Marc Falcón Barau

Bachelor's thesis

Iridium communication for submersible communication buoy

Theoretical and practical implementation

September 2023

NTNU
Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Computer Science

NTNU
Norwegian University of
Science and Technology

Bachelor’s thesis

2023

NTNU
Norwegian University of
Science and Technology

Marc Falcón Barau

Iridium communication for submersible communication buoy

Theoretical and practical implementation

Bachelor's thesis
September 2023

NTNU
Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Computer Science

ABSTRACT

ENGLISH

The main objective of this Bachelor's Thesis was to determine the most suitable service among Iridium Certus 100, Short Burst Data (SBD), and Router-Based Unrestricted Digital Internetworking Connectivity Solutions (RUDICS) for the Ocean Access submersible buoy project. Ocean Access sought to identify the optimal service based on different use cases, with each use case contingent upon the selected service and further defined through the course of the thesis work. The selection of the service would be based on the type of data being collected, the geographical location, and the required frequency of transmission.

During the project, successful implementation of the SBD service was achieved in the existing software infrastructure. However, due to the availability of modules and pre-existing manufacturer implementations, the consideration of Iridium Certus 100 and RUDICS was confined to theoretical evaluations.

Nevertheless, the thesis played a helpful role in assisting Ocean Access in identifying the specific use cases where each of the three services could offer valuable contributions to their research. Additionally, the thesis provided a pathway and guidelines for the future implementation of the remaining two services, Iridium Certus 100 and RUDICS, in their endeavors.

NORWEGIAN

Hovedmålet med denne bacheloroppgaven var å finne ut hvilken tjeneste som er best egnet blant Iridium Certus 100, Short Burst Data (SBD) og RUDICS (Router-Based Unrestricted Digital Internetworking Connectivity Solutions) for Ocean Access' undervannsbøyeprosjekt. Ocean Access søkte å identifisere den optimale tjenesten basert på ulike bruksområder, der hvert bruksområde var avhengig av den valgte tjenesten og ble ytterligere definert i løpet av arbeidet med avhandlingen. Valget av tjeneste vil være basert på typen data som samles inn, den geografiske plasseringen og den nødvendige overføringsfrekvensen.

I løpet av prosjektet ble SBD-tjenesten implementert i den eksisterende programvareinfrastrukturen. På grunn av tilgjengeligheten av moduler og eksisterende produsentimplementeringer ble imidlertid Iridium Certus 100 og RUDICS begrenset til teoretiske evalueringer.

Avhandlingen var likevel nyttig for Ocean Access når det gjaldt å identifisere spesifikke bruksområder der hver av de tre tjenestene kunne gi verdifulle bidrag til forskningen. I tillegg ga avhandlingen retningslinjer for den fremtidige implementeringen av de to gjenværende tjenestene, Iridium Certus 100 og RUDICS, i deres videre arbeid.

PREFACE

I am delighted to present this thesis, which explores the potential of Iridium communication services in the context of submersible communication buoys. This work represents the culmination of my academic journey and research efforts during my time in Norway as part of the Erasmus program.

The main focus of this thesis revolves around the in-depth examination of Iridium communication services and their applicability to the submersible buoy project developed by Ocean Access. The allure of this project and the prospect of collaborating with individuals from a different cultural and academic backgrounds were the initial driving forces behind my involvement.

To investigate the feasibility of incorporating Iridium communication services into the Ocean Access submersible buoy project, I embarked on a journey of research and data collection. This journey involved engaging with key service providers and seeking guidance from my mentor, Simen Helgesen.

The process encompassed comprehending the intricacies of satellite communication, assessing the suitability of various services, deciphering existing code, and studying manuals for different modems associated with each service. While challenges were encountered along the way, the endeavor provided valuable insights into the world of satellite communication.

In the realm of Computer Engineering, this thesis holds the potential to contribute significantly to the understanding and data collection efforts in marine environments. It can aid scientists in monitoring remote sea locations, which might otherwise be challenging to access.

This thesis primarily targets scientists and researchers seeking solutions for monitoring projects or experiments in aquatic or extreme environments. While initially conceived for submersible buoys, the insights presented herein are relevant to a wide range of monitoring projects.

My experience working with international collaborators has expanded my horizons and instilled in me a desire to explore opportunities beyond my home country.

As the final chapter of my undergraduate journey, I hope this thesis serves as a valuable resource for readers, enabling them to make informed decisions about

the suitability of Iridium services for their own projects. I look forward to the impact and potential applications that may arise from this work.

I extend my heartfelt gratitude to Simen Helgesen for his unwavering support, guidance, and assistance throughout the research and implementation phases of this project. I am also indebted to Fredrik Lilleøkdal and the team at Ocean Access for their collaboration and for embracing this project. Furthermore, I would like to acknowledge Arne Midjo for facilitating my connection with Ocean Access, which made this thesis possible.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

ABBREVIATIONS

List of all abbreviations in alphabetic order:

- **AVL** Automatic Vehicle Location
- **BSS** Broadcast satellite service
- **FSS** Fixed satellite service
- **GNNS** Global Navigation Satellite System
- **IoT** Internet of Things
- **ITU** The International Telecommunication Union
- **MSS** Mobile satellite service
- **NASA** The National Aeronautics and Space Administration
- **NTNU** Norwegian University of Science and Technology
- **RUDICS** Router-based Unrestricted Digital Interworking Connectivity Solution
- **SBD** Short Burst Data
- **UART** Universal Asynchronous Receiver-Transmitter
- **UPC** Polytechnic University of Catalonia

CHAPTER

ONE

INTRODUCTION

Ocean Access is a Norwegian startup with a mission to capture previously unrecorded ocean data. The company recognizes the growing demand for remote ocean monitoring across various industries and aims to address this need. Their primary goal is to scale up ocean monitoring efforts, gather extensive oceanic data, and facilitate digitalization in the ocean sector. To achieve this, Ocean Access emphasizes the importance of developing intelligent tools that are affordable and dependable, enabling widespread adoption and utilization.

Ocean Access is currently in the process of developing a fully submersible ocean data buoy capable of vertical movement within the water column. This innovative design allows the buoy to be submerged and positioned subsea, shielding it from potential damage caused by vessels, waves, and wind on the ocean surface. By providing this subsea placement capability, the solution offers substantial cost reductions and enhanced reliability for remote ocean access. The buoy's ability to navigate the water column will enable more extensive data collection and monitoring, contributing to a greater understanding of our oceans.

1.1 Motivation

As a passionate and driven student, I am eager to embark on this project with Ocean Access as I see it as an opportunity to work on a groundbreaking project, developing the communication of the submersible buoy.

Ocean Access presents the perfect platform for me to channel my enthusiasm and academic prowess into tangible advancements in ocean monitoring and data collection. The prospect of working on this project capturing previously unattainable ocean data and transmitting it, is both thrilling and challenging. By being submerged and shielded from surface disturbances, this innovative buoy offers a novel solution to the limitations of traditional monitoring methods.

Moreover, Ocean Access's commitment to affordability and reliability aligns perfectly with my vision of making scientific advancements accessible and impactful. By ensuring that smart tools are not only state-of-the-art but also

cost-effective, we can unlock vast opportunities for digitalization and data-driven decision-making in various industries dependent on oceanic resources.

Undertaking a thesis with Ocean Access would provide me with valuable hands-on experience, enabling me to grow professionally. Also collaborating with a team of experts in their fields would expose me to diverse perspectives and foster interdisciplinary thinking, propelling my learning to new heights.

I am genuinely excited about the possibility of doing the Thesis with Ocean Access, working tirelessly to uncover untapped ocean data and contribute to a sustainable future. Also, I eagerly anticipate the opportunity to contribute my skills and expertise to this project.

1.2 Project description

To expand upon the available geographical areas where deployment of the buoy is possible, there is a need to explore the option of using satellite communication to send gathered data, instead of using cellular infrastructure. Creating a system that does not need to rely on nearby cellular networks will increase the operational range of the submersible

communication buoy. The satellite solution will also be used as a backup in areas with LTE-coverage, in case of network outages outside the control of Ocean Access. Previously there has been a Bachelor thesis written by a group of students at the department of electronic systems at NTNU where the Swarm satellite service was investigated. Due to their work, Iridium is being investigated as an alternative to serve as the provider of satellite communication.

There are three different Iridium services being considered: Certus 100, Rudics, and SBD. Ocean Access would like to find out which of these services should be used based on different use cases. Use case will depend on which service is used and can be further defined through the thesis work, below is a table showing typical use cases and corresponding services that are likely most suitable. The type of data being collected, the location and frequency of transmission should determine what service

is best suited.

The proposed work packages for the Bachelor Thesis at Ocean Access provide a comprehensive roadmap for advancing the capabilities of the existing ocean data system. These packages encompass essential tasks that are vital for the successful implementation and integration of satellite communication technology, specifically focusing on the Iridium network.

1. The first work package emphasizes the importance of conducting a thorough literature search on satellite communication and comprehending the functioning of the Iridium network. Building upon previous theses conducted at Ocean Access, this step will provide a solid foundation of knowledge and insights to inform subsequent work.

2. The second work package centers around the design and development of the code necessary for incorporating Certus 100, Rudics, or SBD into the existing system. This step will involve coding to ensure seamless integration and efficient data transfer capabilities.

3. The subsequent work package, the implementation of the solution into the existing system, highlights the need for the new features to seamlessly coexist with the current codebase for sending data. If adjustments to the codebase are required to accommodate larger data transfers, they should be documented thoroughly, providing a clear rationale for each alteration. Additionally, a driver for the modem should be implemented, ideally utilizing a pre-written driver from Iridium or an Iridium partner.

4. The final work package focuses on comprehensive testing to ascertain the limitations and performance metrics of the implemented solution. Parameters such as power consumption, maximum data transfer capacity within a given time window, and the time required for the system to transition from shutdown to successful data transmission should be meticulously measured and recorded. Furthermore, any other factors that emerge as relevant to the system and its specific use case should also be carefully identified and documented.

By successfully completing these work packages, the bachelor theses will significantly contribute to Ocean Access's mission of capturing untapped ocean data and advancing remote ocean monitoring. The outcome of this research and development effort will enhance the system's overall performance, efficiency, and reliability, ultimately driving progress towards a more sustainable future for ocean exploration and understanding.

1.3 Stakeholders

In this section I am going to list all the main actors in this project.

1.3.1 Ocean Access

Ocean Access is the startup that engaged Marc Falcón Barau into working on the project of the submersible buoy. This startup is located in Trondheim and came up from an NTNU startup project and now they are fully autonomous and have offices in the main point of innovation and ocean research in the city.

I am working mainly with two persons in the team of Ocean Access:

1. Simen Helgesen, who is the technical manager and a Marine Cybernetic Engineer.

2. Fredrik Lilleøkdal, who is the CTO (Chief Technology Officer) of the startup, and is one of the links between NTNU and Ocean Access.

1.3.2 NTNU

NTNU (Norwegian University of Science and Technology) is the university where I am going to develop my Thesis. Located in Trondheim, Norway.

For this Thesis, I needed a supervisor to be able to make the Thesis possible. This supervisor is Arne Midjo, who is acting as a link between NTNU and Ocean Access along with Fredrik.

1.3.3 UPC

UPC (Universitat Politècnica de Catalunya) is my home university. Located in Barcelona. This actor is going to play a secondary role in the development of the Thesis, as it has no jurisdiction accorded by the document of Mobility Thesis Authorization they signed along with Marc Falcón Barau. The only requirement is that the final report has to be delivered to them for a final revision.

1.4 Problem description

When implementing a digital communication system with the Iridium satellite constellation using services like Rudics, Certs 100, and SBD, there are several potential challenges that may arise:

- **Network Latency**: One of the primary issues when utilizing satellite communication is the inherent latency caused by the long distances the signals need to travel. This can result in delays in data transmission, affecting real-time applications and requiring careful consideration during system design.
- **Bandwidth Limitations**: Satellite communication systems often have limitations on available bandwidth. Depending on the selected Iridium service, there may be restrictions on data transfer rates, which can impact the amount and speed of data that can be transmitted.
- **Cost Considerations**: While Iridium services provide global coverage, they typically come with associated costs. Implementing and maintaining a reliable connection with the Iridium satellite constellation can involve expenses, and the specific service chosen may have pricing implications based on usage requirements.
- **Signal Interference**: Environmental factors such as atmospheric conditions, obstructions, or interference from other signals can affect the quality and reliability of satellite communications. It is important to assess and mitigate potential sources of interference to ensure a robust and consistent connection.
- **Integration Complexity**: Integrating the chosen Iridium service (Rudics, Certs 100, or SBD) into the existing system may present technical complexities. Compatibility with the hardware, software, and communication protocols should be carefully addressed to ensure seamless integration and reliable operation.

- **Power Consumption**: Satellite communication systems typically require sufficient power to establish and maintain connections. Managing power consumption becomes crucial, especially for remote or autonomous systems where energy resources may be limited.
- **System Resilience**: It is important to consider backup and redundancy mechanisms to ensure system resilience in case of network outages or service disruptions. Redundant communication paths, alternative service providers, or local data storage can be explored to mitigate potential connectivity issues.

Addressing these challenges requires thorough planning, system design, and testing to optimize the performance and reliability of the digital communication implementation with the Iridium satellite constellation.

CHAPTER

TWO

THEORY

The Theory section of this Bachelor's Thesis establishes the fundamental framework for the entire study. It explores key concepts, models, and existing research relevant to the project objectives and studies.

2.1 Satellite Networks

Satellite networks have become a crucial technology in global communication, providing reliable communication across vast distances and transcending geographical barriers where traditional terrestrial networks encounter limitations. The technology has been around for several decades and has evolved significantly over the years, from the first satellite launched in 1957 to the advanced satellite constellations of today. Satellite networks consist of a network of satellites that orbit the earth and communicate with ground stations to provide various services such as telecommunication, broadcasting, navigation, and remote sensing. The satellites are positioned in different orbits, including geostationary orbit (GEO), medium earth orbit (MEO), and low earth orbit (LEO), depending on the intended application.[1]

The satellite networking industry involves various actors, including satellite manufacturers, launch service providers, ground station operators, and service providers. The industry has grown significantly over the years, driven by the increasing demand for global connectivity and the emergence of new applications and services. The history of satellite networking dates back to the 1950s when the Soviet Union launched the first satellite, Sputnik 1, into orbit. The launch of Sputnik 1 marked the beginning of the space race between the Soviet Union and the United States, leading to the launch of several other satellites for various applications.[2]

Over the years, satellite networking has become an essential technology for various applications, including telecommunication, broadcasting, navigation, and remote sensing. The technology has played a crucial role in providing connectivity to remote and under-served areas, enabling disaster response and recovery, and supporting scientific research and space exploration. The satellite networking market has grown significantly over the years, driven by the increasing demand for global connectivity and the emergence of new applications and services. According to a

report by MarketsandMarkets, the global satellite communication market is expected to grow from $2.8 billion in 2020 to $7.5 billion by 2025, at a CAGR of 21.0% during the forecast period.[3]

Because this project is intended to be part of the data monitoring space inside the satellite networking; Data monitoring is one of the key applications of satellite networking, enabling real-time monitoring and analysis of various parameters such as weather, climate, and natural resources. The data collected from satellites is used for various applications, including precision agriculture, environmental monitoring, and disaster management.

In conclusion, satellite networks have become a crucial technology in global communication, providing reliable connectivity across vast distances and transcending geographical barriers. The satellite networking industry involves various actors, and the technology has evolved significantly over the years, enabling various applications and services. The satellite networking market is expected to grow significantly in the coming years, driven by the increasing demand for global connectivity and the emergence of new applications and services.

2.2 Orbits

Satellites have become indispensable in modern networking, facilitating worldwide communication, internet access, and data exchange. To ensure smooth and effective communication, satellites are strategically positioned in various orbits, considering their specific purposes and operational needs. There are four different orbits + 1 different, where the satellites can be placed and each of them have different characteristics to ensure different results or for different objectives;

1. **Geostationary orbit (GEO)**: Satellites in GEO orbit at an altitude of 35,900km above the equator, and they appear stationary at the same point in the sky. This makes them ideal for communication purposes, as ground station antennas can be aimed permanently at that spot and do not have to move to track the satellite. However, the longer signal delay caused by their great distance from Earth is a downside for real-time communication.[4][5]

2. **Low Earth orbit (LEO)**: Satellites in LEO orbit at an altitude of up to 2,000 km above Earth's surface. They move quickly around the planet, completing an orbit in about 90 minutes. This makes them ideal for Earth observation and remote sensing, as they can capture high-resolution images of the planet's surface. However, they require a large number of satellites to provide global coverage, and their short orbital lifetime means they need to be replaced frequently.[4][6][7][8]

3. **Medium Earth orbit (MEO)**: Satellites in MEO orbit at an altitude of up to 20,000 km above Earth's surface. They are used primarily for navigation purposes, such as the GPS system. They provide better coverage than LEO satellites, but their higher altitude means they have a longer signal delay.[4][8]

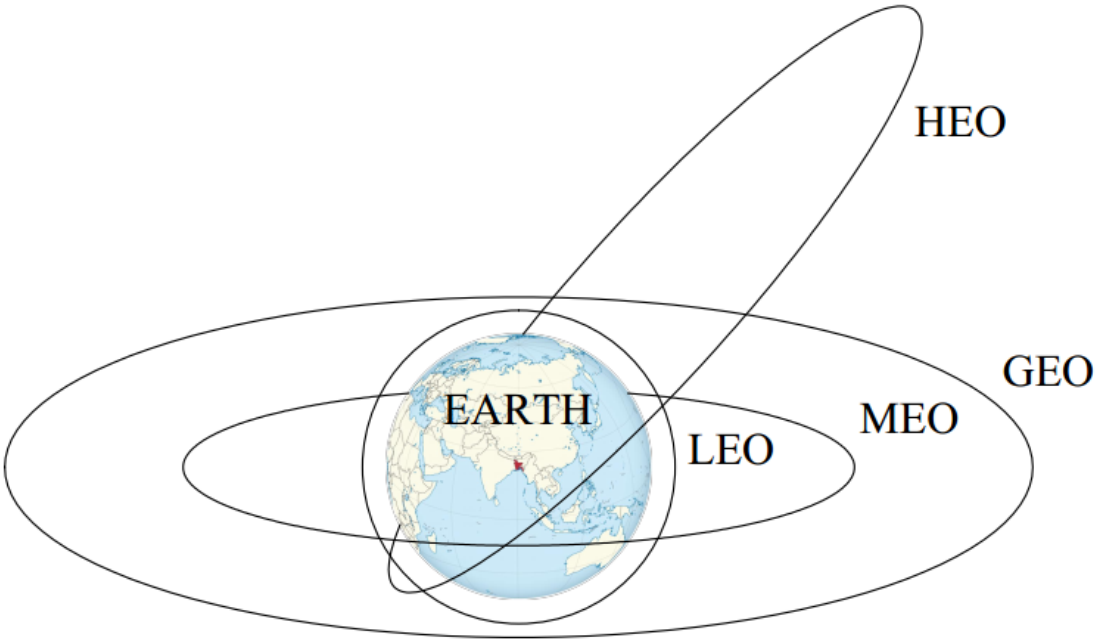

Figure 2.2.1: The arrangement of orbits surrounding the Earth

4. **Highly elliptical orbit (HEO)**: Satellites in HEO have a highly elliptical orbit, which means they spend most of their time at a high altitude and then dip down to a lower altitude for a short period of time. This makes them ideal for communication purposes, as they can cover a large area of the planet's surface. However, their complex orbit requires more fuel to maintain, and they have a longer signal delay than GEO satellites.[7]

5. **Polar orbit**: Satellites in polar orbit pass over the Earth's poles, providing global coverage for Earth observation and remote sensing. They are typically placed in a sun-synchronous orbit (SSO), which means they pass over the same spot on Earth at the same time each day. This allows them to capture images of the planet's surface under consistent lighting conditions, making it easier to compare images taken at different times.[4][6]

In summary, different types of satellites orbits have different characteristics that make them suitable for different purposes. While GEO satellites are ideal for communication purposes, LEO satellites are better suited for Earth observation and remote sensing. MEO satellites are primarly used for navigation, while polar orbit satellites provide global coverage for Earth observation. Highly elliptical orbit satellites are ideal for communication purposes but require more fuel to maintain.

2.3 Satellite Communication System

The satellite communication can be divided into three sections:

1. Uplink Section (Ground Station)
2. Transponder (Airborne Satellite)
3. Downlink Section (Ground Station)

In the uplink section, the user's signal, known as the Baseband Signal, goes through a series of components. The Intermediate Frequency (IF) Modulator converts the baseband frequency to an intermediate frequency using modulation techniques like ASK, FSK, or PSK. The resulting IF signal passes through a Band Pass Filter (BPF) to remove unnecessary frequencies. Next, the IF signal is sent to an Up Converter, where it is converted from MHz to GHz range using a Mixer and Uplink Frequency Microwave Generator. The Radio Frequency (RF) signal is further refined using another BPF, and then its strength is amplified by a High Power Amplifier (HPA) before being transmitted through the Transmitting (Tx) Antenna for long-distance travel.[9][10]

The Transponder, combining a Transmitter and a Responder, receives the uplink frequency through its Receiving (Rx) Antenna. The received RF signal undergoes noise filtering with a BPF and is then amplified by a Low Noise Amplifier (LNA) while keeping the noise level low. To optimize certain advantages, the downlink frequency is usually kept 2 GHz lower than the uplink frequency. A Frequency Translator, involving a Mixer and Microwave Shift Oscillator, performs the frequency conversion. The resulting downlink frequency is further amplified by a Low Power Amplifier (LPA) and transmitted back to Earth through the Transponder's Transmitting (Tx) Antenna.[11][10]

Finally, in the Downlink Section, the received frequency from the Transponder is processed. The Receiving (Rx) Antenna captures the downlink frequency, which is then filtered by a BPF to eliminate unwanted frequencies. The RF signal is amplified by a Low Noise Amplifier (LNA). It is then fed to a Down Converter, where the GHz range is decreased to MHz range using a Mixer and Downlink Frequency Microwave Generator. The Intermediate Frequency (IF) Demodulator converts the IF signal back to the original baseband frequency, representing the user's original signal transmitted via the transponder in the uplink section.[9][12]

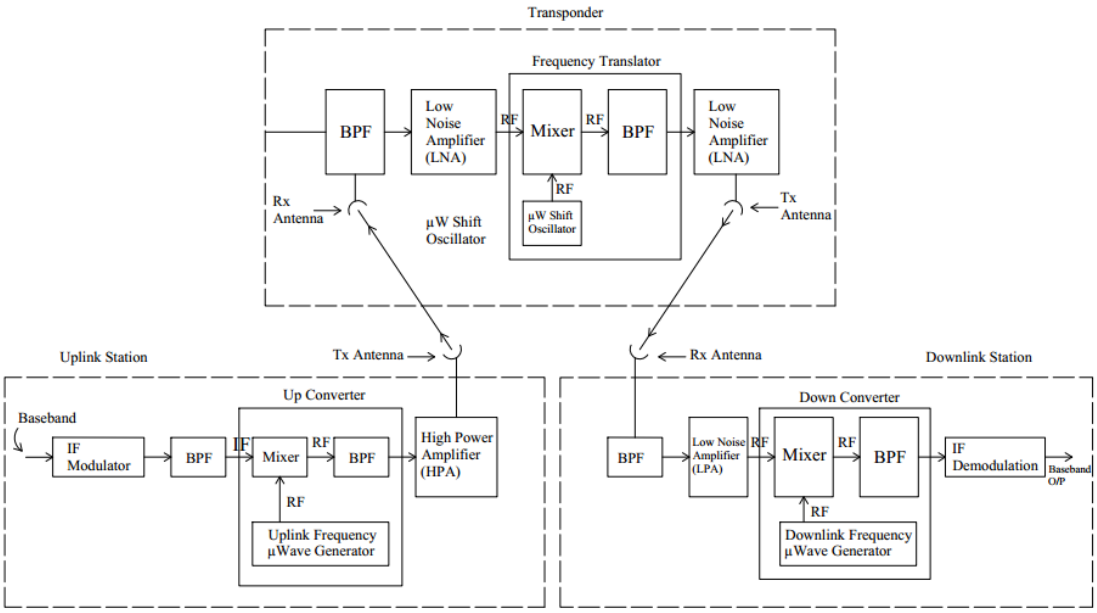

Figure 2.3.1: The complete block diagram of the satellite communication system

2.4 Frequency bands

Satellite communication systems use various frequency bands as commented before, from the electromagnetic spectrum, depending on the market and climatic conditions[13]. The ITU is the global coordinator in satellite frequency allocation, and they have allocated specific bands for satellite communication, including L-band, S-band, C-band, X-band, Ku-band, K-band, and Ka-band[14][13];

- **L-band (1-2GHz)**: Used for MSS, including satellite mobile phones such as Iridium, and Inmarsat providing communications at sea, land, and air[15][14][13].
- **S-band (2-4GHz)**: Used for MSS, NASA, and other applications[14][13].
- **C-band (4-8GHz)**: Primarily used for satellite communication, full-time satellite TV networks, or raw satellite feeds. Also used for FSS and BSS downlinks[15][14][13].
- **X-band (8-12.5GHz)**: Used for meteorological satellite and FSS military[13].
- **Ku-band (12.5-18GHz)**: Used for FSS and BSS[13].
- **K-band (18-26.5GHz)**: Used for FSS and BSS[13].
- **Ka-band (26.5-40GHz)**: Used for FSS and BSS[14][13].

Satellite communication systems are conducted over a wide range of frequency bands, and the typical bands considered for small satellites are UHF, S, X, and Ka[16]. The higher frequency bands typically give access to wider bandwidths, but are also more susceptible to signal degradation due to 'rain fade (the absorption of radio signals by atmospheric rain, snow, or ice)'[15][14]. Because of satellite's increased use, number, and size, congestion has become a serious issue in the lower

frequency bands, and new technologies are being investigated so that higher bands can be used[15][14].

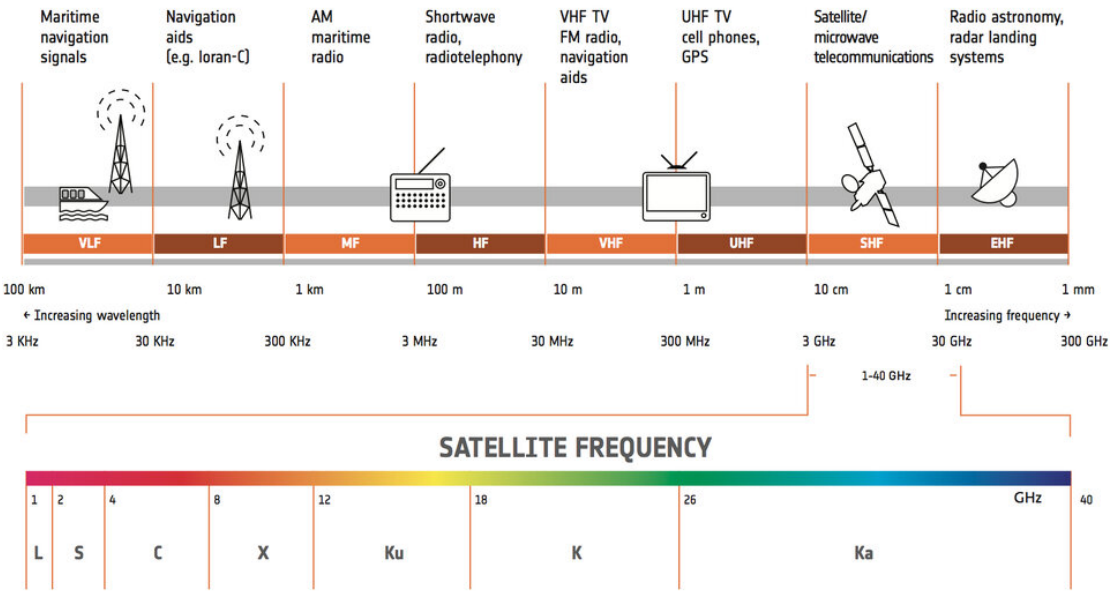

Figure 2.4.1: ESA - Satellite frequency bands, all displayed with the corresponding values

The most commonly used frequency bands for satellite communication are Ku-band and Ka-band[17][18]. These bands are preferred because they offer wider bandwidths, which means higher data rates and higher total capacity[17][18]. The Ka-band provides a wider spectrum availability and high data rate or a bit rate supported communication networks, making it more desirable than low-frequency bands for satellite applications[18]. However, as said earlier high-frequency bands are more susceptible to signal degradation due to atmospheric effects and weather conditions, which can cause signal loss or attenuation[17][18]. Despite this, the technical improvements achieved with higher frequencies make the frequencies more desirable than low-frequency bands for satellite applications[17][18]. But for our project, these great frequency bands are not suitable, because of the harsh weather that rules the Arctic Ocean where our buoy will be deployed.

2.5 Iridium Satellite Network

The Iridium satellite network is a global satellite communications company that provides access to voice and data services anywhere on Earth[19]. The network was founded in 1991 and named after the element Iridium, as it originally consisted of 77 active satellites and additional spares in orbit, matching the atomic number of Iridium (77).
The Iridium network offers complete coverage of the Earth, including remote and polar regions, using a constellation of LEO satellites. This global coverage allows communication services in places where traditional terrestrial networks are not available. The constellation comprises interconnected satellites in LEO. The satellites of Iridium complete an orbit every 100min[20]. To maintain global coverage and seamless connectivity, the Iridium satellites use "inter-satellites links" or "crosslinks". These crosslinks enable data and calls to be relayed between satellites, allowing communication from one satellite to another before reaching the ground station.
The Iridium network offers various communication services;

- **Voice and data communication**: The Iridium satellite network provides worldwide voice and data communication from handheld satellite phones, satellite messenger communication devices, and integrated transceivers[21].The network routes phone calls through space, and calls go from the phone up to the satellite, which passes the call satellite to satellite until it downlinks at the Hub station in Tempe, Arizona.

- **Two-way satellite messaging**: he Iridium network also provides two-way satellite messaging service from supported Android smartphones[21].

- **Internet of Things (IoT) connectivity**: The Iridium satellite network is designed to support the most challenging IoT applications, including remote asset tracking, monitoring of connected vehicles, and remote sensing[22]. This application is the one that interests us the most.

- **Niche markets**: The Iridium satellite network offers communication services to a niche market of customers who require reliable services in areas of the planet not covered by traditional geosynchronous orbit communication satellite services[21].Users include journalists, explorers, and military units.

- **Global coverage**: The Iridium satellite network offers unparalleled coverage and connectivity for various applications[21].The network's unique constellation architecture makes it the only network that covers 100% of the planet[20].

The Iridium satellite network consists of a fully meshed network of 66 low-earth orbiting (LEO) cross-linked satellites, and 9 in-orbit spares, that ensure coverage over the entire globe in a constellation of six polar planes[23]. The satellites are cross-linked to provide reliable, low-latency, weather-resilient connections that enable communication anywhere in the world[20]. The cross-linked satellites are an essential part of the Iridium network, as they allow the satellites to communicate with each other and route calls and data around the globe[22]. The Iridium

network's unique constellation architecture makes it the only network that covers 100% of the planet, and the cross-linked satellites ensure that there are no coverage gaps[20][23]. The Iridium network's cross-linked satellites also provide shorter network registration times and low communications latency, making it an ideal choice for users who require reliable communication services in remote areas of the planet[23]. Overall, the cross-linked satellites from the Iridium satellite network are a crucial component of the network's architecture, enabling reliable, low-latency, weather-resilient connections that enable communication anywhere in the world. The cross-linked satellites ensure that there are no coverage gaps and provide shorter network registration times and low communications latency, making the Iridium network an ideal choice for users who require reliable communication services in remote areas of the planet.

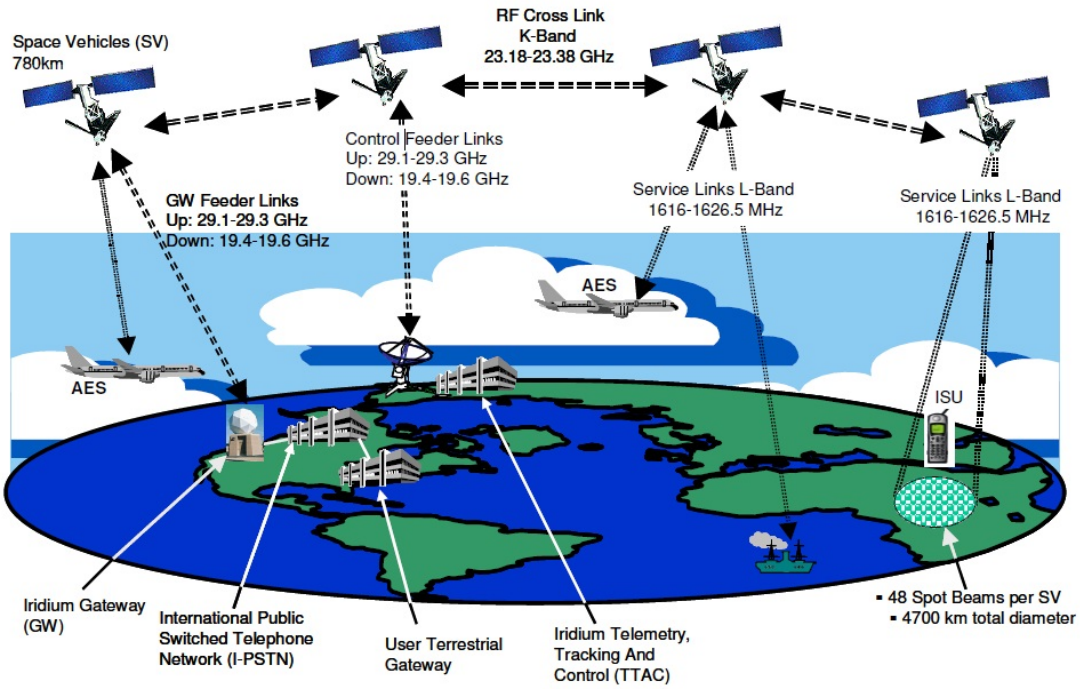

Figure 2.5.1: Operation of the Iridium satellite network on a big scale

For this project we needed a service that ensured the communication between the buoy and the ground station of OceanAcess in order to transmit the information to the customer, that is why the Iridium communication network is a great option for us;

- **Global coverage**: As said earlier the Iridium network has a 100% of the Earth covered, so the Arctic sea which is our goal would be available. Those crosslink satellites will ensure that there are no coverage gaps, making it ideal for monitoring in remote areas of the Arctic Sea [23].
- **Real-time data sharing**: The Iridium crosslink network allows scientists and researchers to share data in real time, even when they are outside the reach of traditional communication networks [20]. This feature is particularly useful for monitoring the Arctic sea by providing reliable communication services in remote areas.

- **Monitoring capabilities**: It can be used for monitoring a variety of sensors, including atmospheric monitors, altimeters for monitoring the sea surface, waves, and ocean drifters and buoys [24]. These capabilities make the Iridium satellite network an ideal choice for monitoring the Arctic Sea.

2.6 Why Iridium? And services implemented

As I said earlier, the Iridium satellites, will be the solution explored in this Thesis to overcome the communication between the buoy deployed on the Arctic Sea, and the ground station of Ocean Access. It was chosen because of some of the capabilities I talked about, earlier while mentioning the Iridium communication network. I'll list some of them so we can have the same point of view; Global coverage was essential, as the Arctic Sea expands to remote areas where some of the swarm and normal satellite communications don't reach, and also gives Ocean Access a suitable platform fit for the project's expansion to different areas all over the world. Reliability is key, reliable connectivity aligns with the project's need for dependable communication in challenging ocean environments. Also, Iridium serves as a reliable backup solution in case of cellular network outages, ensuring continuous data transmission and operational control. In addition, Iridium's two-way communication capabilities match the project's requirements for enabling the buoy's movement and operation in both sub-sea and ocean surface conditions. It is a great fit for the project also because the modems and satellites of Iridium are built to endure harsh conditions, both in space and on Earth, which suits the project's need for communication reliability in adverse weather and sea states. Iridium also offers customization options, allowing us to tailor the satellite communication solution to the specific demands of the buoy's design and function.
In essence, Iridium's global reach, reliability, backup capabilities, two-way communication, resilience to tough conditions, and customization options make it a strong contender to fulfil the communication requirements of the project's objective for enhanced ocean monitoring.

As Iridium was exposed as the solution that had to be explored different services were presented to be investigated, each of them had a different usage and application. The work was to find each of these Iridium services, which was most suitable for the project's different needs in different situations proposed by Ocean Access.

In this section, I am going to present the different services explored in this Thesis and a little explanation of why they were selected to be investigated.

2.6.1 SBD

Iridium SBD is a service that enables the transmission of short data messages between equipment and centralized computer systems[25]. It is a real-time, two-way messaging service that can be used anywhere in the world[25]. SBD is a packet-based service that is designed for frequent short data transmissions between equipment and centralized host computer systems. It is a simple and efficient service that is ideal for remote monitoring applications used for asset tracking, remote

telemetry, and pipeline monitoring. Also this service is especially suitable for low-power devices and applications where energy efficiency is crucial, making it ideal for IoT applications[26].

One of the significant advantages of SBD is its global coverage, providing communication in regions where traditional cellular networks might be unreliable or unavailable. Despite its emphasis on one-way communication, Iridium SBD also supports limited two-way communication. This enables devices to receive short messages or commands for basic remote control or configuration[25].

Another key feature of Iridium SBD is the reliability of Iridium's LEO satellite constellation as explained before, which ensures consistent connectivity, even in challenging environments and remote locations. Due to its cost-effective nature (as it involves transmitting small bursts of data), Iridium SBD is favored for applications that do not require continuous high-bandwidth communication[27].

But as we know this service was considered for this actual project for the submersible buoy that Ocean Access wants to deploy, so is Iridium SBD a service that could help on this project knowing what we know?
We know that the LEO constellation from Iridium provides reliable coverage from pole to pole, including the Arctic Sea, where our buoy will be deployed. This ensures that the submersible buoy can maintain connectivity and transmit data even in remote and challenging environments. It also allows periodic data transmission from the buoy to the central data point, which is what we need as our submersible buoy will have an alarm that will make the buoy float to the surface where is going to do the transmission of the data, this ensures that the data is promptly available for storage and analysis. The burst transmission technique used by SBD devices conserves power and enhances battery life, making it efficient and reliable for data transfer. LEO network enables low-latency connections compared to other types of satellite networks, this means minimal lag with ping times of less than one second, allowing for near real-time remote monitoring and data transfer. If we add this to the two-way messaging, allowing bidirectional communication between the buoy and the central data point, this last actor can be used to send information related to the behavior of the first, like changing the configuration file, as implemented in this Thesis. In conclusion we can say that the Iridium SBD service is a good option to be implemented for this project.

2.6.2 Certus 100

Iridium Certus 100 is a midband satellite service that provides reliable, truly global connectivity for vehicles, vessels, and aircraft all over the world[28]. It is optimized for solutions with strict size, weight, and power requirements, making it ideal for mobile, fixed, and portable operations like workforce communications, remote monitoring, and real-time asset control[28]. The key feature of Iridium Certus 100 is its ability to deliver broadband-level data speeds, enabling fast and efficient transfer of information. This is particularly beneficial in sectors requiring rapid data transmission, such as maritime, aviation, remote industrial operations, and emergency response[28][29].

The service is designed for global coverage, ensuring that users can maintain communication even in remote and challenging environments where traditional terrestrial networks may be limited. This makes Iridium Certus 100 a valuable option for businesses and organizations operating across the globe[30].

With Iridium Certus 100, users can take advantage of both voice and internet services. This multifunctionality enhances its utility in situations where reliable communication and data access are paramount. The service is compatible with a variety of devices, ranging from smartphones to specialized communication equipment[28][30].

Iridium Certus 100 can support email, messaging applications like WhatsApp, media sharing, telemetry reporting, file transfer, internet/VPN access, and up to two simultaneous high-quality voice calls[31]. It can also support low-resolution video transmission for surveillance and monitoring applications when combined with the latest in Iridium partner data compression technologies[31]. Iridium Certus 100 is delivered over the recently upgraded Iridium satellite network, which is the world's most advanced L-band satellite service platform[31].

As we did for the Iridium SBD service we want to know if the Certus 100 service is suitable for this project.
We know that Iridium Certus 100 is a satellite service that can be a good choice for a project involving a submersible buoy in the Arctic Sea. The service provides truly global coverage, as we said earlier, including the Arctic area as the SBD thanks to the LEO satellite network that Iridium has. This ensures reliable connectivity in those remote spots where traditional communication networks can not reach, it also minimizes the risk of signal loss or interruption. The service also supports IP data speeds of up to 88Kbps, which is sufficient for transmitting periodic data from the submersible buoy to the headquarters of Ocean Access as needed. This allows sending important information, such as sensor readings or environmental data, to be stored and analyzed. As said earlier "Iridium Certus 100 is optimized for solutions with strict size, weight, and power requirements" making it easy to integrate into the submersible buoy without adding significant weight or power consumption. As SBD, Certus 100 also has low latency, this means that like in the SBD the transmission time would be less than 1 second, ensuring a near real-time data transmission. Is also suitable for IoT applications, enabling Ocean Access to monitor and control the buoy remotely. So like the SBD we can ensure that Certus 100 could be a possible option to implement into the project.

2.6.3 RUDICS

RUDICS is a data service that provides customers with the ability to transfer large amounts of data reliably and affordably using multiple protocols[32]. RUDICS enables both Mobile Originated (MO) and Mobile Terminated (MT) circuit switched data transfers[32].

At its core, Iridium RUDICS is designed to enable router-based communication, allowing devices and systems to establish reliable data connections using the Iridium satellite network. This service is particularly valuable for applications that demand continuous and uninterrupted data transfer[32].

One of the key features of Iridium RUDICS is its ability to support unrestricted digital internetworking connectivity solutions. This means that users can establish efficient, bidirectional data links between remote locations and centralized systems. It's a versatile solution suitable for various sectors, including utilities, energy, maritime, transportation, and remote industrial operations[33][34]. With that we can say that the service leverages the global coverage of the Iridium satellite constellation, ensuring that communication can be maintained even in the most remote and isolated areas. This is essential for industries that rely on consistent data exchange regardless of location.

Iridium RUDICS offers various data plans and configurations to accommodate diverse communication needs and usage patterns. Users can select plans that align with their data requirements and operational demands[35].

As for the last two services we want to know if the Iridium RUDICS is a valid service to be implemented in this project.
Iridium RUDICS can be a suitable choice for a project involving a submersible buoy in the Arctic Sea that requires periodic data transmission to a central data point, for several reasons. Firstly, it provides reliable connectivity, even in remote and challenging environments, thanks to its utilization of the Iridium satellite network, as we said in the last two services, which ensures data can be transmitted globally regardless of location. Its optimized circuit-switched data approach also makes it efficient and cost-effective for the regular data transfers needed by the project. Additionally, RUDICS is well-suited for integrating with remote oceanographic assets like buoys, enabling effective data collection and communication. Its compatibility with high-value mobile sensor platforms indicates its potential to handle the buoy project's data transmission requirements. Notably, RUDICS has a track record of successful use in remote projects, bolstering its capability to perform in harsh and distant locations. However, considerations such as data volume, transmission frequency, and budget should also be kept in mind. So for this service consulting with Iridium is necessary so they can guide the project in the right direction for the project goal. But otherwise it is seen that RUDICS is also a valuable service that could be implemented to the project and give potential value performance for the project needs.

CHAPTER

THREE

IMPLEMENTATION

In the landscape of satellite communication services, the usage of dedicated modems has enabled the implementation of distinct solutions to address specific communication demands. This Thesis as mentioned before delves into three of the Iridium services; SBD, Certus 100, and RUDICS. Each of these services has been tailored to meet unique connectivity requirements and has been associated with its unique modem configuration.

Each service has its own qualities but we want to know which of these services would be a better solution for the satellite communication dilemma. For that, I'll expose the principal capacities of each service to introduce the main implementation part which is going to be presented what modem was picked for each service and the thought process behind it.

- **SBD**: Serves as an efficient means of transmitting concise data bursts from remote locations to central systems. This service finds utility in applications such as remote monitoring, tracking, and IoT.

- **Certus 100**: Is characterized by its ability to offer high-speed broadband connectivity in areas where conventional networks encounter limitations. This service demands a modem to harness the capabilities of the Iridium network for swift data transfer, serving sectors such as maritime, aviation, and emergency response.

- **RUDICS**: In scenarios requiring comprehensive communication links, RUDICS operates on the framework of router-based communication. This specialized service relies on a dedicated modem to establish bidirectional data connections between remote locales and centralized systems. Its adaptability makes it pertinent for industries like utilities, energy, and transportation that require consistent and reliable data exchange.

Considering these different implementations, the selection of the most suitable service becomes a vital consideration. In the context we are in, the submersible buoy has challenging aspects to be taken into account. The feasibility of implementing these services for the submersible buoy needs research and thought processes to fulfil the communication requisites that Ocean Access has. The assessment of these services, each paired with its tailored modem, will facilitate informed decision-making, and the solution for the communication process for Ocean Access.

As said earlier for each of the services a specific modem has been picked to be the one considered to implement into the project. So for each of these services, a modem is going to be explained and reasoned on why it has been picked and the theoretical benefits in the implementation to the project with the two cases Ocean Access wanted to be solved.

3.1 SBD - 9603N Iridium

The first service that was contemplated was the SBD, for this service specific the modem 9603N from Iridium was picked. The Iridium 9603N modem is a single-board transceiver that provides global coverage through the Iridium constellation. It is a small form factor module that offers unmatched flexibility. The 9603N needs to be connected to a PCB in order to be functionally useful. This was already implemented at the Ocean Access office where it was connected through a PCB to the STM32 micro-controller, but there was an error where the modem itself did not connect correctly to the main program implemented by Ocean Access, this is going to be discussed in the Results part. Anyway, the 9603N as said before is the smallest commercially available satellite module, one-fourth the volume and half the footprint of its predecessor, the Iridium 9602. The Iridium 9603N delivers Short Burst Data, which allows for the transmission of short messages or small amounts of data/telemetry. This modem is lightweight and low-power, making it highly flexible for integration into various applications, whether fixed, mobile, or battery-powered, this is a great feature for the project we are working on. Finally, the modem 9603N is used in various applications, including IoT devices, remote monitoring - our case -, asset tracking, and communication in remote or inaccessible areas.

Figure 3.1.1: Photo of the Iridium 9603N modem

For this project, some key factors must be taken into account in order to have the modem implemented into the buoy. The Iridium 9603N modem's capability to operate efficiently in intermittent communication mode aligns with the project's operational pattern. Since the buoy will spend most of its time submerged and surface only for data transmission, the modem's design suits this usage.

The ability of the 9603N modem to transmit different types of data supports the diverse needs of the potential clients the project could have in the future. Its adaptability supplies to scenarios like an oil rig monitoring or monitoring the weather of a certain region of the ocean to be able to tell the ships where to navigate without danger. The 9603N modem is capable of transmitting data received from AVL equipment to the low-orbit Iridium satellite. This suggests the modem can handle different data types, including location information.

One of the key features that the 9603N offers is that has compact size and low-profile design - as said before - makes it suitable for integration within the buoy's limited space. It is the smaller modem out of the three with these dimensions; 31.5 mm X 29.6 mm x 8.1 mm, which aligns with the factor of conserving space inside the buoy. Also, the weight would not be a problem because the total weight of the modem is 11.4g the lightest in the market.

In the field of power consumption, another key factor, the 9603N has a low power consumption which addresses the requirements for energy-efficient components. It needs an input voltage of 5.0V +/- .5V DC which is very low, for the different states of the modem it has different power parameters. For the Idle mode, the peak performance is 156mA, and the average is 34mA. For the transmission current, the peak is 1.3A, while the average is 145mA. The receive current has different values, the peak performance is 156mA, and the average is 39mA. Finally, for the SBD transfer, we have that the average current is 158mA, while the average power consumption is less than 0.8W. These numbers indicate that the Iridium 9603N is a low-power consumption modem that aligns with the specifications that the buoy needs.

This project is focused mostly on monitoring and transmitting data, but one of the objectives is to send information from places where usual communications cannot be transmitted. That is why using the 9603N modem is useful, this modem is designed to withstand harsh environments. Its rugged construction and ability to function in extreme weather conditions align with the project's need for durability. It can do it because the operational temperature ranges from - 40°C to +85°C cover most of the extreme conditions on Earth.

Like all the Iridium products, the 9603N modem's connectivity through the Iridium network ensures reliable communication from even the remotest areas of the world, which ensures a potential growth to the number of projects that this modem can be effective. Also, it has perfect usage in the buoy project, where we want to drop it in remote locations and gather data, so having a modem that ensures reliable communication with the server is a key factor.

The Iridium 9603N SBD modem works differently as the next two modems, this modem works through messages, while the other two work with IP-based transmissions. The 9603N has on the low end a capacity of 270B per message and on the high end a capacity of 340B per message. So what the 9603N does is transmit short messages or small amounts of data with sizes ranging from 270-340B, which allows us to transmit any type of message with low latency to any destination we want from anywhere in the world.

For the connection with the buoy the modem has to be connected to the STM32 micro-controller that powers all the components of the buoy. For this modem the 9603N, the connection will be via connecting the appropriate pins of the modem to the corresponding pins of the micro-controller, such as the UART pins for serial communication. The 9603N communicates with the STM32 using serial communication, typically through an RS-232 serial port. Once the hardware connection and the serial communication are established, the software logic has to be developed - which we are going to talk about in the next section -, this involves sending commands and receiving responses from the modem using the UART interface.

After having all the information about the Iridium 9603N modem, two cases were presented by Ocean Access to see if the modem we are exploring, is suited for the operations the company wants to conduct.

The first case is to see if the modem could be used as an emergency backup to LTE, for example, if the LTE equipment is broken (trawler, harsh weather, unforeseen wear and tear, LTE network downtime). So in this case what is needed, it would be necessary to send out an SOS containing the location of the buoy and the battery percentage. The 9603N modem has all the features to be used as an emergency backup to LTE. First of all the global coverage of the Iridium satellite network ensures communication connectivity even in remote and isolated areas where LTE might not be available. This global reach is essential for maintaining communication in emergency situations, especially at sea. The Iridium 9603N modem can be programmed to initiate an emergency communication protocol in situations where the LTE connection becomes unreliable, with the usage of the AT& commands - which are used to communicate the modem with the micro-controller - where one of them triggers a message of a maximum of 340B that indicates the SOS containing the location of the buoy that can be extracted from the location information of the satellite network and from the GPS receiver, and the battery percentage from the micro-controller connected through the UART connection to the modem. Also, the 9603N can handle intermittent operation, so when an emergency condition is detected, the modem can activate and establish communication with the Iridium satellite network from the Idle mode. Also, the modem's intermittent operation aligns with the buoy's behaviour of surfacing periodically for data transmission. The Iridium 9603N modem's low power consumption is advantageous for emergency scenarios, ensuring that energy resources are conserved for critical operations. This allows the modem to initiate and maintain communication without overly depleting the buoy's battery, also not consuming roughly any battery while in Idle mode. The compact design ensures that it can

be easily integrated into the buoy's structure without taking up excessive space. Its low-profile form factor aligns with the preference for minimizing the modem's impact on the buoy's internal layout, which for an emergency backup is what we want to not occupy much space and consume less power the better. Finally, the modem's rugged design is also a fantastic quality for withstanding the challenging conditions of the sea, including adverse weather and potential physical impacts. Its durability ensures reliable operation even in harsh environments, where LTE or common communication solutions can not.

In summary, the Iridium 9603N modem is well-equipped to handle emergency backup scenarios and transmit SOS messages from a submersible buoy in the sea. Its global connectivity, capability to send critical SOS messages with accurate location and battery data, intermittent operation, low power usage, compact design, and ruggedness make it a suitable solution for ensuring communication in emergency situations.

The second case exposed by Ocean Access is that, if the buoy is placed in areas too remote for LTE or other types of communication, for example, the buoy is placed in the Arctic Sea in order to do long-term monitoring of environmental conditions. In this case, it could be anywhere from 4kB a day on the low end to 500kB a day on the high end. In both cases, sending data once a week would be necessary so it would be 4*7kB on the low end and 500*7kB on the high end. As said earlier the connectivity would not be a problem thanks to the connection of the modem to the Iridium satellite network that gives the project global connectivity throughout the entire world, so the problem of dropping off the buoy into the Arctic Sea would not be a problem in order to maintain connectivity. Also, for the harsh weather conditions, the rugged design of the modem will be perfect to maintain the communication available as explained in the previous paragraph. The data transmission rate from the Iridium 9603N is that it can send 1 message up to 340bytes per 20s at its maximum. With that, we know how is going to handle the transmissions ranging from 4kB to 500kB a day, or from 28kB to 3.5MB a day for one week of transmissions. Based on the calculations from the table 3.1.1, for the small transmissions the modem can comfortably transmit all the accumulated data even if it has to send all the data once a week. On the higher end, it is different in the case of sending each day's data, it has to take more than 8 hours to transmit all the data, this is a huge amount of time that can be divided into different time frames during the day programming an alarm to submerge and surface every now and then to send data, and not to send it all at once. But in the case of sending all the data of a week on a specific day, it is impossible because the transmission time will be superior to 48 hours so a different approach would be necessary in this case for this modem. We can say that the Iridium 9603N modem can handle these payloads but in the higher-end sizes, a different approach to send data has to be reconsidered.

Data Transmission Requirement	Total Data	Messages Needed	Time Required
Low-End (4kB per day)	28kB per week	83 messages	27.67 minutes
High-End (500kB per day)	3500kB per week	10295 messages	57.19 hours
Individual Data Calculation (Per Day)			
Low-End (4kB per day)	4kB	12 messages	4 minutes
High-End (500kB per day)	500kB	1471 messages	8.18 hours

Table 3.1.1: Iridium 9603N Modem Data Transmission Time and Calculations

3.1.1 SBD Software Implementation

The SBD service has been the only one that could have been implemented into the software of the normal communication system of the buoy. For this implementation, it was necessary to implement the AT& commands that Iridium supports its product with, to be able to connect the modem with the Iridium satellite network. For this implementation, the normal communication system of messaging from Ocean Access was used as the base of the SBD communication. So for this explanation, we are going to take into account that all the software and hardware connections that the simple communication uses are implemented.

For this implementation, a new class was created - iridium_sbd_drv.h.cpp - where all the main functioning of the SBD communication system was thought of. To know how the AT& commands work here in the next label there is a great explanation on how it works, and the main idea when implementing this solution for the communication between the buoy and the satellite network 3.1.

```
1 // Command structure: AT-commands
2
3 // Procedure for connecting and sending message:
4 // AT-> OK -------- we are in contact with the modem
5 // AT+SBDWT=MESSAGE -> OK -------- we have written
     message to modem, the message is stored in modem
     buffer
6 // AT+SBDIX -> +SBDIX: int, int, int, int, int, int
     -------- if the first int is not 0, repeat AT+SBDIX
```

Listing 3.1: Overall functioning of the communication with AT commands

First of all, we send an AT and wait for the response of an OK to know if we are in contact with the modem. After that, once we are in contact with the modem, we have to write the message inside the modem buffer where we use the AT+SBDWT, and if we receive an OK, that means our message is written and waiting to be sent. So finally with the command AT+SBDIX, we sent our message that was inside the buffer.

In the class, I mentioned before, we have the next usage for this AT command idea. We are working with different SBD commands, states, and transitions, to keep track of our buoy and the state of the communication process. This is extremely useful for the implementation because we can model our abstract process into a ruled and categorical functioning of different states, commands, and transitions as we can see in the appendices on the listing 1.

Before entering into the main process of how SBD communication works, a simple explanation of the commands, the states, and the transitions may help in a deep understanding of all the processes. All of the commands, transitions, and states are listed in the code that is in the appendices 1. The transitions depend on the state of the "buoy" because for each state there is a case in the sm_process function 2, that takes into account what transition we are going through and does its work depending on that, changing the oldState for the currentSate which will be updated depending on the transition. The first case is if the modem is off ST_SBD_OFF, so if our transition is SBD_MODEM_BOOTED, then the function will update the current state to be on. Likewise, if the transition is SBD_POWER_OFF_MODEM, the system will send to the main program that the connection has to be closed. This functioning is the same for all the states and transitions over the function that can be seen in the appendices. The commands we are going to use for the communication process, need an encoding so our data is not vulnerable, and we also used that function to complete the final message that would be written inside the modem's buffer. So in the function encode 2, what we do is allocate memory in an encoded message which takes into account the rawData that is our not encoded AT& command, and uses 10 additional strings to encode it. It constructs an AT command by concatenating various components based on the value of the encode_cmd variable which indicates to us what command we are using inside our enumeration. Each of the strings created are always concatenated with the AT string - which is the form of what Iridium needs to understand the message -. So in that function depending on the command that is used, a different encoding process is done.

Once we know how all the transitions, states, and commands work inside their specific functions, a more global look is what is going to be explained next. The main communication process is enherited by the class DigitalTwins which controls all of the communication processes in the software of Ocean Access. In the sm_process function to send something to the modem, let's say, a delete buffer or to write the message, we use the functions inherited by the class DigitalTwins. So it works as follows in the function load_and_send, which is the one that we use to send when transmitting to the modem. We first check for the length of the message so we have the memory allocated in just the necessary space and then calculate the number of packets that will be needed for sending the message. After that, we are going to send each packet so the receiver can reconstruct the final message. Then we send to the DigitalTwin our start loading messages and wait for a response. If we receive a response that is not the OK we talked about in the first stage or SBDNETACQUIRED which means that the network is not acquired, the loading fails. Otherwise, we take the message in the different packets and copy the content inside the sending buffer, and all of that is sent to the modem through the send function of DigitalTwins. After that, we do the same check of the response of the modem, whether it accepted the command/message or not. Finally, we empty the buffer if everything went the right way, and tell the SBD modem to send the payload to the satellite network with the DigitalTwin function send, with the content of transmit_to_satellite_msg, as in the first step of the loading messages. Once we receive the response we check if we got an OK or SBDNETACQUIRED, if not, then we know that our message has not been sent

to the satellite network and that we have to try again. But if we get an OK or an SBDNETACQUIRED, then we can be sure that our message has been sent into the satellite network and will arrive at its destination.

3.1.2 Algorithm Update config file

One of the software applications that Ocean Access wanted to implement was an algorithm that updates the configuration file of the SBD modem while it is functioning, without having to restart physically. For this implementation, we had to take a different approach to what was already implemented, because what was implemented was that we updated the config file through what was written inside the SIM card that the modem had, so we could not update it while it was running. Next, I will show with code snippets what was done to be able to implement the new algorithm.

```
char config[] = "# Specify which connection to use to communicate with relay boards\n"
  "[relays]\n"
  "i2cNumber = 2\n"
  "# Communication channel used to send/receive\n"
  "# data to/from the seabed (or other sensors that act like the seabed)\n"
  "# Means, data from this device need to be sent to the shore\n"
  "[[inComChan]]\n"
  "name = \"seabed\"\n"
  "[inComChan.uart]\n"
  "uartNumber = 3\n"
  "baudrate = 115200\n"
  "# Communication channels used to send/receive\n"
  "# data to / from the shore\n"
  "[[outComChan]]\n"
  "name = \"swarm\"\n"
  "band = \"narrowband\"\n"
  "realtime = \"no\"\n"
  "relayBoardAddress = 24\n"
  "relayChannel = 4\n"
  "driver = \"swarmDrv\"\n"
  "priority = 0\n"
  "[outComChan.uart]\n"
  "uartNumber = 2\n"
  "baudrate = 115200\n"
  "[[outComChan]]\n"
  "name = \"lte\"\n"
  "band = \"broadband\"\n"
  "realtime = \"yes\"\n"
  "relayBoardAddress = 24\n"
  "relayChannel = 3\n"
  "priority = 1\n"
```

```
    "[outComChan.eth]\n"
    "serverAddress = \"192.168.1.40\"\n"
    "serverPort = 10\n";
```

Listing 3.2: Config file example

First of all, the communication process that Ocean Access already had was used. In the class controller.cpp the function that takes the config file and does the parsing and updates was done. This class is the controller specifically for communication channels. It manages the sending of processes on all output communication channels, receives data, and chooses where is sent. The function that was implemented takes the config file and does the corresponding parsing and updates, or it changes everything overwriting all the file for a new configuration.

```
void Controller::updateConfigFile(char* newConfig[],
   toml_table_t* oldConfig) {
  char errbuf[100];
  toml_table_t* conf = toml_parse(newConfig, errbuf,
   100);
  /**
     * Build the software architecture with the new
   specificiations of the config file or update the old
   ones
     */

   // Create and configure the relay board controller
  toml_table_t* relaysConf = toml_table_in(conf, "relays
   ");
  toml_table_t* oldrelaysConf = toml_table_in(oldConfig,
    "relays");
  if(!relaysConf)
    printf("Missing [relays] in configuration file, so
   not updating it\n");
  else if(!oldrelaysConf or oldrelaysConf != relaysConf)
    configureRelayController(relaysConf);
  else
    configureRelayController(oldrelaysConf);

  // Generate output communications channels
  toml_array_t* outComChanArray = toml_array_in(conf, "
   outComChan");
  toml_array_t* oldoutComChanArray = toml_array_in(
   oldConfig, "outComChan");
  if(!outComChanArray)
    printf("Missing [[outComChan]] in configuration file
   \n");
  else if(!oldoutComChanArray or oldoutComChanArray !=
   outComChanArray)
    generateComsCh(outComChanArray, true);
  else
```

```
26      generateComsCh(oldoutComChanArray, true);
27
28    // Generate input communications channels
29    toml_array_t* inComChanArray = toml_array_in(conf, "
       inComChan");
30    toml_array_t* oldinComChanArray = toml_array_in(
       oldConfig, "inComChan");
31      if(!inComChanArray)
32          printf("Missing [[inComChan]] in configuration
       file\n");
33    else if(!oldinComChanArray or oldinComChanArray !=
       inComChanArray)
34      generateComsCh(inComChanArray, false);
35    else
36      generateComsCh(oldinComChanArray, false);
37
38    // Configuring sensors
39      toml_array_t* sensorConf = toml_array_in(conf, "
       sensor");
40    toml_array_t* oldsensorConf = toml_array_in(oldConfig,
        "sensor");
41    if(!sensorConf)
42          printf("configuration file : missing [sensor] \n
       ");
43    else if(!oldsensorConf or oldsensorConf != sensorConf)
44      generateSensor(sensorConf);
45    else
46      generateSensor(oldsensorConf);
47
48    toml_free(conf);
49    toml_free(oldConfig); // free memory
50    printf("Software architecture creation completed, with
        updates\n");
51  }
```

Listing 3.3: Function for parsing and updating the config file in the controller.cpp class

This function just takes the information received with the config array and parses it with the toml format that was already implemented. Once is it all parsed it takes the different key parts of the config file and updates it depending on if the new information is different or not from the old config file. If the new information is different we use it for creating the new software architecture of the project, and if not we use the old configuration.

After having that function implemented, to be able to "fool" the system that this is the configuration file, we have to simulate that a message has entered the system. For this in the message class message.h I will add a command which we can work with to fool the system. The message class is used to represent a real

message entering the system, where a message is the only object used to communicate between threads. These messages are composed of a command, and data, in our case we will create a command to fool the system and the data will be the new config file or the updates we want to add. So I will add our new command to the enumeration of commands that are used for the communication.

```
// Commands that can be exchanged between threads
enum Commands{
  MSGCMD_UNDEFINDED, // Initial state of every message created
  MSGCMD_OPEN_CONNECTION, // To open the connection
  MSGCMD_CLOSE_CONNECTION, // To close the connection
  MSGCMD_CONNECTIONS_CLOSED,
  MSGCMD_START_SPEEDTEST,
  MSGCMD_STOP_SPEEDTEST,
  MSGCMD_SEND, // To send the data (don't forget to give data with)
  MSGCMD_SEND_FAILED, // When the com choosed failed to transmit the message
  MSGCMD_SEND_SUCCESS, // When the com choosed success to transmit the message
  MSGCMD_RECEIVED, // When data have been received (by seabed most of the time)
  MSGCMD_INIT, // To init a SM (of controller for example)
  MSGCMD_SLEEP_TIMEOUT, // Notify that the sleep timer overflowed
  MSGCMD_CONNECTION_TIMEOUT, // Notify that the connection timer overflowed
  MSGCMD_EXITSLEEP, // Notify controller to exit sleep state
  MSGCMD_NO_CHOICE, // No com are in the leaderboard after the choice algo
  MSGCMD_NO_MORE_CHOICE, // No com success to transmit
  MSGCMD_CHOICE_DONE, // When leaderboard is created, choice algo done
  MSGCMD_ENDOFENUM, // Used to check if the message type is valid
    MSGCMD_UPDATE_CONFIG_FILE // Used to fool the system to update the config file
};
```

Listing 3.4: Enum for the different message commands for the communication processes

After adding the message command to the message.h class, the data needs to be added. In the controller.cpp class, there is a function the sm_process, which takes the messages and controls where are they going and their treatment. So in that switch case, we will add a case for our message command. In this switch case, the only thing we will do is call the function that parses the data of the message.

This call to our function will be done inside the ST_WAIT case, where the system waits for a message to be received. For this to function in the controller.cpp class a function called ControllerTask that controls everything, has a switch case to be able to use the sm_process function as the controller of the message commands, where there I will add the MSGCMD_UPDATE_CONFIG_FILE.

```
1
2 Message msg;
3
4 case MSGCMD_INIT: case MSGCMD_SLEEP_TIMEOUT: case
   MSGCMD_EXITSLEEP:
5       case MSGCMD_CONNECTION_TIMEOUT: case
   MSGCMD_NO_CHOICE:
6       case MSGCMD_CHOICE_DONE: case MSGCMD_SEND_FAILED:
7       case MSGCMD_NO_MORE_CHOICE: case
   MSGCMD_SEND_SUCCESS:
8       case MSGCMD_CONNECTIONS_CLOSED: case
   MSGCMD_UPDATE_CONFIG_FILE:
9     // Give to the SM to process
10    controller->sm_process(&msg);
11        break;
```

Listing 3.5: Switch case for the ControllerTask

```
1 case ST_WAIT :
2     if(msg->getCmd() == MSGCMD_RECEIVED)
3     {
4       printf("Message received from a channel\n");
5       // Stop the sleep timer (delay before enter sleep
   mode)
6       HAL_TIM_Base_Stop_IT(timSleepHandler);
7       // Take the first message to transmit
8       currentData = msgToTransmit.at(0);
9       msgToTransmit.erase(msgToTransmit.begin());
10      currentState = ST_OPENCHAN;
11    }
12        if(msg->getCmd() == MSGCMD_UPDATE_CONFIG_FILE)
13    {
14      printf("Message to update the config file\n")
15      // Call the function to update the config file
16      updateConfigFile(msg.getData(), oldConfig);
17    }
```

Listing 3.6: Switch case in the sm_process function

The oldConfig will be an object that controller.cpp can access, this object will be the actual config file, the one that the modem is using at that moment.

Then in the ControllerTask function, which is the task of the communication controller, only declare a message from the message class, add the command that

we created, and add the data that we want to upload to the config file. I will add it once before the infinite loop, to not add it periodically.

```
void ControllerTask(void* argument)
{
    *** WORK DONE HERE ***
    Message msg;
    msg.setCmd(MSGCMD_UPDATE_CONFIG_FILE);
    msg.setData("New Config File or updates that we want
    to make");

    *** INFINITE LOOP HERE ***
}
```

Listing 3.7: ControllerTask function where we add the data needed

With all of this work done, what we do is simulate that a message entered the system and treat it like any other, but adding the command MSGCMD_UPDATE_CONFIG_FILE, and implementing the function updateConfigFile(), what we do is that the data that we receive is the one that we want, updating the configuration file.

3.2 Certus 100 - Quicksilver QS-100 NAL Research

For the Certus 100, the modem Quicksilver QS-100 from the company NAL Research was picked. The Quicksilver QS-100 is a Wi-Fi-enabled modem that provides IP-based communications anywhere in the world[**2**]. It is based on the Iridium Certus 9770 Transceiver and delivers the latest mid-band service in a compact, rugged package for globally available connectivity, and is prepared to be used in harsh environments[**3**]. The modem is suitable for both standalone usage and embedded platform integration - what we want in this project, is to be embedded into the buoy itself - and employs a small detachable passive, low-gain, Omni-directional antenna, simplifying low[**1**]. So the Quicksilver QS-100's flexible integration architecture makes it the ideal upgrade from narrowband solutions to provide mid-band throughput performance.

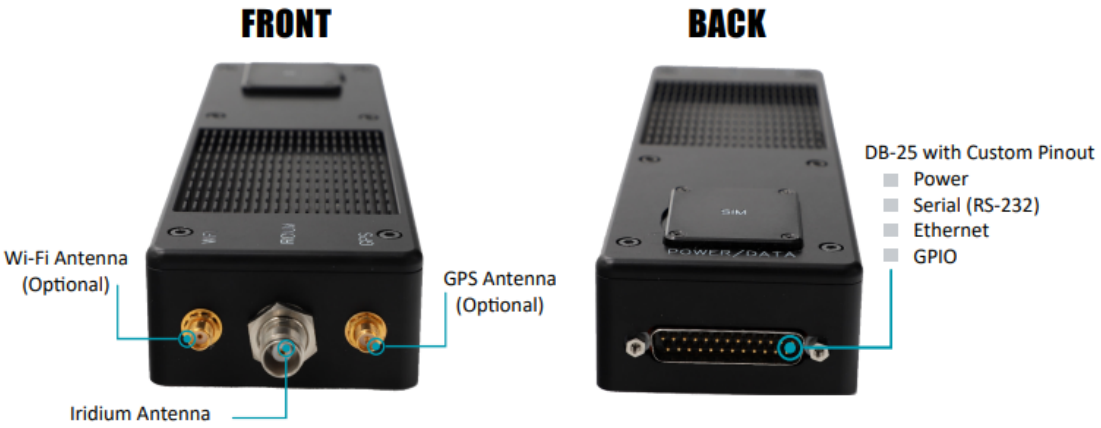

Figure 3.2.1: Photo of the modem Quicksilver QS-100, indicating some of the important components

For this project, we had some specifications that we needed to be covered to be able to implement the modem into the buoy. We know that the Quicksilver QS-100 modem demonstrates its versatility by accommodating various data types, making it an excellent choice for clients with diverse monitoring needs like the project we are working on because as said by Ocean Access the goal of the submersible buoy is to be able to work on different monitoring projects that could go from if an oil rig requires periodic updates about its operations - a common practice in Norway - to monitoring the ecosystem in areas with aquaculture that needs a real-time pH measurement, the QS-100 can efficiently handle both cases.

Given the project's submersible buoy nature where the buoy surfaces intermittently to transmit data, the QS-100 modem's operational pattern is well-suited. It can handle scheduled data transmissions, conserving power during periods of submersion and efficiently utilizing energy during transmission.

One of the other big specifications that we need to cover in this project is the compact design of the buoy, where the modem has to be installed. The QS-100 is of the three modems the biggest, its dimensions are 21.34 cm x 6.86 cm x 3.81 cm, which is a bit big for the buoy itself but it can fit in.

In the order of power consumption, the QS-100 is also the modem that needs more power to work, it needs an input voltage of 10-32V DC in order to function correctly. In the idle mode, it will consume 2.0W and at a full rate, it can consume in the order of 4.8-9.6W. For that, it is recommended a power supply of 18W or higher.

On the other hand the rugged design of the QS-100 modem enables it to endure the Arctic Sea's extreme conditions, with an operational temperature range of approximately -40°C to +65°C.

As said earlier the QS-100 modem's global connectivity, facilitated by the Iridium Certus 9770 Transceiver, allows it to transmit data reliably from virtually anywhere around the world, even in the remote Arctic Sea location where we wanted to drop off our buoy. This is a very good option to have in the buoy because not only now but in the future, more extreme projects could be presented into Ocean Access and this feature would help to not be afraid to go to remote locations with it and have a reliable connection to the buoy.

Finally we know that the QS-100 supports a mid-band throughput performance supporting data transmissions ranging from 4kB to 500kB per day, which aligns with the varying data needs of different applications. For example, if the oil rig requires a daily status update of around 10kB and the sea region with aquaculture requires hourly pH measurements averaging 50kB per day, the modem could handle both scenarios effectively.

For the connection with the buoy as said earlier in the previous service implementation, we use an STM32 micro-controller to connect the hardware with the functionalities of the buoy. In this case, the Quicksilver QS-100 uses Ethernet to communicate with the STM32, where it functions as a link between the control system of the buoy and the Iridium satellite network. As we already know, one of the key features of the Quicksilver QS-100 is that supports different IP protocols that we can use such as TCP, UDP, HTTP, HTTPS, etc. As said by the manufacturer - NAL Research - the best protocols to use are TCP and UDP when we are using the Ethernet pin port, that is what we are going to use to implement it.

After knowing all the information about the Quicksilver QS-100, we want to explore the two cases that Ocean Access brought to the table.

The first case is to see if the modem could be used as an emergency backup to LTE, for example, if the LTE equipment is broken (trawler, harsh weather, unforeseen wear and tear, LTE network downtime). So in this case what is needed, it would be necessary to send out an SOS containing the location of the buoy and the battery percentage. For this case, the QS-100 as said earlier has global connectivity around the world so the connection to the buoy would not be a problem, so it would be suitable for an emergency situation where the LTE may not be available. The QS-100 has a modular design, so this flexibility we talked about earlier allows it to be easily integrated into existing systems as an emergency

backup. The rugged and compact design that the QS-100 has, would make it a good choice for an emergency solution so it could be functioning although for the harsh conditions or the unforeseen wear and tear the LTE may not be able to overcome. In the case of an emergency, the QS-100 is capable of sending an SOS message containing the location of the buoy and the battery percentage. Because the Quicksilver QS-100 works with an internet connection, that means that once installed in the buoy and configured before being deployed, a local server will be available where you can transmit your data. So you can configure an SOS message that when the LTE fails or the buoy goes into the emergency state, the modem could send the location with the GPS that is inside and the battery percentage of the buoy. All the AT& commands used for the SBD service, here are not needed, because they were implemented to be backwards compatible with some of the old products from NAL Research and Iridium, so they are not necessary in order to make the internet connection.

The second case exposed by Ocean Access is that, if the buoy is placed in areas too remote for LTE or other types of communication, for example, the buoy is placed in the Arctic Sea in order to do long-term monitoring of environmental conditions. In this case, it could be anywhere from 4kB a day on the low end to 500kB a day on the high end. In both cases, sending data once a week would be necessary so it would be 4*7kB on the low end and 500*7kB on the high end. As said in the previous case the connectivity would not be a problem thanks to the modem based on the Certus 9770 Transceiver that enables connection all around the world also in the Arctic Sea. The Quicksilver QS-100 modem has an upload data rate of 22Kbps and a download data rate of 88Kbps. To determine if the modem can handle sending 4kB or 500kB of data a day, we need to calculate the time it would take to send that amount of data at the given data rates 3.2.1. Based on these calculations, the modem should be able to handle sending 4kB of data a day with ease but may struggle with sending 500kB of data a day due to the longer upload and download times. If data only needs to be sent once a week, the modem should be able to handle the low and high-end data amounts, but the high-end amount will take significantly longer to upload and download. These calculations were based on the upload and download data rates provided by the manufacturer [36]. But with all these calculations, we can say that the QS-100 will be capable of handling these payload sizes.

Data Amount	**Upload Time**	**Download Time**
4kB/day	0.18 minutes (10.8 seconds)	0.045 minutes (2.7 seconds)
500kB/day	22.73 minutes (1364 seconds)	5.68 minutes (341 seconds)
7*4kB/week	1.27 minutes (76.2 seconds)	0.32 minutes (19.2 seconds)
7*500kB/week	159.09 minutes (9545 seconds)	39.77 minutes (2386 seconds)

Table 3.2.1: Data transfer times for the Quicksilver QS-100 modem

3.3 RUDICS - A3LA-RG NAL Research

For the RUDICS implementation, the modem A3LA-RG from NAL Research was picked. This modem enables SBD, SMS, Data Switch, and RUDICS connectivity to the Iridium network, and it has an internal GPS receiver. It is essentially an A3LA-RM modem (a different version from NAL Research) with an added GPS. It also has a comprised of an Iridium 9523 transceiver. The A3LA-RG is a small, ruggedized satellite tracker with internal antennas and a battery. Is a reliable and versatile solution for providing connectivity in remote and harsh environments. Like the Quicksilver QS-100, it is suitable for both standalone usage and embedded platform integration. The A3LA-RG modem works by enabling RUDICS connectivity to the Iridium network, among other types of connectivity. RUDICS stands for "Router-Based Unrestricted Digital Internetworking Connectivity Solution" and is a protocol that allows for the transfer of IP data over the Iridium network. The A3LA-RG modem, therefore, uses the Iridium 9523 transceiver to connect to the Iridium satellite network and enable RUDICS connectivity. Once connected, the modem can transfer IP data over the network using the RUDICS protocol.

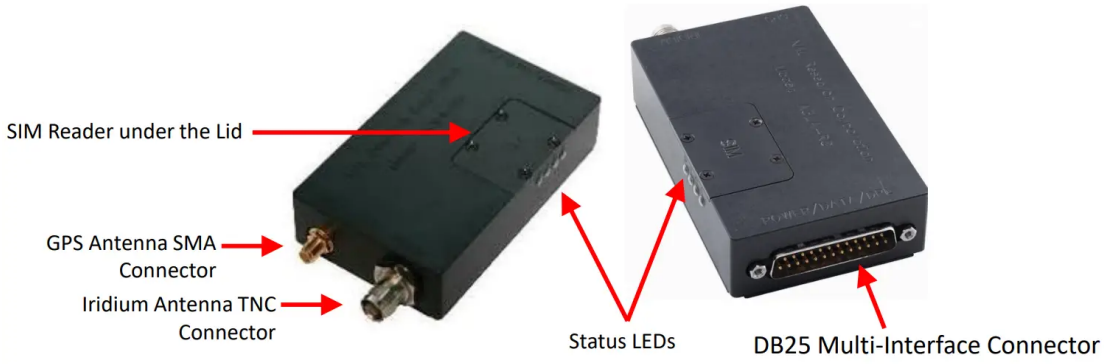

Figure 3.3.1: Photo of the A3LA-RG modem, indicating some of the important components

As for the Quicksilver QS-100, in this project, some goals were specific to be completed to be able to implement the service in the right conditions so that this service can make the tasks that Ocean Access needs. As we said in the previous paragraph the A3LA-RG modem enables SBD, Data Switch, and RUDICS connectivity to the Iridium network. This means that it can transmit different types of data depending on the usage that the client needs, such as informing an oil rig of what is happening or measuring the pH of the water in a sea region with aquaculture.

Because of the compact design of the buoy we need a modem that is capable of being embedded inside the buoy with ease. In this case, this is not a problem because the dimensions of the modem are 102 mm x 61 mm x 24 mm, which are relatively small and can fit easily in the buoy. Also, the weight would not be a problem because it only weighs about 201g.

In the power consumption ambit, the A3LA-RG needs less power to work than the

Certus 100, it needs an input voltage of 4.0V to 5.4V in the low end and 5.0V to 32V in the high end, depending on the usage of the modem. In the idle mode, it will consume 175mA @ 5V, and at a transmitted power it will consume 420mA @ 5V. Also, it has a nominal input of 5VDC on the whole circuit. The recommended power supply is not specified in the documentation of the product but, it will be less than the one in the Quicksilver QS-100.

Like with the Quicksilver QS-100, the rugged design of the modem makes it is equipped to withstand the harsh elements that the environment presents, so it will fit the Arctic Sea project that could be presented. It can function effectively within a temperature of roughly –30°C to +70°C.

As said before and with all the Iridium services it will have the advantage of having global connectivity through all the world, so it will be a good option for future projects where LTE modems and services cannot send information or work on those extreme conditions.

For the connection with the buoy, it is used a UART connection to transmit data between the buoy and the modem all connected with the STM32 micro-controller we are using for all of our implementation. As said earlier the A3LA-RG modem has different types of connection protocols, it enables SBD, Data Switch, and RUDICS. If the data plan is the Circuit Switch Data, data is transmitted over data calls between devices. However, if we have the RUDICS data plan - what we want to use - the A3LA-RG will send the data to a server using TCP.

After knowing all the information about the A3LA-RG, we want to explore the two cases that Ocean Access brought to the table the same as the last two services.

The first case is to see if the modem could be used as an emergency backup to LTE, for example, if the LTE equipment is broken (trawler, harsh weather, unforeseen wear and tear, LTE network downtime). So in this case what is needed, it would be necessary to send out an SOS containing the location of the buoy and the battery percentage. The A3LA-RG modem would be a suitable option in this case. It is an A3LA-RM - as said before - with an internal GPS receiver, which can be useful for sending out the SOS message containing the location of the buoy and the battery percentage. The modem can be configured to an emergency mode where if something happens - one of the cases above - it can transmit through any of the data protocols (SBD, Circuit Switch Data, RUDICS) the SOS message with the location of the buoy, thanks to the built-in GPS and the battery percentage available with the STM32 micro-controller that is connected to every hardware on the buoy including the main battery. Also thanks to the rugged and small modem design it can fit in as an emergency backup, not using much power and space inside the buoy and overcoming the harsh weather or conditions of the environment the buoy can be suffering. So it can be a second option of communication behind the usual LTE services. The configuration will be through a local server where we can manage all the different states the modem can have. In this case, we will have an Idle state where it will be sleeping waiting for an emergency

and then the second state the SOS state will emerge taking control of the data transmissions of the buoy and sending all the information needed for the company.

The second case exposed by Ocean Access is that, if the buoy is placed in areas too remote for LTE or other types of communication, for example, the buoy is placed in the Arctic Sea in order to do long-term monitoring of environmental conditions. In this case, it could be anywhere from 4kB a day on the low end to 500kB a day on the high end. In both cases, it would be necessary to send data once a week so it would be 4*7kB on the low end and 500*7kB on the high end. The A3LA-RG modem can be used for remote data transmission in areas where LTE or other types of communication are not available, such as the Arctic Sea. The modem enables SBD, SMS, Data Switch, and RUDICS connectivity to the Iridium® network, which provides global coverage, making it suitable for remote locations. The Iridium RUDICS service offers an effective data rate of approximately 10 to 12Kbytes per minute. This data rate applies to both upload and download data transfers. So approximately it has a data rate of 200Kbps, which is higher than the Quicksilver QS-100. This is because the A3LA-RG can use different data protocols such as SBD, Circuit Switch Data and RUDICS, and for each case of sending data it can use the one that fits the most for that specific transmission. For example, in the order of small messages, the SBD can be useful, and for messages that are bigger in size, RUDICS can be a better option. Knowing that the data transfer rate is quicker than the Quicksilver QS-100, we can assume that it will be capable of satisfying the needs of sending 4kB or 500kB a day or sending once a week 28kB or 3.5MB. This assumption is also supported by the manufacturer - NAL Research - data, which informed us that this modem would be more than capable of handling these payloads.

CHAPTER

FOUR

RESULTS

In this section, the results from the implementation of each service will be displayed. For the services Certus 100 and RUDICS the physical implementation was impossible due to the high cost of the operation and the time frames in which the project was compressed. Also, the project was to see if the implementation of these services would be a good option for Ocean Access in order to be the main communication service or an emergency backup for the buoy. So the physical/-software implementation was a part of the project that was considered at first but ruled out once started.

4.1 SBD

In the first case, a theoretical implementation was done for the SBD where, after collecting information about satellite communications, from the Iridium satellite network, and from the modem that was considered, the two cases that Ocean Access suggested were studied. After collecting all of the important information the modem that was chosen for the implementation was considered a good choice for a future implementation in the actual project of Ocean Access. All of the information considered and the reasoning for this solution was exposed in the implementation part of the Thesis. So in the theoretical field, the Iridium 9603N modem for the SBD service was a feasible option for the project.

For the SBD a real implementation was done, Ocean Access already started implementing the software and the hardware needed for the connection to the Iridium network. The software implementation was correct though the connection with the Iridium network was inaccessible for hardware complications. The implementation of the AT& commands was carried out according to the specifications of the ISU AT commands manual, which handles the AT commands for the Iridium 9603N modem [37].

The AT commands that were used were the +SBDWT which we used to write the message to the modem buffer - Exec Command: +SBDWT=[<text message>] -, the +SBDIX which is used to specify the location of the modem and has an implementation of 6 integers to disaggregate the information of that specific remote location - Exec Command: +SBDIX[A][=<location>] -. All of these commands

are implemented as intended in the ISU AT commands manual. The implementation can be verified in the appendices code, where that code is explained in the implementation part for the SBD. For all of the other implementation parts, the communication process was done before this Thesis so there is no problem for the connection there.

4.2 Certus 100 and RUDICS

In this section, I am going to talk about Certus 100 and RUDICS like one, they are on the same page as they were not implemented in a practical way. The process of this project made the practical implementation impossible, so only the theoretical implementation was conducted. Which was about knowing and investigating if both of the services with each modem were a feasible option for Ocean Access in their projects, specifically on the communication buoy.

After all the information gathering for both of the services and the review of the two cases that Ocean Access presented, both of the services each with its specific modem were evaluated as good options to be a representative in the communication process for the buoy project. As said in the SBD result section all of the information about how was the theoretical implementation conducted is in the implementation section for both the Certus 100 with the Quicksilver QS-100 from NAL Research, and the RUDICS with the A3LA-RG from NAL Research.

CHAPTER

FIVE

DISCUSSION

In this section, a future work discussion will be presented in order to give some hints on how I would approach this project in the future or how I would carry the following Thesis out.

5.1 Future work

The present research has laid the foundation for the implementation and evaluation of Iridium satellite communication services within the context of the submersible buoy project Ocean Access is developing. While some progress was made, there exist several avenues for future exploration and enhancement, ensuring the project's continuous development and adaptability to emerging challenges and opportunities.

First of all a software implementation for the SBD was revised, but the connection with the satellite was not ensured. So, in a future project, the connection with the satellite and revision of the software implementation could be done, in order to reduce power consumption and enhance data transfer rates. Also, other functionalities of SBD could be investigated in order to ensure a better communication system. Also, to develop protocols for better data encryption, authentication, and protection to safeguard sensitive information could be done.

For both the Certus 100, and RUDICS an availability plan was made in order to know if they were suitable for the buoy project. If they manage to be the best option, the logical step would be the implementation of them into the real-world buoy. So a Thesis project could be the implementation and testing of those technologies. Conducting the feasibility test to asses the real suitability for the main project.

Another option would be, a Hybrid Service Implementation of all of three services, to create a comprehensive communication system. So the work could be to develop a communication strategy that optimizes the strengths of each service for various aspects of the project and ensure the system's robustness and flexibility in adapting to changing project requirements and environmental conditions.

Once the project of the submersible body got to the point of experimenting in real cases, a project that could be desirable is to develop data analysis and visualization tools to process and interpret the incoming data streams effectively. This could be key in the decision-making processes and could be a new product that Ocean Access could offer to their clients, so they just need to receive the information and not treat it.

In the context of potentially sensitive deployment areas, a Thesis could consider conducting an assessment of the project's environmental impact. This Thesis could assess how project operations and communication may affect local ecosystems and propose mitigation strategies if necessary.

In conclusion, this Future Work section outlines various directions for further research and development, each with the potential to enhance the submersible buoy project's capabilities and impact. These suggestions reflect the dynamic nature of the field and the need for ongoing innovation to address emerging challenges and opportunities in environmental monitoring and satellite communication. Future research in these areas will contribute to the continuous advancement of the project's objectives.

CHAPTER

SIX

CONCLUSIONS

For this Thesis, research was done on three of the Iridium services - SBD, Certus 100, and RUDICS - which gave us an insight into which of the three services could be more suitable for the submersible buoy project Ocean Access is developing.

In this research, apart from the review of the software implementation of the SBD done in the last Thesis and implementing an algorithm that updates the config file, an availability plan was made for each of the services in order to know which of them was the one that could fit the most inside the specifications from Ocean Access. After the research and the theoretical implementation of the services, all of them were found suitable to do the job Ocean Access needed. But some conclusions were made.

First of all, the Iridium 9603N SBD modem was the first and simplest modem to be researched, this modem fitted perfectly in the capabilities of the project, and also was the cheapest so to make different experiments was the first to be acquired. This modem had the difficulty that had to be implemented in order to be able to communicate the buoy's microcontroller and the Iridium satellite network. Also, the transfer rate that Ocean Access needed was superior to the one that the modem could handle - but could be a suitable option for different projects, for projects that may need more transfer rates, this modem could not be the most suitable for the project-. However, the small size and the rugged design made him a suitable option to be implemented in the real project.

Secondly, the Quicksilver QS-100 from NAL Research for the Certus 100, offered everything the project needed. The transfer rate was perfect for the communications Ocean Access was expecting in their different sample cases, and the implementation is simple as it had a local server connection granted by Iridium with a guided GUI where you could make all the possible changes to the modem without implementing any code into the codebase. The only downside of the Quicksilver is that out of the three is the biggest modem and could be a problem in order to fit inside the buoy's small capacity.

Finally, the A3LA-RG from NAL Research for the RUDICS service is the most balanced modem out of the three. It has also the simple implementation of the

Certus 100, where almost any code might be implemented. Everything should be configured throughout the local server that Iridium grants. It also can handle the payload sizes that Ocean Access needs for its usual communication processes and could solve the scalability problem for projects with needs that may exceed the common communication processes. Not as the Quicksilver, the A3LA-RG meets the conditions to be a small-sized modem that could fit with any problem inside the buoy's capacity. To conclude, the A3LA-RG, apart from the RUDICS service can offer different communication connectivity such as SBD itself, SMS, and Circuit Data Switch, which can bring different types of connectivities that may upgrade the communication system expanding the research horizons of the project.

In conclusion, all of the services were attested as suitable for the project needs. But if I had to pick one of them, after the research, for this project I would choose the A3LA-RG from NAL Research. Because it has the opportunity to expand the project needs and horizons, in order to be a more complete product that can fit more complex research and scenarios. Therefore, I suggest studying this modem further and trying to implement it physically for the final project.

REFERENCES

[1] Marcin Frąckiewicz. "*The Role of Satellite Communication in Today's World*". In: *TS2* (Mar. 2023). URL: https://ts2.space/en/the-role-of-satellite-communication-in-todays-world/.

[2] John Coykendall et al. "*Riding the exponential growth in space. Higher investment, improved infrastructure, and digital technologies could unlock potential across the space ecosystem*". In: *Deloitte Insights* (Mar. 2023). URL: https://www2.deloitte.com/za/en/insights/industry/aerospace-defense/future-of-space-economy.html.

[3] Qian Wang et al. "An Overview of Emergency Communication Networks". In: *Remote Sensing* 15.6 (2023). ISSN: 2072-4292. DOI: 10.3390/rs15061595. URL: https://www.mdpi.com/2072-4292/15/6/1595.

[4] Kateryna Sergieieva. "*Types Of Satellites: Different Orbits Real-World Uses*". In: *EOS Data Analytics* (Mar. 2023). URL: https://eos.com/blog/types-of-satellites/#ref-1.

[5] "*Communications satellite*". In: *Wikipedia* (July 2023). URL: https://en.wikipedia.org/wiki/Communications_satellite.

[6] Virgil Labrador. "*Satellite communication - Orbit, Signals, Relay | Britannica*". In: *Britannica* (May 2023). URL: https://www.britannica.com/technology/satellite-communication/How-satellites-work.

[7] European Space Agency. "*Types of orbits*". In: *EnablingSupport - ESA* (Mar. 2020). URL: https://www.esa.int/Enabling_Support/Space_Transportation/Types_of_orbits.

[8] Thomas G. Roberts. "*Popular Orbits 101*". In: *CSIS* (Nov. 2017). URL: https://aerospace.csis.org/aerospace101/earth-orbit-101/.

[9] "*Satellite Communication - Quick Guide*". In: *Learn Satellite Communication - artificial satellite* (). URL: https://www.tutorialspoint.com/satellite_communication/satellite_communication_quick_guide.html.

[10] Dick McClure. "*Overview of Satellite Communications*". In: *GMU* (). URL: https://olli.gmu.edu/docstore/800docs/0909-803-Satcom-course.pdf.

[11] "*Satellite Communication*". In: *Wordpress* (). URL: https://elearningatria.files.wordpress.com/2013/10/ece-vi-satellite-communications-10ec662-notes.pdf.

[12] Michael Komara. "*Translator for time division multiple access wireless system having selective diversity circuits*". In: *Google patents* (Dec. 1999). URL: https://patents.google.com/patent/US5970406.

[13] Cadence PCB Solutions. "*Satellite Frequency Allocation and the Band Spectrum*". In: *Cadence* (). URL: https://resources.pcb.cadence.com/blog/2023-satellite-frequency-allocation-and-the-band-spectrum.

[14] "*Satellite Frequency Bands: L, S, C, X, Ku, Ka-band – UPSC*". In: *Lotus Arise* (Dec. 2020). URL: https://lotusarise.com/satellite-frequency-bands-upsc/.

[15] "*Satellite frequency bands*". In: *The European Space Agency* (). URL: https://www.esa.int/Applications/Connectivity_and_Secure_Communications/Satellite_frequency_bands.

[16] "*NASA - 9.0 Communications*". In: *NASA* (). URL: https://www.nasa.gov/smallsat-institute/sst-soa/communications.

[17] "*How satellite communication works*". In: *Ovzon* (). URL: https://www.ovzon.com/en/how-satellite-communication-works/.

[18] "*The Pros and Cons of Ka-Band Applications*". In: *Cadence* (). URL: https://resources.pcb.cadence.com/blog/2022-the-pros-and-cons-of-ka-band-applications.

[19] "*Iridium main website*". In: *Iridium* (). URL: https://www.iridium.com/.

[20] "*Iridium network site*". In: *Iridium* (). URL: https://www.iridium.com/network/.

[21] "*Iridium satellite constellation*". In: *Wikipedia* (). URL: https://en.wikipedia.org/wiki/Iridium_satellite_constellation.

[22] Marcin Frąckiewicz. "*The Role of Iridium Satellites in Internet of Things (IoT) Connectivity*". In: *TS2-Space* (Mar. 2023). URL: https://ts2.space/en/the-role-of-iridium-satellites-in-internet-of-things-iot-connectivity/.

[23] "*Iridium Communications Network*". In: *Roadpost* (). URL: https://www.roadpost.com/iridium-satellite-network.

[24] "*Iridium-NEXT*". In: *Space Flight 101* (). URL: https://spaceflight101.com/spacecraft/iridium-next/#google_vignette.

[25] "*Iridium Short Burst Data® (SBD®)*". In: *Iridium* (). URL: https://www.iridium.com/services/iridium-sbd/.

[26] "*Short Burst Data (SBD) Services*". In: *Beam* (). URL: https://www.beamcommunications.com/services/short-burst-data-sbd.

[27] "*Iridium Burst®*". In: *Iridium* (). URL: https://www.iridium.com/services/iridium-burst/.

[28] "*Iridium Certus® 100*". In: *Iridium* (). URL: https://www.iridium.com/services/iridium-certus-100/.

[29] "*Iridium Certus 100 Equipment*". In: *Ground Control* (). URL: https://www.groundcontrol.com/us/products/iridium/iridium-certus-100-range/.

[30] "*Iridium Certus® 100*". In: *Skytrac* (). URL: https://www.skytrac.ca/iridium-certus-plans/certus-100/.

[31] Iridium Inc. "*Groundbreaking Iridium Certus® 100 Service Launches with Partner Products for Land, Sea, Air and Industrial IoT*". In: *Cision PR Newswire* (Nov. 2021). URL: https://www.prnewswire.com/news-releases/groundbreaking-iridium-certus-100-service-launches-with-partner-products-for-land-sea-air-and-industrial-iot-301420816.html.

[32] "*Iridium RUDICS*". In: *Iridium* (). URL: https://www.iridium.com/services/iridium-rudics/.

[33] "*RUDICS Solution Overview*". In: *Uplogix* (Nov. 2011). URL: https://uplogix.com/docs/pdf/RUDICS%20Summary%204.4.pdf.

[34] "*Iridium® RUDICS Service*". In: *Git Satellite Communications* (). URL: https://gitsat.com/products/Iridium_RUDICS_Service-118-79.html.

[35] "*Iridium RUDICS*". In: *Dynautics* (). URL: https://www.dynautics.com/products-unmanned-surface-vehicle/communications-systems/iridium-rudics/.

[36] "*Quicksilver QS-100*". In: *CLS Telemetry* (). URL: https://www.cls-telemetry.com/iridium-solutions/iridium-products/hardware/quicksilver-qs-100/.

[37] "*MAN0009 ISU AT Command Reference*". In: *Iridium* (). URL: https://www.groundcontrol.com/en/wp-content/uploads/2022/02/IRDM_ISU_ATCommandReferenceMAN0009_Rev2.0_ATCOMM_Oct2012.pdf.

APPENDICES

SBD Software code - C++

In this section of the appendices, I am going to show the code that was used for the implementation in the SBD case. Including the both header file and the cpp file.

iridium_sbd_drv.h

```
1
2  #ifndef INC_IRIDIUM_SBD_DRV_H_
3  #define INC_IRIDIUM_SBD_DRV_H_
4
5  #include "iDriver.h"
6  #include <stdio.h>
7  #include <cstring>
8  #include <string.h>
9
10 enum SBD_CMD{
11   DC_LOAD,
12   DC_INIT,
13   DC_INIT_LONG,
14   DC_LOAD_LONG,
15   DC_SEND
16 };
17
18 enum SBD_STATES {
19   ST_SBD_OFF,
20   ST_SBD_ON,
21   ST_SBD_READY,
22   ST_SBD_ENC_START_SEND,
23   ST_SBD_TRANSMITING,
24   ST_SBD_DELETE,
25   ST_SBD_POWER_OFF_1,
26   ST_SBD_POWER_OFF_2
27 };
28
29 enum SBD_TRANSITION {
30   SBD_MODEM_BOOTED,
31   SBD_MODEM_CONNECTED,
32   SBD_ENCODE_START_SEND,
33   SBD_CMD_OK,
34   SBD_CMD_ERROR,
```

```
35    SBD_MSG_SEND_SUCCEDED,
36    SBD_MSG_SEND_FAILED,
37    SBD_POWER_OFF_MODEM,
38    SBD_MODEM_OFF
39  };
40
41
42  class IridiumSBDDrv : public IDriver{
43  public:
44    IridiumSBDDrv();
45    ~IridiumSBDDrv();
46
47    // Functions inherited from interface
48
49    virtual char* getTestMsg();
50    virtual void send(const char *raw_data);
51    virtual char* decodeAndProcess(char *enc_data);
52    virtual void disconnect();
53    //start messaging to modem
54
55
56
57  private:
58    // Private function specific to Swarm modem
59    virtual char* getConnectedMsg();
60    virtual char* getSendMsg();
61
62    //check if we can see satelite
63    char* checkConnectivity();
64
65
66    virtual char* encode(char *raw_data);
67    virtual char* decode(char *data);
68    void load_and_send();
69
70    void sm_process(SBD_TRANSITION tr);
71    // Variables for state machine
72    SBD_STATES currentState = ST_SBD_OFF;
73    const char* currentData = nullptr;
74    uint8_t attempt = 0;
75    bool waitForBootMsg = true;
76
77    size_t max_msg_len=340U;
78    SBD_CMD encode_cmd = DC_INIT;
79
80    const char* start_loading_msg="AT+SBDWT\r\n";
81    const char* transmit_to_satelite_msg="AT+SBDIX\r\n";
82    const char* boot_modem = "AT+CIER=1,0,1\r\n";
83    const char* clearSBDmessageBuffer="AT+SBDD0\r\n";
84    const char* SBDOK="OK\r\n";
85    const char* SBDNETACQUIRED="+CIEV: 1,1\r\n";
86
87
88  };
89
90
91  #endif /* INC_IRIDIUM_SBD_DRV_H_ */
```

Listing 1: IridiumSBDDrv.h code

iridium_sbd_drv.cpp

```
#include "iridium_sbd_drv.h"
#include <string>
#include "iComChan.h"
#include <algorithm>
/*
 * Constructor
 */
IridiumSBDDrv::IridiumSBDDrv() {
  setName("IridiumSBDDrv");   // Name to precise in the configuration file
                  // to get access to this driver
}

/*
 * Destructor
 */
IridiumSBDDrv::~IridiumSBDDrv() {
  delete getName(); // Release memory allocated to save the name
}


void IridiumSBDDrv::send(const char *raw_data) {
  currentData = raw_data;
  sm_process(SBD_ENCODE_START_SEND);
}

/*
 * Function called by a digital twin using this driver
 * to encode data.
 */
char* IridiumSBDDrv::encode(char *raw_data) {
  // We will add 4 bytes for cmd, 3 bytes for the checksum,
  // a space, a line return and the end char term in the worst case

  //TODO check in \n serves as <CR><LF> in c++ as it does in python


  char* encData = new char[strlen(raw_data)+10];
  memset(encData, '\0', strlen(encData));
  if(encData) // Check if memory as been allocated
  {

    strcat(encData, "AT"); // Char before all commands
    switch(encode_cmd){
    case DC_LOAD:
      strcat(encData, "+SBDWT");  // Write to modem command
      strcat(encData, "=");   // = before data
      strcat(encData, raw_data);//this method only gives us 120 bytes of message
      strcat(encData,"\r\n");
      //If any data is currently in the mobile originated buffer, it will be overwritten
      delete raw_data; // Release memory allocated for the raw data
```

```
51        return (encData);
52        break; // Should never come here
53      case DC_INIT_LONG:
54        strcat(encData, "+SBDWT\r\n");
55        delete raw_data;
56        return (encData);
57      case DC_LOAD_LONG:
58        encData=new char[strlen(raw_data)+4];
59        strcat(encData, "\r");
60        delete raw_data;
61        return (encData);
62      case DC_SEND:
63        strcat(encData, "+SBDIX\r\n");
64        delete raw_data;
65        return (encData);
66        break;
67      case DC_INIT: //TODO implement this into ComChanAT ?
68        strcat(encData, "+CIER"); // Write to modem command
69        strcat(encData, "=");   // = before data
70        strcat(encData, "1,0,1,0\r\n");
71        delete raw_data;
72        return (encData);
73        break;
74      default:
75        // Command not implemented
76        printf("%s : cmd asked, not implemented\n", getName());
77        delete raw_data;
78        return (nullptr);
79      }
80    }
81    else
82    {
83      printf("%s : memory not allocated\n", getName());
84      delete raw_data; // Release memory allocated for the raw data
85      return nullptr;
86    }
87    if (true){ //TODO implement
88
89    }
90  }
91
92  /*
93   * Called by a digital twin using this driver to decode
94   * received data.
95   *
96   * Return nullptr if the data received are not used by
97   * our program.
98   * Return data decoded if it's pure data to transmit or
99   * to send to someone else. We can imagine, maybe later
100  * send it to the buoyancy controller.
101  */
102 char* IridiumSBDDrv::decode(char *data) {
103   //TODO fix this function to decode iridium
104   bool processed = false;
105   //std::string temp(data);
106   //char * dataResult;
107   switch(data[0]){ //TODO maybe rewrite to one huge switch?
108   case '+':
```

```
109     if (strstr(data,"+SBDIX: 0")){
110       processed=true;
111       //if +SBDIX: 0 exists in the string, we know sending the
      data was successfull
112       return data; //TODO check if this makes sense
113     }
114     else if(strstr(data,"+SBDIX:")){
115       processed=true;
116       //if it does contain +SBDIX: but not +SBDIX: 0, the sending
      failed
117       return (nullptr);
118     }
119     else{
120       processed=true;
121       return (nullptr); //default return nullptr
122     }
123
124   case 'O':
125     if (strstr(data,"OK")){
126       processed=true;
127       return (data);
128     }
129     else{
130       processed=true;
131       return (nullptr);
132     }
133   case 'R':
134     if (strstr(data,"READY")){
135       processed=true;
136       return (data);
137       break;
138     }
139
140   default:
141     return (nullptr); //in case we have no idea what is coming in,
      we just say "fuck it"
142
143   }
144   if(!processed){
145     return nullptr;
146     }
147   else{
148     printf("%s : Critical error! Should not reach this line!",
      getName());
149     return nullptr;
150     }
151
152
153 }
154
155 char* IridiumSBDDrv::decodeAndProcess(char *enc_data){
156   return ((char*) IridiumSBDDrv::decode(enc_data));
157 }
158 void IridiumSBDDrv::disconnect(){
159   sm_process(SBD_POWER_OFF_MODEM);
160 }
161
162
```

```
163 char* IridiumSBDDrv::getSendMsg(){
164   return ("AT+SBDIX");
165 }
166
167
168 /*
169  * Return a test message
170  * To test if there is somebody (a modem) connected at the
171  * other side of the connection.
172  */
173 char* IridiumSBDDrv::getTestMsg() {
174   //TODO delete
175   return "$CS*10\n";
176 }
177
178
179 /*
180  * modem.
181  * It's mean that the modem acquire a connection.
182  * In this case
183  * Send AT
184  * Get OK
185  *
186  */
187 char* IridiumSBDDrv::getConnectedMsg() {
188
189
190
191
192 }
193
194
195 /*
196  * State machine of the SWARM driver
197  */
198 void IridiumSBDDrv::sm_process(SBD_TRANSITION tr) {
199   SBD_STATES oldState = currentState;
200   printf("in the SBD drv SM \n ");
201   // Exit and while instruction
202   switch(currentState)
203   {
204   case ST_SBD_OFF:
205     if(tr == SBD_MODEM_BOOTED)
206       currentState = ST_SBD_ON;
207     if(tr == SBD_POWER_OFF_MODEM)
208       // Update state of the digital twin
209       getDigitalTwin()->connectionClosed();
210     break;
211   case ST_SBD_ON:
212     if(tr == SBD_MODEM_CONNECTED)
213       currentState = ST_SBD_READY;
214     else if(tr == SBD_POWER_OFF_MODEM)
215       currentState = ST_SBD_POWER_OFF_1;
216     break;
217   case ST_SBD_READY:
218     if(tr == SBD_ENCODE_START_SEND)
219       currentState = ST_SBD_ENC_START_SEND;
220     else if(tr == SBD_POWER_OFF_MODEM)
```

```
      currentState = ST_SBD_POWER_OFF_1;
    break;
  case ST_SBD_ENC_START_SEND:
    if(tr == SBD_CMD_OK)
      currentState = ST_SBD_TRANSMITING;
    else if(tr == SBD_CMD_ERROR)
    {
      // Re-encode data (try again)
      //const char * enc_data = encode(currentData);
      // Call Send callback function of the digital twin

      load_and_send();
    }
    break;
  case ST_SBD_TRANSMITING:
    if(tr == SBD_MSG_SEND_SUCCEDED)
    {
      // Notify the digital of the success through his callback
   function
      getDigitalTwin()->sendSucceded();
      currentState = ST_SBD_READY;
    }
    else if(tr == SBD_MSG_SEND_FAILED)
    {
      if(attempt >= 5)
      {
        currentState = ST_SBD_DELETE;
      }
      else
      {
        // Need some improvement here
        //        osDelay(2000);
        //        // Ask modem if remain unsent messages
        //        callbckFctSend(msgAskCountRemainUnsent);
        //        attempt++;
      }
    }
    break;
  case ST_SBD_DELETE:
    if(tr == SBD_CMD_OK)
      currentState = ST_SBD_READY;
    else if(tr == SBD_CMD_ERROR)
      // Send delete all unsent message cmd
      getDigitalTwin()->send(clearSBDmessageBuffer, true);
    break;
  case ST_SBD_POWER_OFF_1:
    if(tr == SBD_CMD_OK)
      currentState = ST_SBD_POWER_OFF_2;
    else if(tr == SBD_CMD_ERROR)
      // Send power off cmd
      getDigitalTwin()->send(nullptr, true);
    break;
  case ST_SBD_POWER_OFF_2:
    if(tr == SBD_MODEM_OFF)
    {
      // Update driver state
      currentState = ST_SBD_OFF;
      // Update digital twin state
```

```
278         getDigitalTwin()->connectionClosed();
279       }
280       break;
281     }
282
283     if(oldState != currentState)
284     {
285       // Entry instruction
286       switch(currentState)
287       {
288       case ST_SBD_OFF:
289         /*
290          * Don't care about received command except
291          * $M138 BOOT,RUNNING*2a <- Which means, modem is ON
292          */
293         waitForBootMsg = true;
294         break;
295       case ST_SBD_ON:
296         // Now we can care about all received commands
297         waitForBootMsg = false;
298         break;
299       case ST_SBD_READY:
300         attempt = 0;
301         // Call Ready callback function of the digital twin
302         getDigitalTwin()->setState(COMCHAN_READY);
303         printf("%s : ready to transmit\n", getName());
304         break;
305       /*case ST_SBD_ENC_START_SEND:
306         // Call Send callback function of the digital twin
307         getDigitalTwin()->send(encode(currentData), true);
308         break; */
309       case ST_SBD_TRANSMITING:
310         // Need some improvement here
311         //      osDelay(2000);
312         //      // Ask if the msg has been sent
313         //      callbckFctSend(msgAskCountRemainUnsent);
314         //      attempt++;
315         load_and_send();
316         break;
317       case ST_SBD_DELETE:
318         // Send delete all unsent message cmd
319         getDigitalTwin()->send(clearSBDmessageBuffer, true);
320         // Notify the digital twin that the send failed
321         getDigitalTwin()->sendFailed();
322         break;
323       case ST_SBD_POWER_OFF_1:
324         // Send power off cmd
325         getDigitalTwin()->send(nullptr, true);
326         break;
327       case ST_SBD_POWER_OFF_2:
328         // Nothing to do on entry here
329         break;
330       }
331     }
332 }
333
334 void IridiumSBDDrv::load_and_send(){
335   //verbose feedback
```

```
336   printf("we are in the load and send function");
337   printf("the message is %s", currentData);
338   //check for msg len
339   size_t msg_len = sizeof(*currentData)/sizeof(currentData[0]);
340   // calculate the number of packets needed
341   size_t num_msg = (msg_len + max_msg_len - 1) / max_msg_len;
342
343     // send each packet
344   char buf[max_msg_len];
345   for (size_t i = 0; i < num_msg; i++) {
346     size_t start_index = i * max_msg_len; //find the index in msg
      we start at
347     size_t end_index = std::min(start_index + max_msg_len, msg_len
      ); //find the index in msg that we end at
348
349       //tell the SBD modem that we are about to load it with data
350     getDigitalTwin()->send(start_loading_msg,true);
351     // TODO probably should get some form of feedback from iridium
      modem
352     char* response = getDigitalTwin()->receive();
353     printf("%s", response);
354     if (response != SBDOK or response != SBDNETACQUIRED){
355       printf("the iridium modem refuses to accept the command");
356     }
357
358     //send the payload to the SBD modem
359     std::copy(currentData+start_index,currentData+end_index, buf);
360     getDigitalTwin()->send(buf,true);
361     // TODO probably should get some form of feedback from iridium
      modem
362     response = getDigitalTwin()->receive();
363     printf("%s", response);
364     if (response != SBDOK or response != SBDNETACQUIRED){
365       printf("the iridium modem refuses to accept the command");
366     }
367     std::memset(buf,0,sizeof(buf)); //empties the buffer
368     //tell the SBD modem to send the payload to the satelite
     network
369     getDigitalTwin()->send(transmit_to_satelite_msg,true);
370     // TODO probably should get some form of feedback from iridium
      modem
371     response = getDigitalTwin()->receive();
372     if (response != SBDOK or response != SBDNETACQUIRED){
373       printf("the iridium modem refuses to accept the command");
374     }
375   }
376
377 }
```

Listing 2: IridiumSBDDrv.cpp code