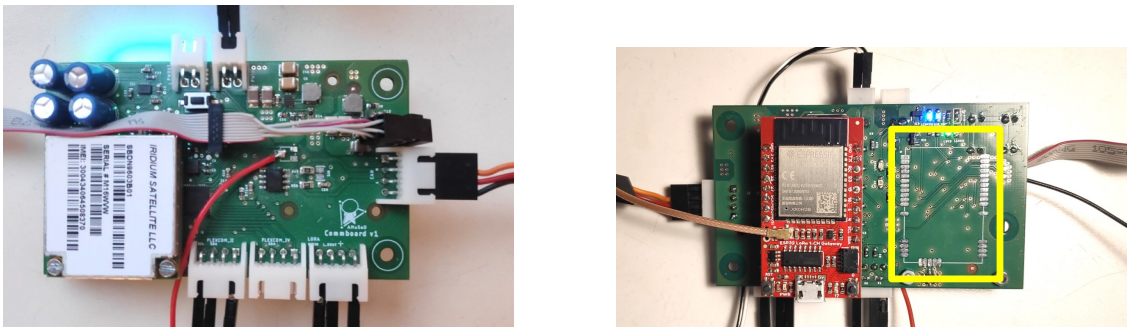
and user-friendly projects for development tools based upon Micropython[39]. For instance, CircuitPython[40] and its environment Mu Editor[33]. These are great help to provide a modern alternative to Arduino, without reducing their technical capabilities, even improving the access to them, while exploring Python language.

Mu Editor is able to program the Adafruit Feather M4 CAN boards. The latter uses a bootloader called UF2 USB Bootloader[41] that runs a tailored-made image of mycropython.

By providing the implementation code with this programming environment, requirement 22 is fulfilled. Moreover, the hardware and the base *canio* library are still and fully compatible with other better-known IDEs like Arduino, thus delivering even more flexibility for the conservative user.

4.7 Final Prototype

The next figures show the final *commboard*, highlighting the ISU attached and a SWD-standard programming interface (top), together with the LoRa-board attached along with the space for an extra Kinéis module (bottom). The prototype of a CAN-node using Feather M4 CAN board with twisted-wire connection is also shown.



(a) *Top view with ISU modem* **(b)** *Bottom view with LoRa module*

Figure 24: *Commboard PCB.*

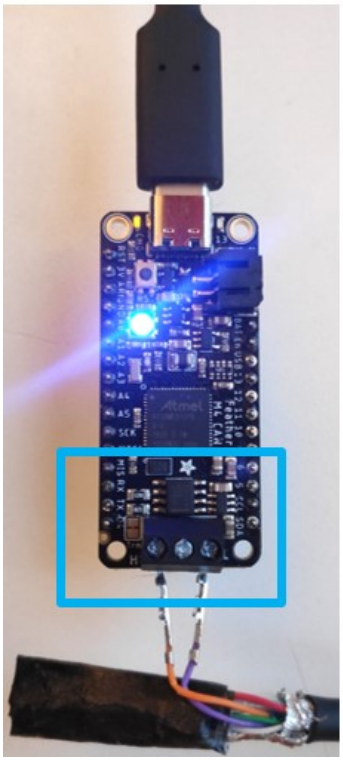


Figure 25: *CAN Node running on Adafruit Feather M4 CAN board.*

5 Discussion

Important results were presented in the previous section, some of them are more relevant as they are related to the overall power consumption of the prototype. This aspect will be of great importance for future steps in the AMuSeD project. This topic, along with few others like vulnerabilities of Iridium network and the data integrity of the CAN bus, are briefly but concisely commented in the following subsections.

5.1 Software potential for power saving of ISU

This section focuses on the discussion derived from the ISU's power consumption. Its possible high value was introduced in the concept and measured on the results. As a result, the overall and plain conclusion is that the prototype consumes relatively high power, namely $790mW$. Another practical value comes from literature referenced in the implementation, specifically a collection of facts about SBD services integrated in a small satellite[8]. The latter provided an example of effective data rate; however, it also provides an insight about its main device, being power supplied over CAN bus while transmitting data. A value of $\mathbf{0.48}W$ average with peaks of $11W$ is reported, from which peaks of $7W$ are linked directly to the satellite transmission. Even though, it cannot be seen as an "equivalent" application, due to different working conditions, it delivers a good reference about the power consumption that such applications may present. This Master's project prototype shows a relatively similar power consumption characteristics.

The scope of this project did not include the optimization itself of the power consumption, since there cannot be any optimization without having first a prototype to work with. Nevertheless, the development of this prototype did take into account these power challenges, offering a platform and a basic power profile to work with.

For instance, as the base firmware was developed using FreeRTOS, its native features for controlling the CPU frequency can be directly explored to improve the sleep states, making the prototype consume less energy when no task is dispatched, evenmore this feature in FreeRTOS framework is called tick-less power saving mode, and can be deeply explored[42].

Moreover, the Section A.5.2 can be consulted to understand, for example, how to check up the signal quality before each satellite transmission. The latter could leverage a power saving strategy. In this regard, the base functions included in the developed libraries give access to implement the commented signal monitoring.

Finally, in Section A.5.1 it is presented the way to power cycle the module to avoid data corruption. This is important, since a better usage strategy implies turning on the satellite only when required. Besides the power-saving-related information, it is also added, how to set up the satellite module by the first time. Thus allowing it to communicate with only two wires (rx and tx pins); in this way, avoiding communication errors from Host to ISU. These errors may appear due to the way the module has its RS-232 interface at 3.3V. Note that, the notification of the latter issue to the Iridium customer service allow them to detect and fix a hardware bug on the modem 9603N.

5.2 Iridium's SBD's vulnerabilities

Besides power issues, the following lines will layout security-related conditions of the implemented technology. Security is rather a broad topic that was not included in the general scope of this MA; however, due to its relevance in information technology, some information is presented to be considered for further improvement.

Iridium security features are present in some modems, mainly those that implement the GSM-specified algorithm A3 for authentication security. However, the modem 9603N, which does not include a SIM card, lacks this characteristic.

To understand the vulnerabilities, Figure 26 shows where the oportunites for eavesdropping of Iridium bearer channels can be found. Namely L-Band and K-Band channels, basically up- and downlinks between ISU-SV and SV-Gateway (Iridium architecture shown in Figure 12). The SBD's guideline[28] mentions, nonetheless, the high demands of setting up eavesdropping devices, thus decreasing this vulnerability. For instance, demanding factors are technical and economical obstacles derived from large, continually changing doppler shifts, frequent inter-beam and inter-SV handoffs, time-division multiplexed burst mode channels, complicated modulation, etc. However, eavesdropping is still technically possible.

Additional security measures could be implemented by encrypting the data before transmitting to the destination point, and decrypting it at its arrival or usage-whenever it is needed. In this way, the exposed data would not be readable by the highly unlikely eavesdropper.

It is worth to mention, that security analysis is required to spot other vulnerabilities, that may be probably located on the user side, rather than in the technology implemented. For it is the exposed flank of the application. Furthermore, these vulnerabilities may be of simpler nature. For instance, access credentials to the email server or to the data server where the collected data will be stored.

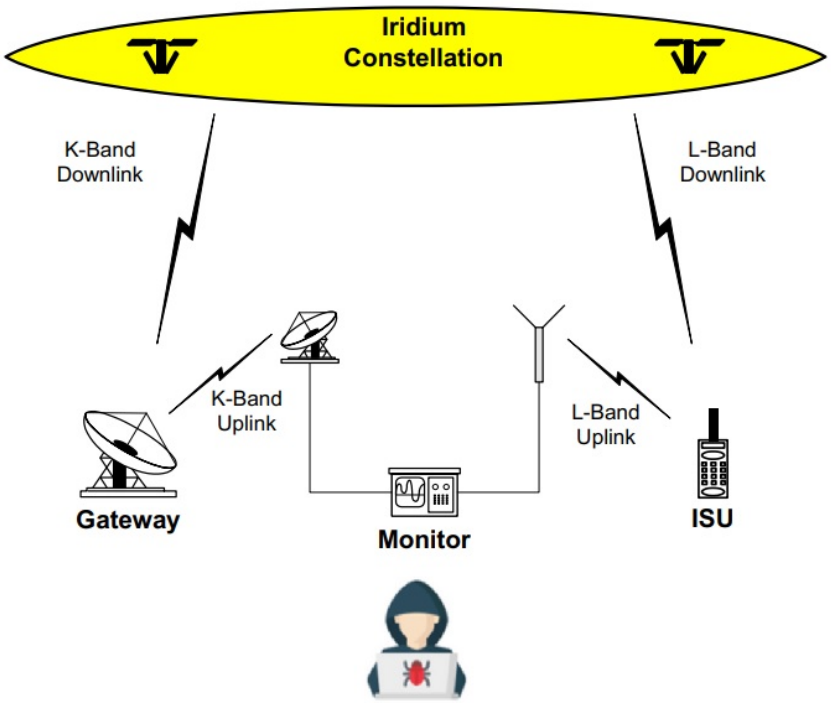


Figure 26: *The 9603N modem's vulnerabilities.*
The ISU is technically unprotected from eavesdropping at two transmission links; however, the practical implementation of a monitor device is highly unfeasible.

From a practical and simplistic perspective, the sole implementation of the data transmission as RAW byte data instead of ASCII based message, is already a layer of security. As the RAW data requires from the user to know the structure of the data packet, for example, at which offset each data is located or what data type and what is their meaning (recall Section 4.4.2).

5.3 Data integrity over on-board bus

Related to the overall reliability of the system and its parts. The data integrity of the messages among CAN nodes is highlighted. For instance, as mentioned in Section 3.4.2 that the full data packets over CAN are sorted correctly after identification of time-windows, and discarded whenever they are incorrect. This characteristic was observed to be working even if the node was disconnected-and reconnected- from the bus.

Furthermore, the CAN protocol defines no less than five different ways of detecting errors. Two of these works at the bit level, and the other three at the message level. This [43] tutorial can be a quick reference about this features.

- Bit Monitoring.
- Bit Stuffing.
- Frame Check.
- Acknowledgement Check.
- Cyclic Redundancy Check.

The CAN 2.0 specification also defines error confinement mechanisms that are aimed to provide tools for proper error handling, for instance, error counters at the physical layer and error states at the link layer[43]. These are dependent on software libraries, transceivers and their drivers. In this project, all five error checks are automatically integrated at physical and data link layer, for the transceiver and MCU meet the standard. Regarding the error counters, the python library has simple functions to detect change of states (recall the user-friendly interface Section 4.6), and the developed library has the interruption enabled for the same events. In this way, the base is set for exploring all these features.

5.3.1 CAN vulnerabilities

CAN bus is a communication protocol that is simple and reliable for communication, it is safe when it comes to its physical interface and the message handling, however, safe and secure refer to different concepts. Furthermore, secure communication protocols locate themselves mainly in high-level implementations (referring OSI model). Hence, it could be even said that this vulnerability is common in all transmission mediums; therefore, and only if required, the data involved in the exchange is the one to protect. This, however, would make only sense if the access to the CAN bus is exposed. Situation that is not present in the physically-enclosed drifter buoy. Nonetheless, as mentioned in Section 5.2, a formal security analysis is required to spot other vulnerabilities.

5.3.2 LoRa's commercial restrictions and outlook

As mentioned in the concept, all the exploration of LoRa technology is of extra value for this project. Nevertheless, allowing the *commboard* to interface with the transceiver and having the latter tested its basic communciation features, is crucial for any further step towards a full-integrated system (Section 3.7). The hardware and software tests with LoRa UDP-forwarder/end-node were paused and continued several times following priority adjustments. However, valuable information was

obtained even if some tests failed, for instance, while attempting to automate the decoding and response of a payload at the application server (Section 4.3).

Regarding task number 7 (refer Tab. 20), there are at least three different types of payload formatters: Javascript, CayenneLPP and Repository[44]. All of them depend on the application server. During the development of this project, it was however found that TTN server was porting its services to a newer version. Hence, limiting its compatibility for prototyping, specially with UDP-forwarders (original SEMTECH's prototyping method)[45]. In order to automate decodification of payload (data sent by a node), payload formatters at the application server are unavoidable. Nevertheless, TTN is not the only solution to an application server, namely a MQTT server (integrator) can be directly connected to the Network Server to carry out those tasks. Further research showed that there are different integrators, such as webhooks, storage integration, AWS IoT and LoRa Cloud[44]. Since MQTT can subscribe to uplink messages or publish downlink messages, it is proposed for future phases (2 and 3) to implement an instance of a MQTT server. Some examples for MQTT clients are Eclipse Mosquitto, Eclipse PAHO, HIVEMQ, MQTTBOX, MQTT Brocker within NODE RED.

Besides the communication with the standardized communication with the server, there are other interesting challenges to face. For example, whenever the *switching* feature (recall Section 3.5.2) is reliable in the next prototype, the logical step would be diving into the selection of a mesh network that keeps power consumption low. The latter is a goal that has been already spotted as a challenge, and lately being researched[10]. In a similar line, there has been an interesting approach to integrate positioning of devices via satellite into a LoRa WAN, thus, better allocating devices using a new Multiple Access Controller layer (SA-LoRaWAN). Furthermore, that strategy uses a "star of stars" topology, assuming a satellite-node that would behave as Lora-Gateway-Lora-Server. This approach was found to be in really early stages, as the foundations of the strategy have been implemented only in simulations with several constraints. Thus, a prototype, that includes in its hardware both technologies and ensures its efficient *switching* feature, would allow to try out such a strategy[11].

The overall result of the LoRa-related tasks led to the understanding that, it is indeed possible to have a device that behaves as both end-node and an UDP-forwarder. This, however, comes with some compatibility issues with commercial platforms, basically because the result would not be like any commercial device. Nonetheless, it is technically possible.

This panorama ends up not being a surprise, for the application itself is demanding from its mere integration purposes. The fine-tunning of such *switching-device*, should adjust, for instance, the two main libraries into two parallel tasks that can be

managed with a basic task scheduler. This scheduler plus the customized libraries will be part of the 1-Ch Gateway's running code on its own MCU (or any other board with a HopeRF RFM95[36]). The latter, in turn, should be synchronized with the firmware running on the *commboard* (or any other host board).

6 Conclusion

In the scope of this thesis, a concept for on-board and global (off-board) communication was designed and implemented, resulting in the prototype called *commboard*. This implementation fulfilled the requirements, highlighting those focused on upgrading the original main-board prototype, to make it able to exchange data over a robust and safe wired bus, namely CAN bus. This bus was extensively compared against other buses, like TIA-485, at the level of their physical layer, this is, the power-saving potential of their transceivers. The hardware includes an interface with a LoRa transceiver. The latter, in addition with a valuable exploration of the technology, led to a plan for the development of a *switching* feature and identification of usage scenarios, that would allow the prototype to behave as end-node and UDP-forwarder within a LoRa WAN.

Another highly important upgrade is the ability to interface with an Iridium's SBD modem, that by setting up all the necessary hardware needed to make it run without experiencing power-shortages during satellite transmissions. Highlight from its integration is the power supply, finely tuned to have low-noise output, in-rush current protection and a super-capacitors charger. The final implementation took place after developing a base software framework on top of a RTOS, allowing the prototype to be further optimized for power saving, while being compatible with other generic interfaces based on task schedulers. Moreover, the final results provide a prototype's power profile that was compared with data from other similar applications. It showed a relatively high power consumption, but being around the expected values; as these applications were researched to be power consuming. The *commboard* is now capable to be used to collect and transmit data over Iridium's satellite network, improving consequently the conditions of AMuSeD project for field tests, altogether with the main-board.

Finally, pieces of information were summarized containing an overall security status of the project, plus some strategies to decrease the power consumption.

7 Future work

This final section summarizes briefly some hints for the future work, for instance, improvement of the communication protocol in CAN bus and power saving recommendations for Iridium's modem.

7.1 CAN improvements

- Two-sided communication: To improve payload integrity and better synchronization between communication board and node, acknowledgment messages could be added on communication board's side. Through this mean, every time a node sends part of its data chunk, its counterpart would handshake and allow the next part. For this purpose, regular messages can be used, however, there is another type of messages that are also defined in the CAN specification but not usually used, namely RemoteTransmissionRequest Frames. Use the following article as introduction to this type of frame [43].

- Switching-off the transceiver: As commented in the implementation, the RST pin had a faulty connection. However, this pin allows the transceiver to enter to low-power mode.

- Data integrity at application level: Python is found to be flexible, hence, some of its core features make it different when it comes to fix some properties, namely size of data. This is because some data types are of dynamic length, for instance, floats and integers. When the external user starts to modify the provided test code in Pyhton, it could lead to flaws if lengths are not kept fixed. This is crucial as CAN data packet expects fixed sizes. Therefore, it is recommended to pay extra attention on double-checking sizes or limiting how much a variable can increment or decrement. This is a task given to the user. Sometimes a simple modulo check is practical to ensure a limit of a size (refer to implementation code in Python).

```
>>> 1022 % 256
254
>>> -1 % 256
255
```

7.2 Iridium's modem

A possible interesting feature is that the Iridium's SBD service makes it possible for one ISU to send a message direct to another ISUs, without the message passing to the Vendor Application. The second ISU destination IMEI must be programmed on-line by the VAR using the Iridium SPNet provisioning tool (refer to Figure 12 for the SBD's structure). However, only one mode in the modem (email or ISU-ISU) is permitted. Moreover, up to 5 IP socket addresses/ports for each MO session can be set up[27].

7.2.1 Optimizing the SBD integration

In addition to what was discussed in Section 5.1, the following information could result intersting for further works. When deploying units configured to transmit at regularly scheduled intervals, Iridium strongly advises that transmissions are randomized as much as possible, down to fractions of a minute. This will ensure the health of the network, improve security (transmission not predictable) and ultimately benefit all SBD users[28]. To optimize transmissions the following is recommended:

- Avoid transmitting at the top of the second, minute, hour and day[28].
- Space individual transmissions throughout the minute and ensure messages are evenly distributed over time[28].
- SBD response codes should be used to identify whether a transmission has completed and its status, for example, success\fail.
- Before transmitting, checking for good signal quality and network availability saves power. If the session fails, a backing off scheme starting with 60 seconds may be introduced. It is not recommended to continuously retry[28].

In Section 4.4.1 two important resources were referenced to give the reader an overall understanding of the transmission capabilities of the satellite modules. However, the original test scenarios are too specific to consider them as a one-to-one reference. Hence, more numerous tests can be set with the communication board. For instance, $TIME\_BETWEEN\_MO\_TRANSMISSIONS$ directive in the header file of the iridium library ($amused\_iridium.h$) can be used to queue a specific number of mobile originated messages (70-bytes long) and compare the number of emails received in a certain amount of time. Varying the scenarios would lead to a more

reliable description of the transmission capabilities of the prototype. Special considerations should be taken into account regarding the SBD Service Plan and the available amount of data.

A Appendix A

A.1 Requirements

The following table summarizes the contents of Section 3.1 into numbered items, such that the reader can relate them within the different parts of this document.

Table 14: *Full list of requirements.*

Index	Short title	Description	Value (if applicable)
1	PCB upgrade	Redesign of previous PCB using Eagle and its manufacturing.	-
2	PW input protection	HW feature to protect from reverse input voltage and current, avoiding destruction of prototype after accidental flipped-connection.	12 24 V expected
3	PW supply	HW change including a PS for larger input voltage.	Input: 12 V Output: 3, 5V
4	PS noise profile	Conformity with PS profile of modem 9603N.	Evaluated by functioning
5	ISU PW charger/protection	HW feature that complies with ISU SBD's PW requirements (Section A.1.1).	-
6	ISU HW interface	HW Serial interface to communicate with modem according to ISU specifications.	RS-232 @ 3V3
7	2nd modem HW interface	HW commuication interface with a second satellite module if external commuication with Kinéis is needed.	Serial
8	Satellite communication	SW means for sending data from *commboard* to an on-land device.	-
9	Data rate over satellite	Minimum data rate according to application (40 bytes each 5 minutes).	1.066 bps
10	LoRa HW interface	HW communication interface with a LoRa device to explore the technology.	Serial expected
11	LoRa communication	SW means for sending data from one node to a gateway (optional).	-

12	Global communication	Proposal of a fail-safe strategy for global communication optimized for power saving and distance transmissions.	-
13	On-board HW communication interface	HW upgrade from I2C on-board communication to achieve a certain degree of robustness.	No baseline provided.
14	On-board communication	SW means of data transmission over a robust bus to show benefits against I2C.	-
15	On-board fail-safe communication	Strategy to include fail-safe design features in on-board communication.	No baseline provided.
16	Multi-node on-board communication	Addressability of multiple nodes. I2C as basic reference.	-
17	Data rate for on-board interface	Data rate based on standard I2C-bus (standard mode).	100 kbps proposed
18	Compatibility	Design takes into account parallel developments within AMuSeD project, namely the SW "generic interface".	Details unknown
19	SW libraries	FW modular libraries to be added into existing ATMEL projects.	Embedded C
20	PW Libraries	Part of the library should contain some functions aimed to PW control of external devices.	-
21	Validation	Tests to ensure the existence of a working prototype.	-
22	User-friendly	The final prototype should take into consideration that the final user has access to external nodes tha communicate with the main-board or *commboard*.	Arduino platform as reference

A.1.1 Satellite modem's power requirements

Providing a high quality power supply is of high importance for ensuring a correct setup of a SBD session. This is a consequence from the increment current consumption during SBD transmissions. Hence, in addition to general requirements 3

Table 15: *Power requirements as stated in 9603N technical specification. More specific were critical for PCB upgrade. Note: The average power consumption will vary depending on the view of satellite constellation from the antenna.*

Parameter	Value
Supply Input Voltage Range	5.0V DC +/-0.2V**
Supply Input Voltage Ripple	< 40 mV pp
Typical Power Consumption at +5.0 VDC	*Value*
Idle Current (average*)	34mA
Idle Current (peak)	156mA
Transmit Current (peak)	1.3 A
Transmit Current (average*)	145mA
Receive Current (peak)	156mA
Receive Current (average*)	39mA
SBD message transfer - average current*	158mA
SBD message transfer - average power*	<= 0.8 W

and 5 (Tab. 15), this concept complies with the following points as stated in 9603N technical specification.

- The supply voltage droop over for a 8.3 ms burst of 1.5 A current should not be more than 0.5 Volts.
- The power supply should limit the in-rush current to 4 A maximum.
- The power source shall provide for over current protection in case of device malfunction
- The supply noise should be less than the following limits:
  - 100 mVpp from 0 to 50 kHz.
  - 5 mVpp at 1 MHz measured in 50 kHz bandwidth.
  - 10 mVpp at 1 MHz measured in 1 MHz bandwidth.
  - 5 mVpp above 5MHz measured in 1 MHz bandwidth.

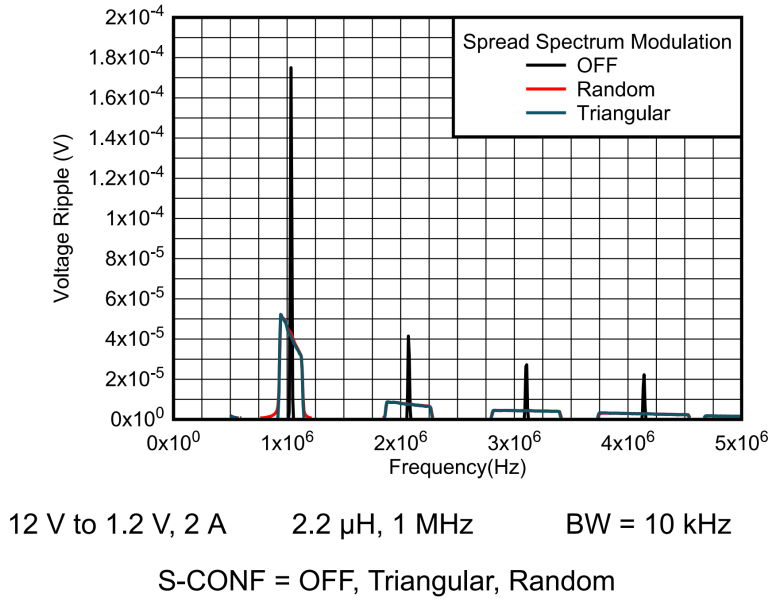


Figure 27: *Effect of triangular SSM on output voltage ripple[37].*
In this prototype the repetitions would be spaced 2.2MHz from each other.

A.2 Power supply details

A.2.1 DC-DC buck converter configuration

This section shows relevant technical information about the configuration of the DC-DC converter that was selected to substitute the previous IC in the main-board. The latter in response to two requirements, first, the high-quality power source that demands the ISU modem, and the IC-shortage.

The theoretical calculations for an output of 5V are located in the range of 6 V to the 17 V. The selection of this buck converter corresponds its efficiency and the configurable low-noise output. This characteristic is relevant because, if the original DC-DC converter (RP509N331B) would have been placed, the board would have been under the effects of two high frequencies. That is, 2.2 MHZ switching frequency of the first DC-DC converter plus the forced PWM at 6MHz of the second DC-DC converter.

In order to reduce the ripple effect on output voltage, a triangular spread spectrum modulation was configured. Notice that the ripple and the repetitions of the noise at higher frequencies is significantly lower in Figure 27.

Another advantage of this modulation is that a second stage filter L-C is not needed.

Part	Manufacturer	Notes	Availability
Ideal diodes			
LTC4376	Analog devices	See Demon Manual DC2706A	N
	Analog devices	See Batery Charger with recerse Voltage protection AN-171	
LM74700-Q1	Texas Instruments	Ideal diode Reverse battery protection	N
LM66100DCKT	Texas Instruments	Hot Swap Voltage Controllers 1.5-V to 5.5-V, 1.5-A, 0.5-uA IQ ideal diode with Integrated FET 6-SC70 -40 to 105	Y
LM5051MAE/NOPB	Texas Instruments	Hot Swap Voltage Controllers Lo Side OR-ing FET Cntlr	Y
LM74801QDRRRQ1	Texas Instruments	Power Management Specialised - PMIC 3-V to 65-V, automotive ideal diode controller driving back to back NFETs 12-WSON -40 to 125	y
LTC4372CDD#TRPBF	Analog devices	Power Management Specialised - PMIC Low Icc Ideal Diode Controller w/SHDN	Y
LTC4376IDHD#PBF	Analog devices	Power Switch ICs - Power Distribution 7A Ideal Diode with Reverse Input Protection	Y
Hot-swap controller			
LT1641	Analog devices	Reverse Battery, In-rush current and reverse current protection, see 48V Hot swap circuit blocks	N
ADM4210	Analog devices	Only In-rush current protection	N
TPS2421-x	Texas Instruments	20v 5A Fet hot swap	N
E Fuse			
TPS2660x	Texas Instruments	Efuse with Reverse input polarity protection	N
TPS26600PWPR	Texas Instruments		N
LTC4211CMS8#PBF	Analog devices		Y
LT1640HCS8#PBF	Analog devices	Hot Swap Voltage Controllers Hot Swap Controller for -48V apps	Y
STEF4SPUR	STMicroelectronics	Hot Swap Voltage Controllers Electronic fuse 4V 3.6A 40mW 4.5V	Y
LT4256-1IS8#PBF	Analog devices	Hot Swap Voltage Controllers Hot Swap Contr. +48V Latch-Off Mode	Y
STEF05DPUR		Hot Swap Voltage Controllers POWER MANAGEMENT	Y

Table 16: *Selection table with different protections.*
An ideal diode controller was found to met efficacy, cost and availability criteria.

A.2.2 Reverse polarity and current protection

This section shows only a brief summary of the selection process (Tab. 16). Basically there are 5 types of protections: schottky diodes, P-CH Mosfets, N-Channel Mosfets, ideal schottky diodes and efuses (Figure 28). These prototype uses a power management IC that complies with the ideal diode characteristics[46].

A.3 PCB tests

This appendix contains the summary of the main validation of the PCB designed and manufactured within the scope of this Master project.

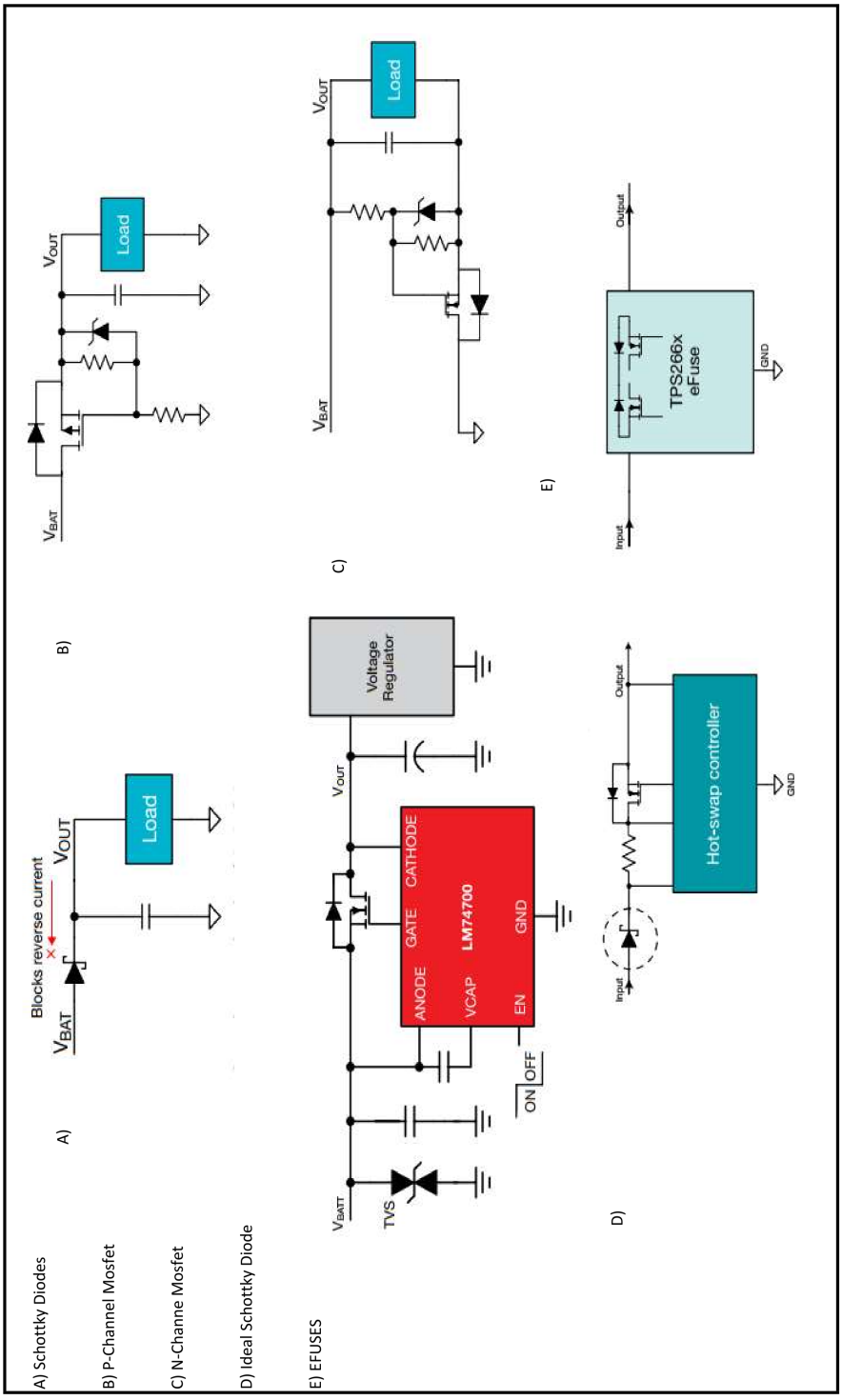


Figure 28: *Types of reverse polarity protections.*
This prototype implements an ideal diode controller.

Table 17: *Record of the validation tests for basic features.*

Tests		Result	Date	Notes	Conclusions
LEDs		**OK**	**2021.10.04-10.05**	-	Correctly working.
	General 3v3	OK	-	-	-
	Charging	OK	-	-	-
	Charged	OK	-	-	-
	Control MOSFET	OK	-	-	-
MCU base functionalities		**Partly OK**			
	Vcore voltage	OK	2021.09.29	-	Right 1.2V measured at Buck converter
	SWD correct pinout	OK	2021.09.29	-	Correctly working.
	SWD read of basic MCUs ID through the ICE Debugger	OK	2021.10.04	Device signature read with Microchip Studio 0x61810006	Memories can be accessed
	Basic programming over SWD	OK	2021.10.04	Erasing of Chip OK	Basic programming of code OK
	Basic debugging over SWD	OK	2021.10.04	Stepping from delays to delays now tested.	Debugging basic integration OK
	Test all routed GPIOs	NOK	**2021.10.05/12.xx**	From all GPIOs. The one connected to RS-pin of CAN transceiver does not change the state. All others OK	Failure in PCB trace. Solution>>Unsolder slope-control variable resistor, soldering in its place a cable for manual turn-on and off for the transceiver. There is no slope-control.

Table 18: *Record of the validation tests for PW features.*

Test	Result	Date	Notes	Conclusions
Reverse polarity connection	Partly OK	2021.09.29	OK-Polarized >> Voltage went through correctly: Vin measured in the DCDC converters. NOK-Polarized >> From -12V, -1.4 V were measured at DCDC inputs, ideally expected were 0v. This may be harmful though.	Not enough information to affirm that the reverse polarity protection fully works as there were still -1.4 Voltage coming through the input. However, the prototype seems not to be affected by the remaining -1.4V.
DCDC to 5V	OK	2021.09.29	One Res was found to be mistakenly ordered, so it needed to be replaced in both DCDC. Once after change 5.1 V can be measured at DCDC output. Behavior with different input voltages is not included in this first test.	DCDC works if further information about filtering and quality of output is needed, specific tests need to be further scheduled.
DCDC to 3.3V	OK	2021.09.29	Same problem found with RES, solved and tested the basic Voutput.	DCDC works and output is measured to be within the 3.3V range. Filtering quality and behavior depending on different input voltages is to be tested separately.
PW Charger/In-Rush current protection HW features	OK	2021.12-2022.01	The PW charger notifies the change of state to fully-charged by setting on the LED charged. ISU does not present any PW related issue that keeps it from transmitting SBD messages. Hence, base PW features OK.	Correctly working.

Table 19: *Record of the validation tests for communication features.*

Tests	Result	Date	Notes	Conclusions
CAN transceiver	**Partly OK**			
Differential pairs	OK	2021.12-2022.01	Transmission of data is possible. Tested until CAN nodes were correctly setup.	Correctly working.
Slope-control pin	NOK	2021.12	See the note in "Test all routed GPIOs ".	Slope control is an advance feature based on HW and only available in this transceiver to control the noise rejection in the bus. The prototype does not uses this feature, in turn, avoid PW-consuming high baud rates.
Iridium basic features	**OK**			
GPIO pin signals at SMD connector	OK	2021.11	This test needed an extra PCB being manufactured 2 weeks after the first one.	Correctly working.
Serial communication pins	OK	2021.12	This test needed an extra PCB being manufactured 2 weeks after the first one.	Correctly working.
LoRa interface	**Partly OK**			
Mounting dimensions	OK	2021.10	-	Correctly working.
PW delivered to LoRa node	OK	2021.10	-	Correctly working.
GND common with LoRa node	NOK	2021.10	Floating pin	Short-term solution. Manually bridged to next common ground point.
Serial communication interface	Partly OK	2021.11	Pins at the LoRa development board are shared with those of its USB interface. Hence, when being used along with a PC Host, the LoRa board cannot interface with commboard.	Short-term solution. All configuration and communication tests with LoRa were done de-attached from commboard. Fully integration with commboard was re-prioritized and moved out of the scope of the MA, to leave space to Iridium and CAN tests.
Other communication interfaces	OK			
I2C @SERCOM ports	OK	2021.12	Tested with external I2C node.	Correctly working.
UART @SERCOM ports	OK	2021.12-2022.02	All tested while debugging FW.	Correctly working.

Table 20: *HW and FW used for off-board LoRa tests.*

Item	Description	Comment
1	Communication board	Used as HW interface with LoRa module
2	PC Host	Used for programming and commission of LoRa server. Win 10.
3	ESP32 LoRa 1-CH Gateway	Affordable development board with HopeRF RFM95 LoRa transceiver. This development board got commercially retired during final stages of this project.
4	2.8dBi LoRa Antenna - SMA male - 868 MHz and IPEX adapater	Antenna and accessories for tests.
5	Single Channel LoRaWAN Gateway v 6.2.8	Library compatible with ESP32 LoRa 1-CH Gateway[47]
6	Arduino-LMIC library v 1.5	Library compatible with ESP32 LoRa 1-CH Gateway to make it behave (with some constraints) as an ABP device Type A. This library is now deprecated, see MCCI LoRaWAN LMIC Library[48].
7	TTN v2 and TTN v3 (transition)	Official LoRa web server from The Things Network. The now final version of TTN v3 does not integrate modified ABP devices nor UDP forwarders, as they are out of standard, the latter is the case of the ESP32 1-CH Gateway.
8	PlatformIO/Arduino studio	IDEs for library and HW tests

A.4 LoRa extra info

A.4.1 Activities and material for tests

A.4.2 Off-board communication strategy

This is the detailed state machine for the communication strategy.

Table 21: *Carried out tasks to build up the off-board communication concept.*

Item	Goal	Result/Comments	Instance involved	Distance and data size
1	Selection of LoRa transceiver	HopeRF RFM95 transceiver identified as the most flexible one. ESP32 LoRa 1-CH Gateway selected.	NA	NA
2	Setup LoRa Node	Successful. Working library identified.	End-node	NA
3	Setup LoRa UDP-Forwarder	Successful. Working library identified.	UDP-Forwarder	NA
4	Monitor whether Gateway receives Node Data (Local)	Successful. Local server connected to host PC.	Both	1-4m, 8-16 bytes
5	Setup of TTN Server	Successful. Web server setup.	Web Server	NA
6	Monitor whether Gateway receives Node Data (Web-Server)	Successful. Established connection with the UDP-forwarder as a non-compliant Gateway on TTN	Web Server and UDP-Forwarder	1-4m, 8-16 bytes
7	Decode automatically the payload and send it back to end-node as response payload	Failed. TTN v3 features are not compatible with non-standard Gateways and ABP devices. End-node message is not decrypted, however payload can be decrypted manually.	Web Server and end-node	1-4m, 8-16 bytes
8	Redefinition of concept according to results	Successful. Phases as stated in the concept.	NA	NA

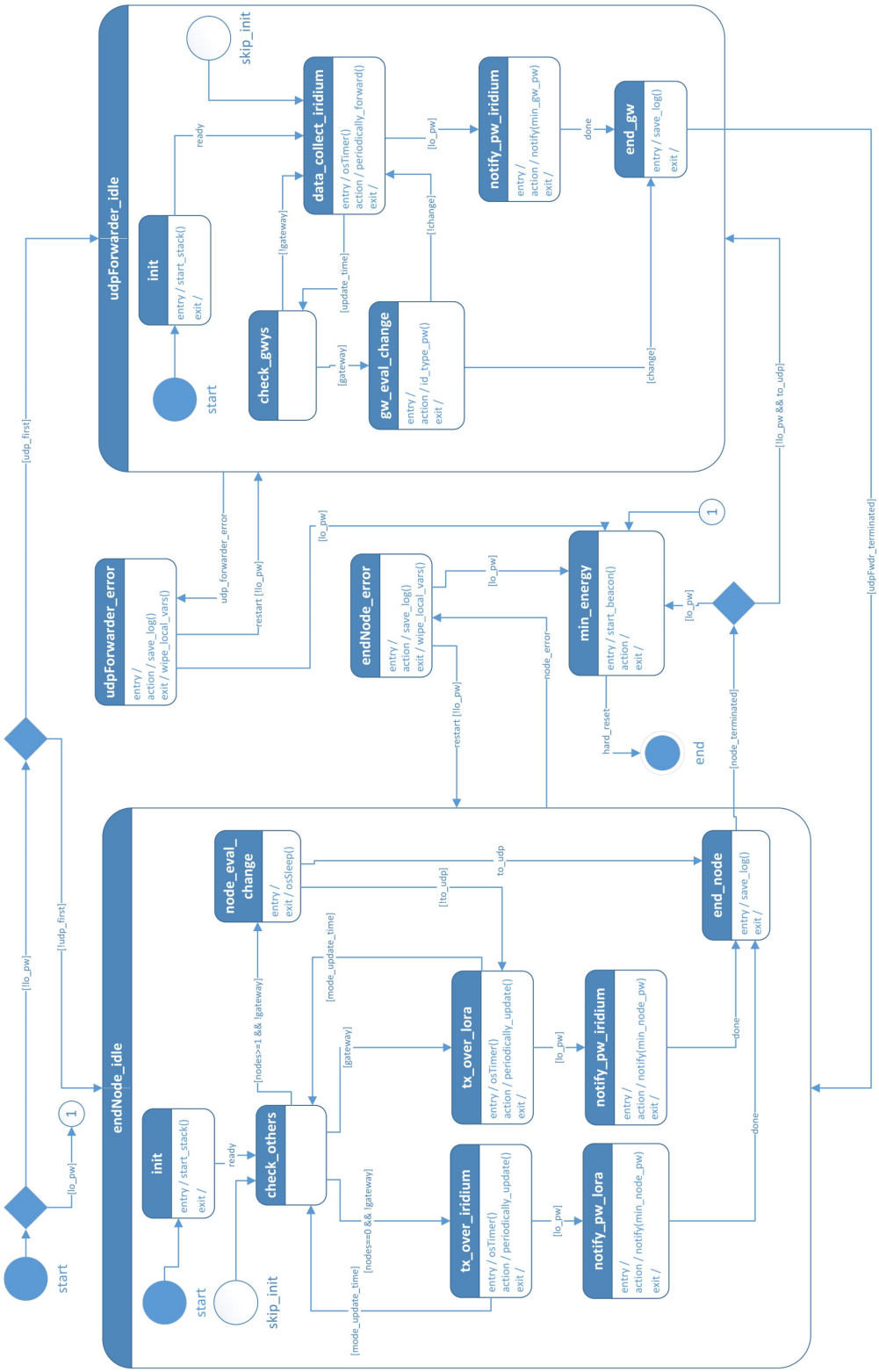


Figure 29: *This SM represents the off-board communication strategy that assumes both SBD and LoRA technologies working flawless in the drifter buoys.*

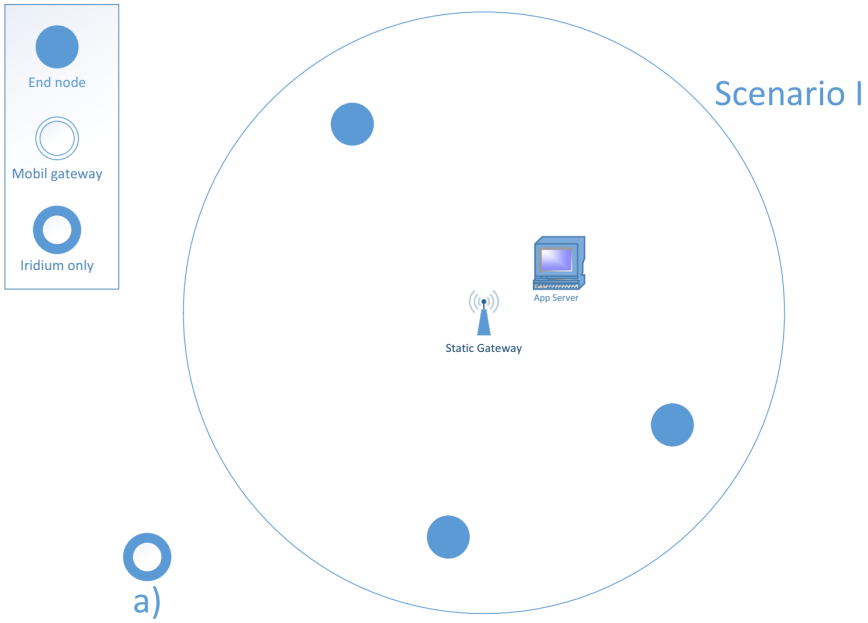


Figure 30: *An isolated end-node sends data over Iridium's SBD.*

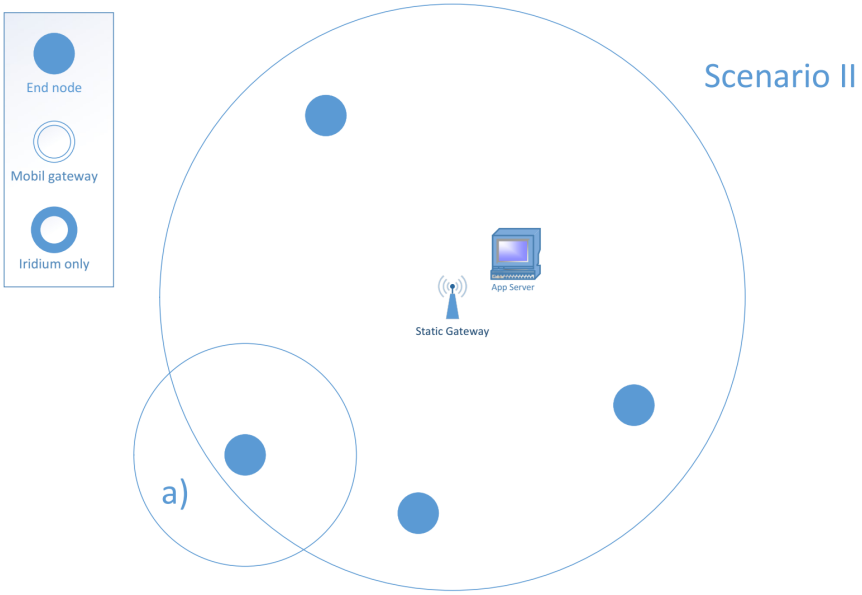


Figure 31: *An isolated end-node joins a gateway-like device.*

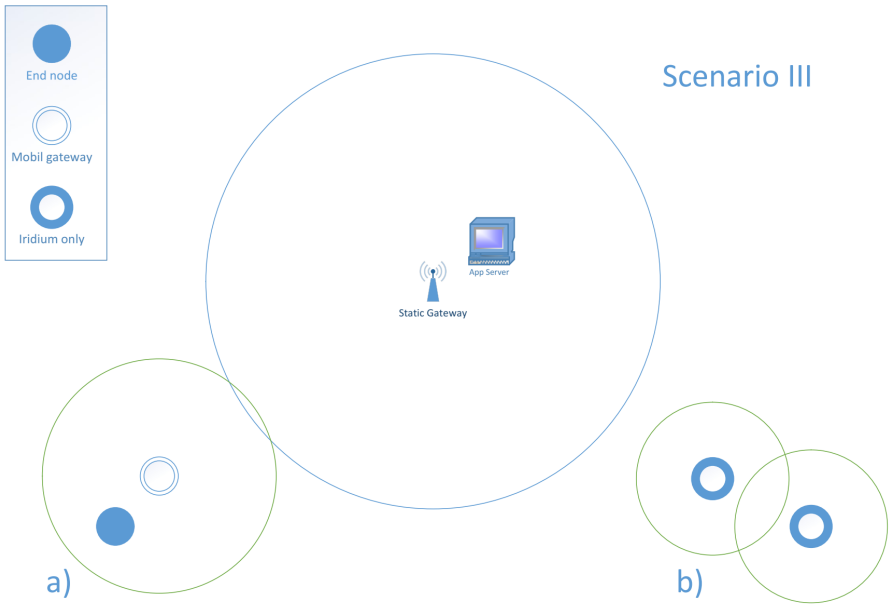


Figure 32: *End-nodes decide whther change to udp-forwarder*
a) one of the end-nodes changed into udp-forwarder while the second keeps transmitting its data. b) two isolated nodes coming into range, such that one of those would end-up becoming udp-forwarder.

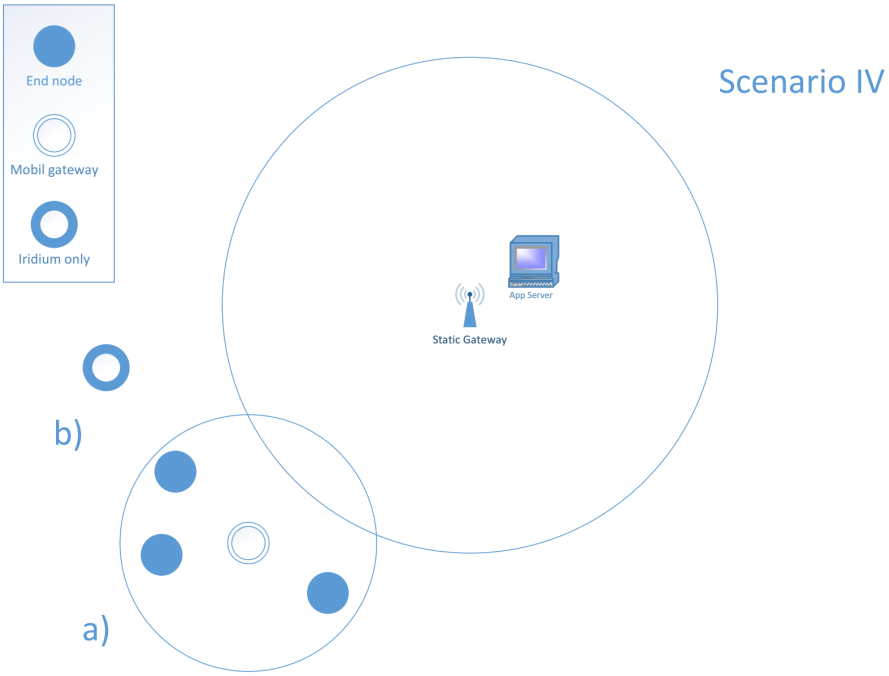


Figure 33: *End-nodes in, and out the sight of an udp-forwarder.*
a) an udp-forwarder would keep collecting data from as many as reachable end-nodes. b) an end-node that goes beyond sight would end-up being isolated, thus turning off its LoRa features and transmitting its data over SBD.

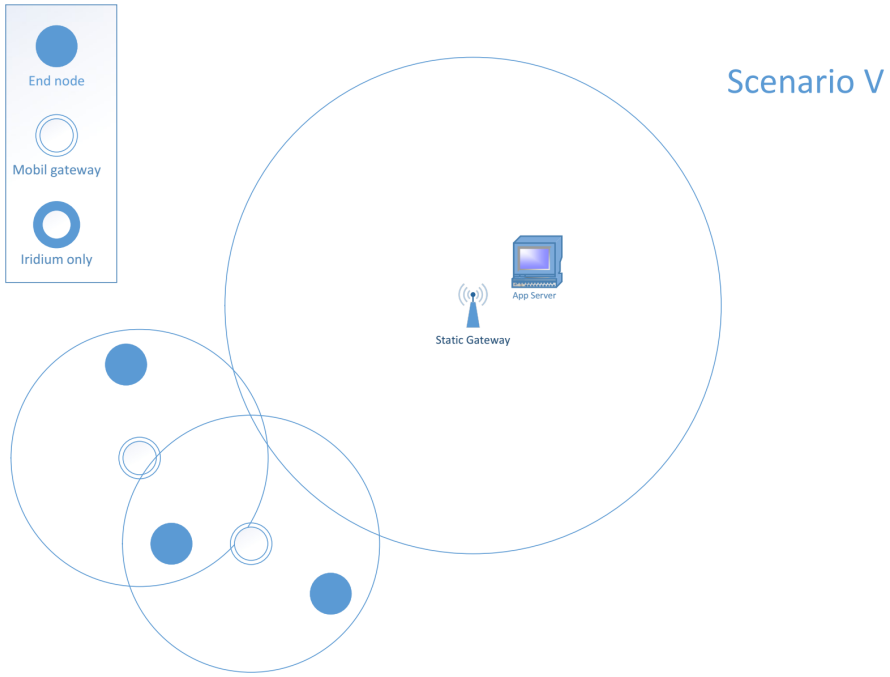


Figure 34: *Two udp-forwarders encounter each other.*
This would trigger a decision algorithm, such that one of them would eventually turn into an end-node.

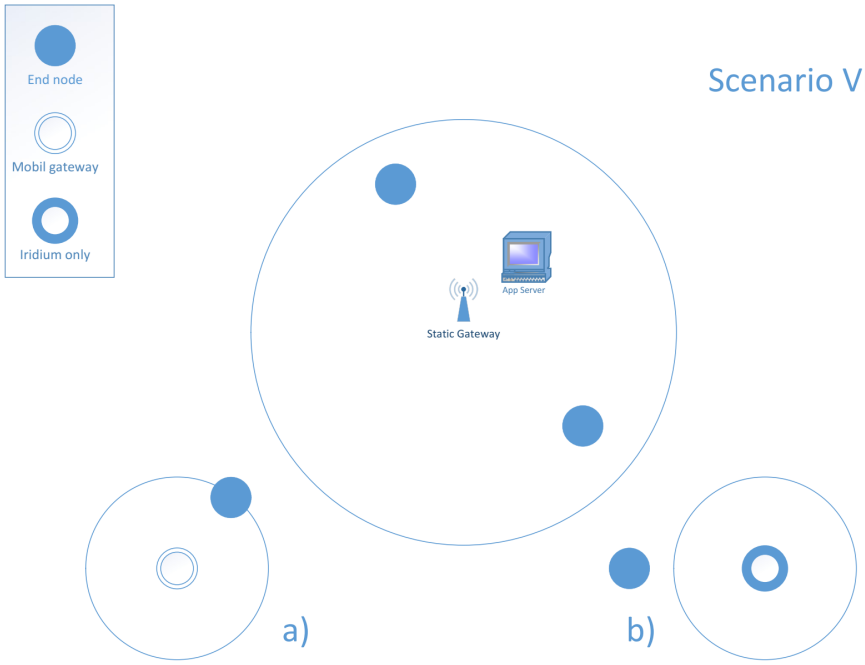


Figure 35: *Isolated udp-forwarder.*
a) an udp-forwarder remains so until the last end-node is out of sight. b) once an udp-forwarder is isolated, it will turn off its LoRa features and would send its data over SBD. Any end-node isolated would be in scenario I.

To ISU (from DTE)	To DTE (from ISU)	Description
AT*F|		DTE tells ISU to flush pending writes to Eeprom and waits for completion
	OK	
<binary transfer>		DTE may now safely disconnect the ISU power supply.

Figure 36: *Powering down the ISU 9603.*
This function is available as an irApp function. The ISU flushes any pending EEPROM writes before shutting down[38].

To ISU (from DTE)	To DTE (from ISU)	Description
AT&K0		Disable RTS/CTS flow control
	OK	
AT+SBDMTA=1		Enable SBD ring indications
	OK	
AT&W0		Store the configuration as profile 0
	OK	
AT&Y0		Select profile 0 as the power-up default
	OK	

Figure 37: *Setting the ISU's Default Configuration.*
This configuration has to be done over a serial interface with control flow activated and specifically 3V3 voltage level compatibility. After it has been set, the two-wire configuration can be safely used[38].

A.5 IRIDUM extra info

A.5.1 Power cycling and configuring default configuration

Long times could arise while setting up the Iridium Module for the first time, related mainly to possible inconsistencies with the default serial communication. To avoid this, the following modes were configured to be part of the default configuration profile (Figure 37). Basically they allow the module to use only UART working mode (3 pins mode, RX, TX, GND) to bypass any miss function with the DATA/FAX mode due to some floating pins and HW-manufacturer-related conditions. Note that the SBD Ring notifications is not enabled, as the prototype works only with MO[38].

```
>>  AT&D0
>>  AT&K0
>>  AT&F0
```

To ISU (from DTE)	To DTE (from ISU)	Description
AT+SBDREG?		Query the ISU registration status
	+SBDREG:0	ISU is detached, i.e. un-registered
AT+SBDREG		Tell the ISU to register for automatic notifications
	+SBDREG:2,0	ISU is now registered
AT+SBDREG?		Query the ISU registration status

Figure 38: *Useful commands while testing the SBD.*

A.5.2 Evaluating the signal strength indicator

Evaluating the strength of the signal is a good practice while testing and optimizing the application. This may have large positive impact for power saving (requirement 20), for repeated failed attempts of data transmission cause an unnecessary loss of energy. For this variable to be test, the appLib function "signalCheck" can be used in future iterations of the developed firmware.

Consider the following information after having sent the AT+CSQ command (by calling up "signalCheck"):

Signal strength of 0 indicates the following.

- Either the antenna is not connected properly;
- The signal loss in the antenna cable, connectors is too high resulting in excessive signal loss.
  - Ensure that the cable length, type and design frequency is appropriate for Iridium.
    - Iridium operates between 1616 and 1625 MHz
    - The entire RF loss of the antenna installation (including cable, connectors and lightning arrestors) shall not exceed 3dB.
- The antenna does not have a clear view of the sky. The optimal view is 360 degrees of azimuth and above 8 degrees of elevation.
- Interference from a local high power L-Band transmitter. For instance, another satellite telecommunications provider.

Signal strength of 1 or higher should permit an SBD call to be made. Check signal strength over a period of 10 to 20 minutes to characterize the received signal strength for your antenna location. If you consistently see:

- A signal strength of 4 or higher you appear have a good installation.

- A varying signal strength ranging from 0/1 to 4/5 then your antenna location appears to have partial line of sight blockage. Identify the blockage and determine if you can relocate the antenna.

A.5.3 Example for trnsmission a simple MO

The following is a real example of transmitting a simple "hello" string from the prototype. HOST and ISU labels were added to ease the identification of the message origin. The simple steps are taken from [28], however, the responses are those from our prototype. This could help while testing new response handlers, specifically for cases where extra SBD-related information is required to be identified.

```
HOST:          AT+CSQ..
ISU:               ..+CSQ:5....OK..

HOST:          AT+SBDWB=5..
ISU:               AT+SBDWB=5....READY..

HOST:          68 65 6c 6c 6f 02 14 0d 0a
ISU:               ..0....OK..

HOST:          AT+SBDIX=5332.746,+00101.958..
ISU:               AT+SBDIX=5332.746,+00101.958..
ISU:               ..+SBDIX: 0, 2, 0, 0, 0, 0....OK..

HOST:          AT+SBDD0..
ISU:               AT+SBDD0....0....OK..
```

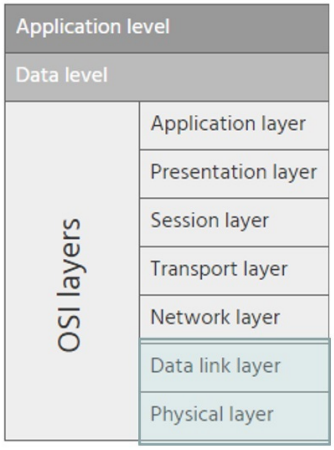


Figure 39: *OSI model characterizes and standardizes communication functions Its goal is the interoperability of diverse communication systems with standard communication protocols[49].*

A.6 CAN extra info

Control Area Network (CAN bus)

CAN bus is a communication bus protocol that started originally as a vehicle bus. However, its commercial availability of standalone CAN transceivers and more families of microcontrollers with integrated CAN support, make it an alternative solution in and out of industrial applications. CAN bus corresponds to the physical and data link layer of the OSI model, and it has been standardized already in ISO 11898-1, -2 and -3. Its definition covers multi-node addressability, bus arbitration with prioritization, collision control and error detection features. Some standardized high-layer protocols based upon it are CANopen, DeviceNet, EnergyBus, ISO-TP, NMEA 2000 among others.

A.6.1 Bus selection

A criteria-based matrix technique was used to back the selection of the on-board communication bus, as there were various options that complied with one or more of the requirements (see Section 3.4). The criteria taken into account is summarized in the Tab. 2.

- Robust transmission medium refers to the overall performance of the bus against mounting conditions, for instance, cabling longer than 10 cm, device

physically exposed to environment, no maintenance available after deployment, etc. Hence, bus technologies that are typically used in harsh industrial environments were preferred.

- Multi-node with basic QoS refers to the capability of connecting several devices into a bus, while providing a strategy to avoid or handle message collisions. Quality of service (QoS) is a term to describe the general performance of a service, in this case communication from node to host, number of nodes and type of data that can be sent.

- Overall power consumption refers to empirical results when implementing this interfaces in physical circuits. Moreover, their possible power-saving features at HW and SW level.

- Simplicity of an end-node Since user-friendliness is a requirement (requirement number 22), the end-node concept was kept as simple as possible. Hence, opting for technologies and user-interfaces that are already tested and can be easily implemented played an importan role.

- Driver/Library ease of usage Less complex, hence, flat libraries are preferred to keep computation power low saving consequently power. For example, Transport-Layer-based protocols have larger overhead in the code implementation than Data-Link-layer based ones. The latter due to the number of services available. Moreover, more complex libraries take longer to be correctly set up by developers and commonly depend on experience.

- Market availability refers specifically to the IC shortage global issue that has been affecting the supply chain in the last year. Some ICs that are widely used due to their practical convenience, price, their technology, have face critical shortage in their production, even though, they started the year with large stocks. Hence, replacements are urgently searched by numerous manufacturers around the world, leading even to change of technologies in some applications.

- Active technology There are buses that have been around the industry for more than two or three decades. However, as they evolve to so-called legacy technologies, their compatibility with current or next generation devices is not promising. Furthermore, this criteria also focus on the degree of accessability to development tools; a good access would eventually lead to more information resources easing the learning curves when exploring technologies. LIN, for instance, was a wide-spread protocol in the automotive industry that evolved to be only an extension of CAN buses. However, in the current times, even simple peripherals in an autos demanded higher QoS for communication, therefore LIN transceivers became less common.

Other common factors indirectly considered within the above mentioned criteria were as follows: ease of implementation, potential problems during implementation, potential effects on users or developers, and time until solution is fully implemented.

A.6.2 CAN data structure of the prototype

Tab. 22 shows the structure of the data packet being transmitted from each CAN node requiring share data with the *commboard*.

A.6.3 Expansion of a CAN data packet

For sake of simplicity, the CAN data packet has a fixed length. However, some configuration bytes were foreseen for an eventual expansion of the data packets. For instance, the current implementation corresponds to v1 and a length of 32 bytes. If a longer packet needs to be defined, the developer can make use of the version byte (Tab. 22) and would need to modify the CAN library to read out the contents of the first bytes. consequently, he would identify and handle different versions accordingly. It is then recommended to expand the data packet in factors of 32 bytes to make the transmission straightforward, while complying with the 8-byte segments standardized in CAN 2.0A.

Configuration bytes were foreseen for a possible expansion of the data packets, refer to first 7 pieces of data in Tab. 22.

A.6.4 Selection of CAN transceiver

Table 22: *The structure's concept allows up to 6 values (float and integer types) to be transmitted from each node. Hence, up to 6 different sensors can be mapped. This logic structure can expand in factor of 32 bytes to allocate more data.*

Item	Size in bytes	Accumulative size	Default Value	Comments
Configuration				
Node ID	1	1	0x11	Default or given by the user
Version of data chunk	1	2	1	Do not change. 1: 32 bytes 2: 64 bytes (not implemented) 3: 128 bytes (not implemented)
Flag for integers	1	3	1	1: True (Default) then they will be read on comm board's side, 0 : False they wwill be discarded
Number of integers to send over satellite	1	4	3	Dependent on Version of data chunk. Default and maximum for Version 1 is 3 integers.
Flag for floats	1	5	1	1: True (Default) then they will be read on comm board's side, 0 : False they wwill be discarded
Number of floats to send over satellite	1	6	3	Dependent on Version of data chunk. Default and maximum for Version 1 is 3 floats.
Reserved	2	8	-	For alignment or future configuration flags
Data				
Integer 1	2	10	0	
Integer 2	2	12	0	
Integer 3	2	14	0	
Reserved	6	20	-	For alignment or future expansion
Float 1	4	24	0	
Float 2	4	28	0	
Float 3	4	32	0	

Table 23: *Selection table for different CAN transceivers available in the market.*

CANbus Transceivers

	ESD/EMI protection	CAN compliment	PW-Sleep modes	Speed	Short-circuit protection	Op current	Vcc	Price	Availability	Distributor	Quantity
LT1796	Y (IEC-1000-	Y (ISO 11898)	Y (High Impedance Outputs When Off o	Y			4.75-5.25V		Y	LT1796IS8#PE	360
				125kbps half		23	4.75-5.25V	3.38	Y	Digikey	2400
MAX3058ASA+T	Y	**N**	Y Standby mode (15	500kbps-1Mbps	Y	40		1.6	Y	700-MAX3058	2987
								1.26	Y	Digikey	**236**
MAX3059ASA+	Y	**N**	**N**	1Mbps	Y	40	4.75-5.25V	1.6	Y	700-MAX3059	**369**
MAX3059ASA+T	Y	**N**	**N**	1Mbps	Y		4.75-5.25V	1.65	y	Digikey	5875
MAX13052ESA+	Y	Y (ISO 11898)	**Y + Silent mode (25**	500kbps-1Mbps	Y + SPLIT mode	**30**	**4.75-5.25V**	3.33	Y	700-MAX1305	1352
						30	**4.75-5.25V**	2.63	Y	Digikey	875
MAX13052ASA+	Y	Y (ISO 11898)	Y + Silent mode (25	1Mbps	Y	30	4.75-5.25V	3.19	Y	Digikey	**197**
MAX13051ESA+	Y	Y (ISO 11898)	**Y (10uaA)**	500kbps-1Mbps	Y + AUTOBAUD	**30**	**4.75-5.25V**	2.81	Y	Digikey	537
SN65HVD235QDRQ1	Y	Y (ISO 11898)	Y (200uA)	1Mbps	Y	**18**	3-3.6	3.35	Y	595-SN65HVE	65
				1Mbps (half)	Y	**18**	3-3.6	3.3	Y	Digikey	1410
SN55HVD251DRJR		Y (ISO 11898)	**Y (200uA)**	1Mbps	Y	**30**	4.75-5.25V	2.56	Y	595-SN55HVE	1836
XR31235ED				1Mbps						701-XR31235	104
SN65HVD230		Y (ISO 11898)	**Y (standby)**	1Mbps			3-3.3	2.67	Y	595-SN65HVE	10000
SN65HVD231		Y (ISO 11898)	**Y (sleep)**	1Mbps			3-3.3	2.7	Y	595-SN65HVE	2223
HVDA553QDRQ1				1Mbps		50	4.75-5.25V	2.23	Y	595-HVDA553	**223**
									Discontinued	Digikey	**0**
XR31235ED				1Mbps		0.2	?	2.43	Y	701-XR31235	**243**
						0.2	3V ~ 3.6V	2.41		Digikey	**93**
TLE7250GXUMA1										NA	
IFX1040SJXUMA1										NA	

A.7 FSM implementation with RTOS

The functions are the native ones as there is no abstraction layer in between, even though, abstractions layers are useful and tend to be simpler, they may add overhead to the application code, being that a relative disadvantage. Additionally, the implementation of an API for a RTOS, for instance CMSIS, would have required some extra work adjusting the initial code created by ATMEL START tool. However, CMSIS is still an option to simplify the introduction of a RTOS in prototypes[50].

Generic tools that are common on RTOS were used to synchronize and handle the asynchronous events generated by peripherals or tasks:

- TaskNotifyTake
- TaskNotifyGive
- TaskSuspend
- TaskResume
- YieldFromISR

A.7.1 About FreeRTOS prioritization scheme

RTOS ports that support interrupt nesting have the concept of a maximum system call (or maximum API call) interrupt priority. Interrupts that are above the maximum system call priority are keep permanently enabled, even when the RTOS kernel is in a critical section, but cannot make any calls to FreeRTOS API functions. If $configASSERT()$ is defined in $FreeRTOSConfig.h$ then *portASSERT_IF_INTERRUPT_PRIORITY_INVALID()* will result in an assertion failure if a FreeRTOS API function is called from an interrupt that has been assigned a priority above the configured maximum system call priority. Only FreeRTOS functions that end in FromISR can be called from interrupts that have been assigned a priority at or (logically) below the maximum system call interrupt priority. FreeRTOS maintains a separate interrupt safe API to ensure interrupt entry is as fast and as simple as possible[51].

The priority of the peripheral should be between the low and high interrupt priority set by chosen RTOS, Otherwise, some of the RTOS APIs may fail to work inside interrupts In case of FreeRTOS,the Lowest Interrupt priority is set by *configLIBRARY_LOWEST_INTERRUPT_PRIORITY* and Maximum interrupt priority by *configLIBRARY_MAX_SYSCALL_INTERRUPT_PRIORITY*, So Interrupt Priority of the peripheral should be between *configLIBRARY_MAX_*

SYSCALL_INTERRUPT_PRIORITY and *configLIBRARY_ LOWEST_ INTERRUPT_ PRIORITY* [52].

A.7.2 FreeRTOS timers

As mentioned before, software timers impacts efficiency of the code. The following information is of high importance when porting the libraries to another scheduler.

Software timer functionality is easy to implement, but difficult to implement efficiently. The FreeRTOS implementation does not execute timer callback functions from an interrupt context, does not consume any processing time unless a timer has actually expired, does not add any processing overhead to the tick interrupt, and does not walk any link list structures while interrupts are disabled.

The timer service task (primarily) makes use of existing FreeRTOS features, allowing timer functionality to be added to an application with minimal impact on the size of the application's executable binary.

Furthermore, timer callback functions execute in the context of the timer service task. It is therefore essential that timer callback functions never attempt to block. For example, a timer callback function must not call $vTaskDelay()$, $vTaskDelayUntil()$, or specify a non zero block time when accessing a queue or a semaphore[53].

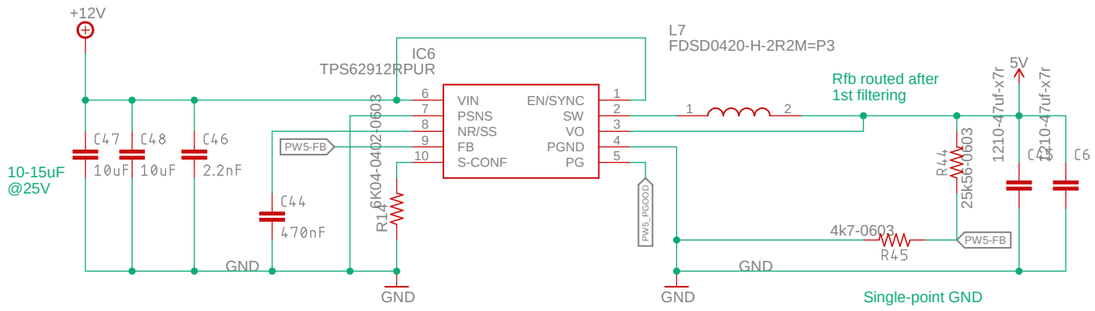
In the current release of the firmware one-shot timers are implemented.

B Appendix B

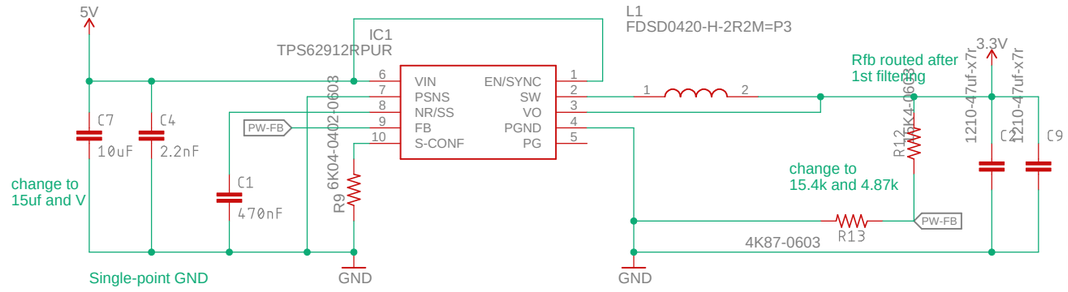
B.1 Schematics

PW and Charger

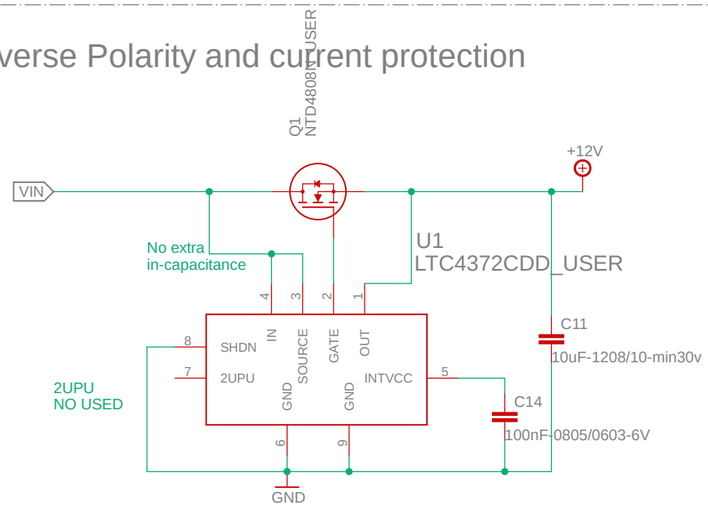
DC-DC 12-5

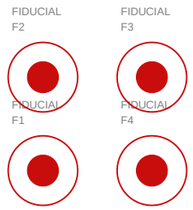


DC-DC 5-3.3



Reverse Polarity and current protection





	Reyes, Carlos
	CommPCB
	9/6/2021 4:09 PM
	Sheet: 1/4

Satellite communication

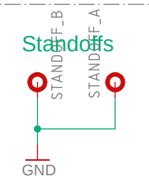
Footprint ref

H1
MOUNT-HOLE3.2

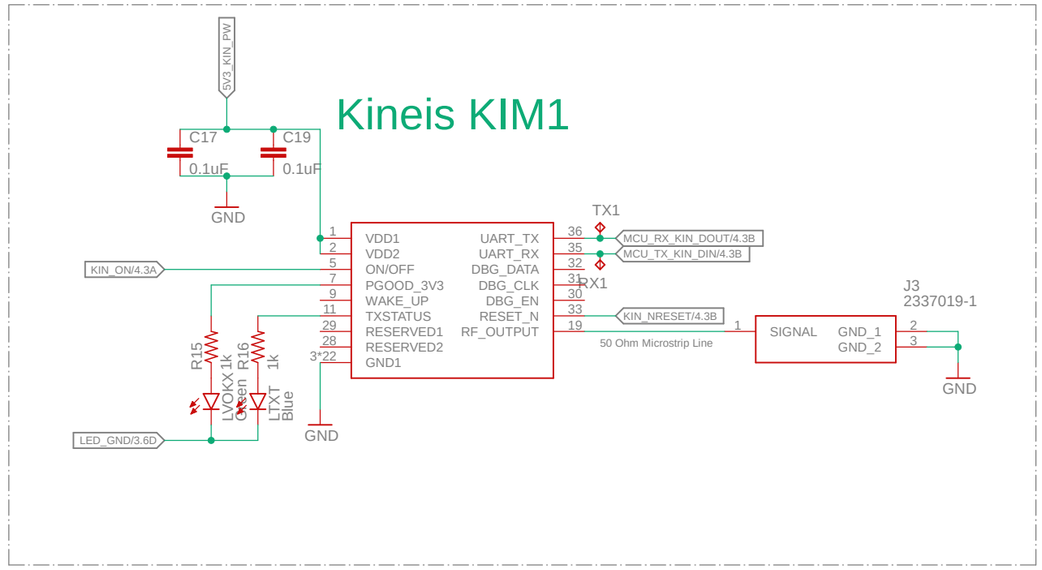
H2
MOUNT-HOLE3.2

H4
MOUNT-HOLE3.2

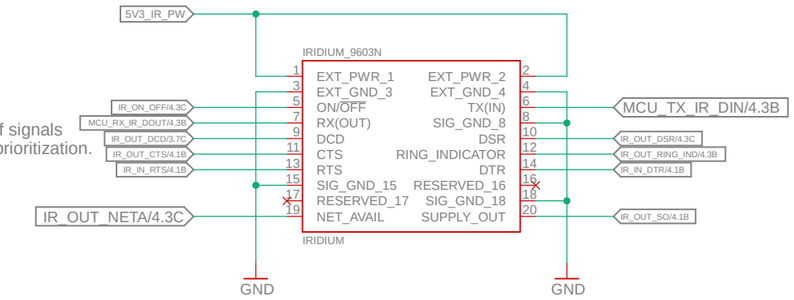
Standoffs



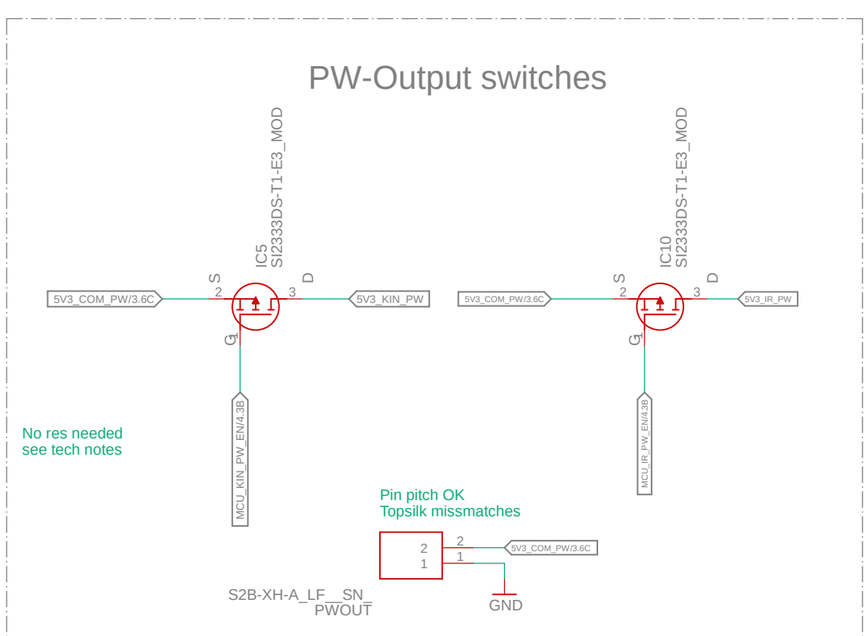
Kineis KIM1



See the list of signals on Excel for prioritization.



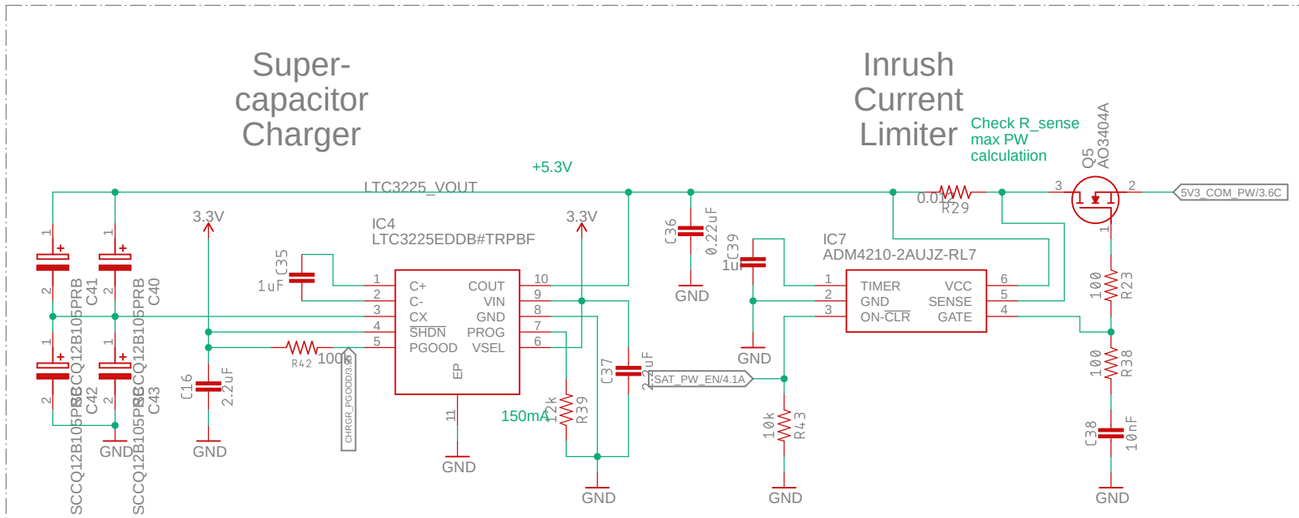
PW-Output switches



Super-capacitor Charger

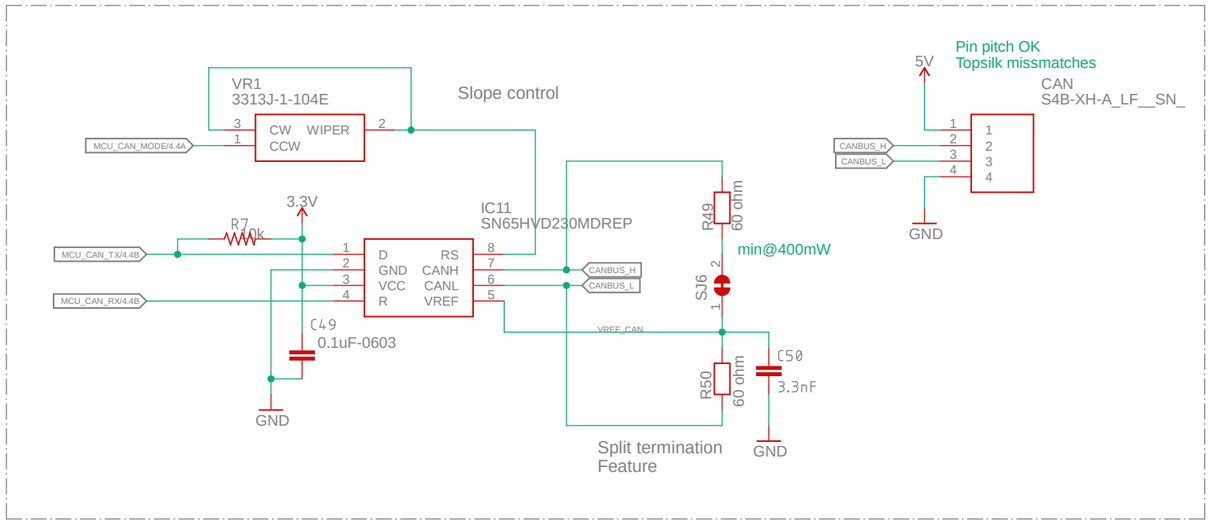
Inrush Current Limiter

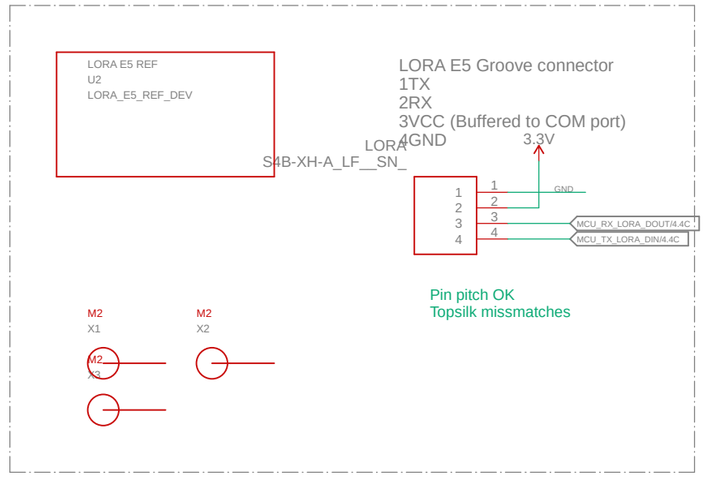
Check R_sense max PW calculatiion

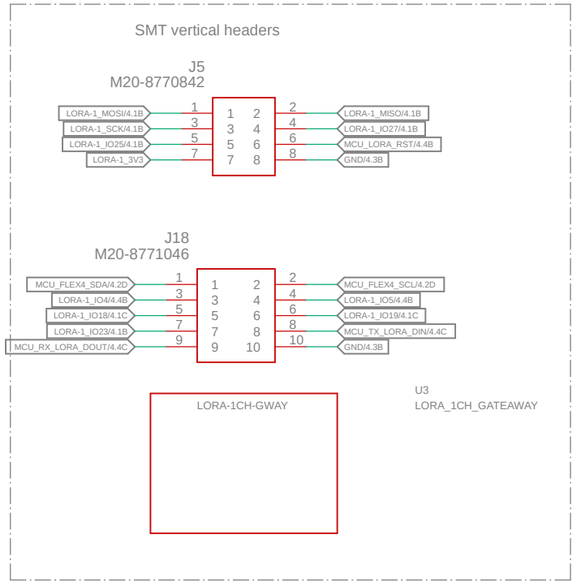


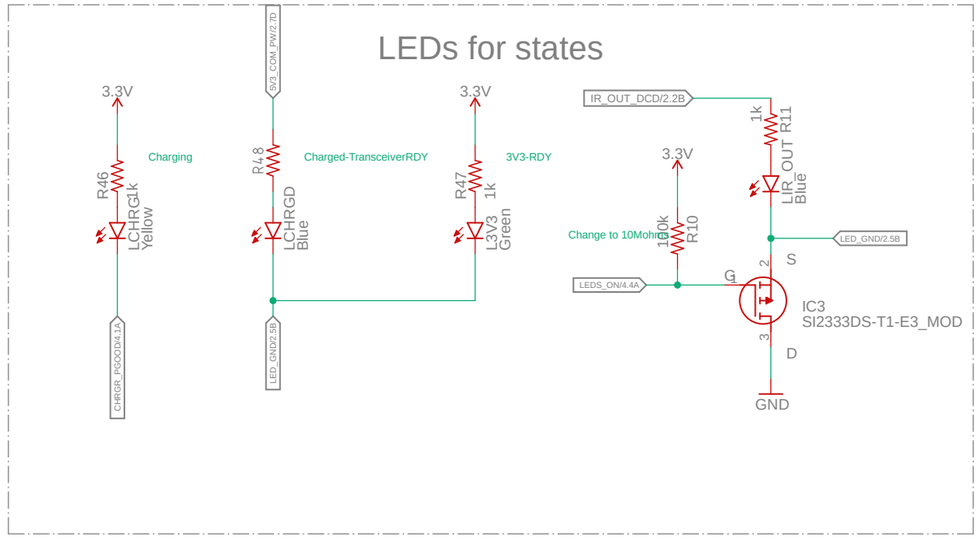
	Reyes, Carlos
	CommPCB
	9/6/2021 4:09 PM
	Sheet: 2/4

CAN and LoRa transceiver + LEDs









	Reyes, Carlos
	CommPCB
	9/6/2021 4:09 PM
	Sheet: 3/4

MCU Logic

List of Not-linked signals

CHRGR_PGOOD/3.5D

CHRGR_SHDN is not needed, always charging

CHRGR_SHDN

SAT_PW_EN/2.3E

IR_OUT_CTS/2.2C

IR_OUT_SO/2.3C

IR_IN_RTS/2.2C

IR_IN_DTR/2.3C

LORA-1_MOSI/3.1C

LORA-1_SCK/3.1C

LORA-1_MISO/3.2C

LORA-1_IO27/3.2C

LORA-1_IO25/3.1C

LORA-1_IO23/3.1D

LORA-1_IO19/3.2D

LORA-1_IO18/3.1D

IC9 ATSAME51G18A-MUT

3.3V

VCORE_TP

R6 1K 0603

KIN_ON/2.5B

SAT_PW_EN/2.3E

MCU_CAN_MODE/3.2B

LEDS_ON/3.7D

CHRGR_PGOOD/3.5D

49	GND	EP	
48	3.3V	PB03	
47	VSW	PB02	
46	GND	PA31	
45	VCORE	PA30	
44	NRST	VDDIO_3	
43	SWDIO	VSW	
42	SWCLK	GND_3	
41	3.3V	VDDCORE	
40	VSW	RESETN	
39	GND	PA27	
38	VCORE	PB23	
37	CHRGR_PGOOD/3.5D	PB22	

1	MCU_TX_KIN_DIN/2.7B	PA00	
2	MCU_RX_KIN_DOUT/2.7B	PA01	
3	MCU_KIN_PW_EN/2.6D	PA02	
4	KIN_NRESET/2.7B	PA03	
5	GND	GNDANA	
6	3.3V	VDDANA	
7	MCU_IR_PW_EN/2.8D	PB08	
8	IR_OUT_RING_IND/2.3C	PB09	
9	MCU_TX_IR_DIN/2.3B	PA04	
10	MCU_RX_IR_DOUT/2.2B	PA05	
11	IR_OUT_NETA/2.2C	PA06	
12	IR_ON_OFF/2.2B	PA07	
36	3.3V	VDDIO_2	
35	GND	GND_2	
34		PA25	MCU_USB_DP
33		PA24	MCU_USB_DM
32		PA23	MCU_CAN_RX/3.1B
31		PA22	MCU_CAN_TX/3.1B
30		PA21	
29		PA20	LORA-1_IO4/3.1D
28		PA19	LORA-1_IO5/3.2D
27		PA18	MCU_LORA_RST/3.2C
26		PA17	MCU_TX_LORA_DIN/3.8B
25		PA16	MCU_RX_LORA_DOUT/3.8B

13	MCU_FLEX2_SCL	PA08	
14	MCU_FLEX2_SDA	PA09	
15	MCU_FLEX2_3	PA10	
16	MCU_FLEX2_2	PA11	
17	3.3V	VDDIO_1	
18	GND	GND_1	
19	IR_OUT_DSR/2.3B	PB10	
20		PB11	
21		PA12	
22	MCU_FLEX4_SCL/3.2D	PA13	
23	MCU_FLEX4_SDA/3.1D	PA14	
24	MCU_FLEX4_3	PA15	

Connectors

Pin pitch OK
Topsilk missmatches

FLEXCOM_IV
S4B-XH-A_LF__SN_

MCU_FLEX4_3
MCU_FLEX4_2
MCU_FLEX4_SDA/3.1D
MCU_FLEX4_SCL/3.2D

FLEXCOM_II
S4B-XH-A_LF__SN_

MCU_FLEX2_3
MCU_FLEX2_2
MCU_FLEX2_SDA
MCU_FLEX2_SCL

SWD

SWD
FTSH-105-01-L-DV

3.3V
GND
SWDIO
SWCLK
NRST

Reset

3.3V
R3 22k 0603
R4 330 0603
NRST
0.1uF
C33
GND

Pull-up RES for I2C

3.3V
MCU_FLEX4_SDA/3.1D
MCU_FLEX4_SCL/3.2D
5K6 R8
5K6 R5
MCU_FLEX2_SDA
MCU_FLEX2_SCL
2K2 R2
2K2 R1

Filtering

L3
LQH3NPZ100MMEL
VSW 1 2 VCORE

3.3V
C15 10uF changed manually
C10 0.1uF
C12 0.1uF
C13 0.1uF
C5 0.1uF
C8 0.1uF
C3 4.7uf changed manually
GND

Reyes, Carlos	
CommPCB	
9/6/2021 4:09 PM	
Sheet: 4/4	

C Appendix C

C project and the AMuSeD libraries are provided separately within the attached CD.

C.1 Python codes

```
#   Author: Carlos Reyes
#   Created on: 2022.01.14
#   Brief:  Second simple Python code for the Drifter's Communication Board test (reception)
#   Requisites:
#   Working time: 1 hr (21 21:21)
#                 1.5 hrs (From v2)
#   Notes:  This test code works with the CAN implementation of canio lib. For maximum throughput a implementation of CAN FD should be implemented.
#           Info about structs to format a message: https://docs.python.org/3/library/struct.html#format-strings
#                 Currently messages are sent using unsigned int, little-endian and its default padding
#
import board
import time
import neopixel
import digitalio
import struct
import canio

print("Hallo Grom: This is receiver v3")


# If the CAN transceiver has a standby pin, bring it out of standby mode
if hasattr(board, 'CAN_STANDBY'):
    standby = digitalio.DigitalInOut(board.CAN_STANDBY)
    standby.switch_to_output(False)

# If the CAN transceiver is powered by a boost converter, turn on its supply
if hasattr(board, 'BOOST_ENABLE'):
    boost_enable = digitalio.DigitalInOut(board.BOOST_ENABLE)
    boost_enable.switch_to_output(True)
```

```
# Use this line if your board has dedicated CAN pins. (Feather M4 CAN and Feather STM32F405)
can = canio.CAN(rx=board.CAN_RX, tx=board.CAN_TX, baudrate=125_000, auto_restart=True, silent=False, loopback=False)
# On ESP32S2 most pins can be used for CAN.  Uncomment the following line to use IO5 and IO6
#can = canio.CAN(rx=board.IO6, tx=board.IO5, baudrate=250_000, auto_restart=True)
#   Specific CANbus identifiers
int_canRcvrId = 0x408
float_timeout = 2.0 #   Timeout in seconds
listener = can.listen(matches=[canio.Match(int_canRcvrId)], timeout=float_timeout)

old_bus_state = None
old_count = -1

#   Enablling red LED
led = digitalio.DigitalInOut(board.LED)
led.direction = digitalio.Direction.OUTPUT
#   Enabling the NEOPIXEL power
neop = neopixel.NeoPixel(board.NEOPIXEL, 1)
b_dummyCtrl = digitalio.DigitalInOut(board.NEOPIXEL_POWER)
b_dummyCtrl.switch_to_output(True)

counter = 0


while True:
    bus_state = can.state
    neop.fill((255, 0, 0))
    led.value = False
    if bus_state != old_bus_state:
        print(f"Bus state changed to {bus_state}")
        old_bus_state = bus_state

    message = listener.receive()
    if message is None:
        print("No messsage received within timeout")
        continue
    # A message was received!
    neop.fill((0, 255, 0))
    led.value = True
    data = message.data
    if len(data) != 8:
        print(f"Unusual message length {len(data)}")
        continue
```

```
    #   Unpacking raw data
    print(f"Unpacked data as second segment is: ")
    canDataPacketX = struct.unpack('<hhhh',data)
    print(" ".join(str(n) for n in canDataPacketX))

    print(f"Unpacked data as third segment is: ")
    canDataPacketX = struct.unpack('<hhf',data)
    print(" ".join(str(n) for n in canDataPacketX))

    print(f"Unpacked data as fourth segment is: ")
    canDataPacketX = struct.unpack('<ff',data)
    print(" ".join(str(n) for n in canDataPacketX))

    print(" ".join(hex(n) for n in data))
    data2 = struct.unpack('<BBBBBBBB',data)
    print(f"Unpacked little-endian is:")

    #   Periodic print of state
    if counter%20==0:
        print(f"Current Bus state {bus_state}")


    #   Default sleep time
    time.sleep(0.5)
    counter = (counter+1)%20
```

```
    #   Author: Carlos Reyes
#   Created on: 2021.12.18
#   Brief:  Simple Python code for the Drifter's Communication Board test
#   Version: 2.0
#   Requisites:
#   Working time:   4.5 hrs (as for 2022.01.31)
#                   6 hrs (From V1)
#   Notes:  This test code works with the CAN implementation of canio lib. For maximum throughput a implementation of CAN FD should be implemented.
#           Info about structs to format a message: https://docs.python.org/3/library/struct.html#format-strings
#               Currently messages are sent using unsigned int, little-endian and its default padding
#
import board
import time
import neopixel
import digitalio
```

```
import struct
import canio

print("Hallo Grom")

#   Initial config
# If the CAN transceiver has a standby pin, bring it out of ↩
    standby mode
if hasattr(board, "CAN_STANDBY"):
    print("CAN was on STANDBY")
    standby = digitalio.DigitalInOut(board.CAN_STANDBY)
    standby.switch_to_output(False)
# If the CAN transceiver is powered by a boost converter, turn on↩
    its supply
if hasattr(board, "BOOST_ENABLE"):
    boost_enable = digitalio.DigitalInOut(board.BOOST_ENABLE)
    boost_enable.switch_to_output(True)    #True if 5V

can = canio.CAN(rx=board.CAN_RX, tx=board.CAN_TX, baudrate=125↩
    _000, auto_restart=True)
monitorBusState = canio.BusState

#   Specific CANbus identifiers
#   Total size 32 bytes for version 1
int_nodeTxMsgId = 0x111   #   Filter that commboard is listening ↩
    to
#   Data size should be kept as specified, otherwise ↩
    inconsistencies may occur.
#--- 1 - 8 bytes
uchar_nodeID = 0x11          #   Not larger than 7F. See ↩
    documentation. Filters: Rx = 0x110 Tx = 0x111
uchar_version = 0x01
uchar_intFlag = 0x01
uchar_nrOfInt = 0x03
uchar_floatFlag = 0x01
uchar_nrOfFloat = 0x03
uchar_command = 0x00
uchar_reserved01 = 0xF0
#   Payload
#--- 9 - 16 bytes
short_int0 = 0x00
short_int1 = 0x00
short_int2 = 0x00
short_reserved00 = 0x0FFF
#--- 17 - 24 bytes
short_reserved01 = 0x0FFF
short_reserved02 = 0x0FFF
float_float0 = 0.0
#--- 25 - 32 bytes
```

```
float_float1 = 0.0
float_float2 = 0.0
#--- 33 - 40 bytes
#   Not implemented
#--- 41 - 48 bytes
#   Not implemented
#--- 49 - 56 bytes
#   Not implemented
#--- 57 - 64 bytes
#   Not implemented


#   Enablling red LED
led = digitalio.DigitalInOut(board.LED)
led.direction = digitalio.Direction.OUTPUT
#   Enabling the NEOPIXEL power
neop = neopixel.NeoPixel(board.NEOPIXEL, 1)
b_dummyCtrl = digitalio.DigitalInOut(board.NEOPIXEL_POWER)
b_dummyCtrl.switch_to_output(True)

#   Some print variables
old_bus_state = None
count = 0
#   Some user variables variables
user_int0 = 0
user_float0 = 0.0

#   Some useful data

print(f"Node's ID (dec): ",uchar_nodeID)
print(f"Node's CAN Packet version: ",uchar_version)
print(f"Node's TX MSG ID: ",int_nodeTxMsgId)

#   This is the main execution loop
while True:
    # print("Hey Grom!!")
    neop.fill((255, 0, 0))
    led.value = True
    # ------------------------------
    #   User modifies the intx or floatx according to limits. Do ↩
       not go beyond their defined size.
    #------------------------------

    #   Example: This only increases the value of one integer and↩
        one float (no overflow).
    user_int0 += 10
    if user_int0 > 100:
        user_int0 = -100
```

```
    user_float0 += 10.5
    if user_float0 > 100.0:
        user_float0 = -100.0

    #   Copying the user data to common can data packets
    short_int0 = user_int0
    short_int1 = user_int0+10
    short_int2 = user_int0+20
    float_float0 = user_float0
    float_float1 = user_float0+.5
    float_float2 = user_float0-5.25

    #---------------------------------
    #   Here finishes space for user's modification
    # --------------------------------


    #   Monitoring the state of the device
    bus_state = can.state
    if bus_state != old_bus_state:
        print(f"Bus state changed to {bus_state}")
        old_bus_state = bus_state

    #   Total size 32 bytes for version 1

    #   Packing up the common CAN data packets
    canDataPacket0 = struct.pack("<BBBBBBBB",
                                         uchar_nodeID,
                                         uchar_version,
                                         uchar_intFlag,
                                         uchar_nrOfInt,
                                         uchar_floatFlag,
                                         uchar_nrOfFloat,
                                         uchar_command,
                                         uchar_reserved01)
    canDataPacket1 = struct.pack("<hhhh",
                                         short_int0,
                                         short_int1,
                                         short_int2,
                                         short_reserved00)
    canDataPacket2 = struct.pack("<hhf",
                                         short_reserved01,
                                         short_reserved02,
                                         float_float0)
    canDataPacket3 = struct.pack("<ff",
                                         float_float1,
                                         float_float2)

    canDataPacket = (canDataPacket0, canDataPacket1, ↩
```

```
        canDataPacket2, canDataPacket3)

    if bus_state == monitorBusState.ERROR_ACTIVE or True:   #   Always send over CAN
        now_ms = (time.monotonic_ns() // 1_000_000) & 0xFFFFFFFF
        print(f"Sending message: count={count} now_ms={now_ms}")

        print("Message to be sent:")
        for i in range(4):
            print("".join(hex(n) for n in canDataPacket[i]))
        for i in range(4):
            message = canio.Message(
                id=int_nodeTxMsgId, data = canDataPacket[i]
            )
            can.send(message)
            #time.sleep(0.2)    #   No delay between the 8-bytes segments


        #   Simple print of the state every 5 messages
        count += 1
        if count % 5 == 0:
            print(f"Current bus state{bus_state}")
    else:
        print(f"No message will be sent as the bus is not active")

    # ------------------------------
    led.value = False
    neop.fill((0, 0, 255))
    time.sleep(6)   #  Each 2 seconds sends a new common can data packet
```

```
#   Author: Carlos Reyes
#   Created: 2022.01.29
#   Brief:  Reads out a SBD file received from a Mobile Originated Message according to v1 of AMUSED's communication Prototype


import struct

#   Open the desired .sbd file
f= open(r"example_file5.sbd",'rb')
content = f.read()
print("Hallo Amused User!")
print("This is the binary content of the SBD file:")
```

```
print(content)
f.close()
canDataPacketX = struct.unpack_from('↩
    BBBBBBBBhhhhhhfffBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBB',↩
    content,0)
print("This is the tuple that contains the decoded 70-bytes-data ↩
    according to V1 of the data packet:")
print(canDataPacketX)
```

References

[1] iMEK TUHH. *Autonomous Multi-Sensor Drifter*. 2022. URL: https://www.tuhh.de/imek/forschung-und-projekte/projekte/autonome-multi-sensor-drifter.html (cit. on p. 1).

[2] Schliefkowitz Lars-Ole. *Hardware Implementierung einer Selbstversorgenden Modularen SensorPlattform mit geringemEnergieverbrauch*. 2021 (cit. on pp. 3, 36).

[3] u-blox AG. *SARA-U2 series: HSPA modules with 2G fallback*. 2021 (cit. on p. 4).

[4] AnSem NV. *ENA303 ARTIC R2 User Datasheet*. 2020 (cit. on p. 4).

[5] SparkFun. *SparkFun LTE CAT M1/NB-IoT Shield - SARA-R4*. 2022. URL: https://github.com/sparkfun/LTE_Cat_M1_Shield (cit. on p. 4).

[6] Jean Rabault et al. *OpenMetBuoy-V2021: an easy-to-build, affordable, customizable, open source instrument for oceanographic measurements of drift and waves in sea ice and the open ocean*. 2022. arXiv: 2201.08384 [physics.ins-det] (cit. on p. 4).

[7] Sparkfun. *SparkFun IridiumSBD I2C Arduino Library*. URL: https://github.com/sparkfun/SparkFun_IridiumSBD_I2C_Arduino_Library (cit. on pp. 5, 21).

[8] Vincent J. Riot, Lance M. Simms, and Darrell Carter. "Lessons Learned Using Iridium to Communicate with a CubeSat in Low Earth Orbit". In: *Journal of Small Satellites* 10.1 (Feb. 2021) (cit. on pp. 5, 20, 33, 41).

[9] Margaret M. McMahon and Robert Richard Rathburn. "Measuring Latency in Iridium Satellite Constellation Data Services". In: 2005 (cit. on pp. 5, 20, 32, 33).

[10] Christian Ebi et al. "Synchronous LoRa Mesh Network to Monitor Processes in Underground Infrastructure". In: *IEEE Access* 7 (2019), pp. 57663–57677. DOI: 10.1109/ACCESS.2019.2913985 (cit. on pp. 5, 45).

[11] Tian Deng, Jiang Zhu, and Zhiqiang Nie. "An Adaptive MAC Protocol for SDCS System Based on LoRa Technology". In: *Proceedings of the 2017 2nd International Conference on Automation, Mechanical Control and Computational Engineering (AMCCE 2017)*. Atlantis Press, 2017/03, pp. 825–830. ISBN: 978-94-6252-308-1. DOI: https://doi.org/10.2991/amcce-17.2017.146. URL: https://doi.org/10.2991/amcce-17.2017.146 (cit. on pp. 5, 45).

[12] Bryan Palmintier. "The EMERALD Protocol Suite: Design and Implementation of a Modular, Distributed Architecture for Small Satellite Command, Telemetry, and Power Systems". MA thesis. STANFORD UNIVERSITY, 2004 (cit. on p. 10).

[13] NXP. *2-channel multipoint Fast-mode Plus differential I2C-bus buffer with hot-swap logic*. 2021 (cit. on p. 11).

[14] Linear Technology. *Low Voltage I2C/SMBus Accelerator*. 2008 (cit. on p. 11).

[15] Esther Shein. *Global chip shortage: Everything you need to know*. 2021. URL: https://www.techrepublic.com/article/global-chip-shortage-cheat-sheet/ (cit. on pp. 11, 30).

[16] Robert Gee. "CAN vs. RS-485: Why CAN Is on the Move". In: *MAXIM Integrated publishments* (2021) (cit. on p. 12).

[17] Telecommunications Industry Association (TIA). *TIA-485 Standard*. 2012. URL: https://global.ihs.com/doc_detail.cfm?&csf=TIA&item_s_key=00032964&item_key_date=870024&input_doc_number=485&input_doc_title=&org_code=TIA#abstract-section (cit. on p. 12).

[18] CAN in Automation. *Physical layer options*. 2022. URL: https://www.can-cia.org/can-knowledge/can/systemdesign-can-physicallayer/ (cit. on p. 12).

[19] Texas Instruments Incorporated. *SN65HVD23x 3.3-V CAN Bus Transceivers*. 2018 (cit. on p. 12).

[20] Texas Instruments Incorporated. *THVD15xx 5-V RS-485 Transceivers With ±18-kV IEC ESD Protection*. 2018 (cit. on p. 12).

[21] XELA Robotics. *Tactile Sensor (XR2144) - Instruction Manual*. 2022 (cit. on p. 13).

[22] Dmitry Murzinov. *A curated list of awesome tools, hardware and resources for CAN bus*. 2022. URL: https://github.com/iDoka/awesome-canbus (cit. on p. 13).

[23] Semtech Corporation. *An In-depth look at LoRaWAN® Class A Devices*. 2019. URL: https://lora-developers.semtech.com/documentation/tech-papers-and-guides/lorawan-class-a-devices/ (cit. on p. 15).

[24] The Things Network Support Forum. *Single Channel Packet Forwarders (SCPF) are obsolete and not supported*. 2019. URL: https://www.thethingsnetwork.org/forum/t/single-channel-packet-forwarders-scpf-are-obsolete-and-not-supported/31117 (cit. on p. 15).

[25] The Things Stack. *Major Changes In The Things Stack*. 2021. URL: https://www.thethingsindustries.com/docs/getting-started/migrating/major-changes/ (cit. on p. 15).

[26] Swarm Technologies Inc. *Developer Tools Your guide to getting started with Swarm*. 2022. URL: https://swarm.space/developertools/ (cit. on p. 19).

[27] Iridium Communications Inc. *Iridium 9603 SBD Transceiver Product Developers Guide Revision 2.1 DRAFT 3*. 2014 (cit. on pp. 19, 37, 49).

[28] Iridium Satellite LLC. *Iridium Short Burst Data Service Developers Guide MAN0025 V3.2*. 2017 (cit. on pp. 20, 33, 42, 49, 68).

[29] Hussien Saleh. "Software Implementation Of Energy-Efficient Self-Sufficient Modular Multi-Sensor Platform For Autonomous Ocean Drifter". MA thesis. Hamburg University of Technology, 2022 (cit. on pp. 21, 23, 38).

[30] Amazon Web Services. *The FreeRTOS™ Kernel Market leading, de facto standard, and cross platform RTOS kernel*. 2022. URL: https://freertos.org/RTOS.html (cit. on p. 22).

[31] EmbeTronicX. *RTOS Basics Concepts*. 2022. URL: https://embetronicx.com/tutorials/rtos/freertos/rtos-basic-tutorial-for-beginners/#RTOS_Basics_Tutorial (cit. on p. 22).

[32] Florian Kromer. *FreeRTOS in a nutshell*. 2021. URL: https://medium.com/geekculture/freertos-in-a-nutshell-dd44ab147fa5 (cit. on p. 22).

[33] Nicholas H.Tollervey. *About Mu*. 2022. URL: https://codewith.mu/en/about (cit. on pp. 28, 39).

[34] Adafruit CircuitPython. *canio - CAN bus access*. 2022. URL: https://circuitpython.readthedocs.io/en/latest/shared-bindings/canio/index.html (cit. on p. 28).

[35] Semtech Corporation (cit. on p. 30).

[36] HOPE Microelectronics CO. *RFM95W LoRa Module*. 2018. URL: https://www.hoperf.com/modules/lora/RFM95.html (cit. on pp. 30, 46).

[37] Texas Instruments Incorporated. *TPS6291x 3-V to 17-V, 2-A/3-A Low Noise and Low Ripple Buck Converter with Integrated Ferrite Bead Filter Compensation*. 2021 (cit. on pp. 35, 54).

[38] Iridium Communications Inc. *ISU AT Command Reference MAN0009 Version 5*. 2014 (cit. on pp. 37, 66).

[39] Paul Sokolovsky Damien P. George and contributors. *MicroPython language and implementation*. 2022. URL: http://docs.micropython.org/en/latest/reference/index.html (cit. on p. 39).

[40] Adafruit CircuitPython. *Supported Ports*. 2022. URL: https://circuitpython.readthedocs.io/en/7.1.x/ports/atmel-samd/README.html (cit. on p. 39).

[41] Cian B. *UF2 Bootloader: Creating Custom Boards*. 2019. URL: https://www.hackster.io/wallarug/uf2-bootloader-creating-custom-boards-c9620c#toc-what-is-uf2-1 (cit. on p. 39).

[42] Amazon Web Services. *Low Power Support*. 2021. URL: https://www.freertos.org/low-power-tickless-rtos.html (cit. on p. 41).

[43] KVASER. *The CAN Bus Protocol Tutorial*. 2022. URL: https://www.kvaser.com/can-protocol-tutorial/#errorHandling (cit. on pp. 43, 44, 48).

[44] The Things Network. *Applications and Integrations*. 2021. URL: https://www.thethingsnetwork.org/docs/applications-and-integrations/ (cit. on p. 45).

[45] The Things Network. *Semtech UDP Packet Forwarder*. 2021. URL: https://www.thethingsnetwork.org/docs/gateways/packet-forwarder/semtech-udp/ (cit. on p. 45).

[46] ANALOG DEVICES INC. *Low Quiescent Current Ideal Diode Controller*. 2021 (cit. on p. 55).

[47] M. Westenberg. *Single Channel LoRaWAN Gateway*. 2021. URL: https://github.com/things4u/ESP-1ch-Gateway (cit. on p. 60).

[48] MCCI Corporation. *Arduino-LMIC library ("MCCI LoRaWAN LMIC Library")*. 2021. URL: https://github.com/mcci-catena/arduino-lmic (cit. on p. 60).

[49] Keith Shaw. *The OSI model explained and how to easily remember its 7 layers*. 2021. URL: https://www.networkworld.com/article/3239677/the-osi-model-explained-and-how-to-easily-remember-its-7-layers.html (cit. on p. 69).

[50] Carlos Reyes. *Development of an embedded communication hub for sensor data acquisition in a robotic system*. Tech. rep. Research Group smartPORT Hamburg University of Technology, 2022 (cit. on p. 74).

[51] Amazon Web Services. *Running the RTOS on a ARM Cortex-M Core*. 2021. URL: http://www.freertos.org/RTOS-Cortex-M3-M4.html (cit. on p. 74).

[52] Amazon Web Services. *FreeRTOS Kernel:include libraries*. 2022. URL: https://github.com/FreeRTOS/FreeRTOS-Kernel/tree/main/include (cit. on p. 75).

[53] Amazon Web Services. *Software Timers*. 2021. URL: https://www.freertos.org/RTOS-software-timer.html (cit. on p. 75).