Cape Peninsula
University of Technology

AN INVESTIGATION OF AN ALGORITHM FOR ERROR DETECTION AND CORRECTION IN TRANSCEIVERS FOR NANOSATELLITES

by

THEMBA SIVATE

Thesis submitted in partial fulfilment of the requirements for the Degree

Master of Engineering: Satellite Systems & Applications

in the Faculty of Engineering and Built Environment

at the Cape Peninsula University of Technology

Supervisor: Prof B Vipin

Bellville, Cape Town

Date Submitted: November 2024

CPUT copyright information
The thesis may not be published either in part (in scholarly, scientific or technical journals), or as a whole (as a monograph), unless permission has been obtained from the University

DECLARATION

I, Themba Mthembane Sivate, declare that the contents of this thesis represent my own unaided work, and that the thesis has not previously been submitted for academic examination towards any qualification. Furthermore, it represents my own opinions and not necessarily those of the Cape Peninsula University of Technology.

Signed: TM Sivate **Date:** 06-11-2024

ABSTRACT

One of the biggest problems faced by many organizations in the space science industry is the inability to determine the accuracy of the data received from satellites. A satellite collects data from a point of interest, store the data on the onboard storage, then transmits it to the ground when the ground station is visible. The problem arises when the satellite loose connection to the ground station while it was transmitting data, resulting in corrupt and unusable data. The purpose of this study is to identify a possible solution to this problem by investigating and developing an algorithm that can be used to determine the accuracy of the data received from the satellite and provide an 8-bit error correction. This study focuses on Very High Frequency (VHF) and Ultra High Frequency (UHF) Transceivers which are used for light weigh data transfer between a nanosatellite and a ground station. Since the transceiver normally transfers data at a rate of 9600bps (depending on the application), we can easily simulate such data transfer using a universal asynchronous receiver-transmitter (UART) based transceiver such as nRF24L01+ with a baud rate of 9600bps. In the nanosatellite industry, little to no research about error detection and correction, especially for VHF/UHF Transceivers.

The proposed solution uses an ASCII table as a base for signing and validating data exchanged wirelessly between the nanosatellites and the ground stations. The error detection is archived by computing both the number of characters and the sum of the characters as the corresponding decimal value of each character is found in the ASCII table. Both variables are then set as a header separated by a pipe (vertical bar) into the actual payload sent by satellite to the ground station. Once the data arrives at the ground station, we recompute to check the payload's accuracy and the possibility of data recovery if any was lost. Data recovery is archived by computing the difference between the sum from the header and the sum computed on arrival, and by comparing the number of characters from both ends. When 8 bits (single character) of data is lost, such data is recovered from the ASCII table using the difference computed. This method was chosen because any programming language understands the ASCII standard and can be implemented at both application and firmware level, which is crucial for bare metal implementation as it supports power efficiency.

The experimental results show that the proposed algorithm always detects invalid payloads. Furthermore, the experimental results show that the data recovery rate is 100% when the data loss is a single byte (8 bits). An increased payload size was also noticed due to the appended header, which then increased the time it takes for data to be transferred from the satellite to the ground station. A response command was also sent indicating the data integrity status of the payload received, allowing the satellite to delete such payload as the onboard storage is limited.

ACKNOWLEDGEMENTS

I wish to thank my supervisor Prof. B Vipin of Cape Peninsula University of Technology for the support and guidance during the years of this research. Without his guidance and support, I could not have completed this thesis. The financial assistance of the National Research Fund towards this research is acknowledged. Opinions expressed in this thesis and the conclusions arrived at, are those of the author, and are not necessarily to be attributed to the National Research Foundation.

TABLE OF CONTENTS

Declaration ii
Abstract iii
Acknowledgements iv
Abbreviations and acronyms ix
Glossary x

CHAPTER 1: BACKGROUND TO THE RESEARCH PROBLEM

1.1 Introduction 1
1.2 Background of the study 1
1.3 Research questions 2
1.4 Problem statement 2
1.5 Proposed methodology 2
1.6 Objectives of the study 4
1.6.1 General objective 4
1.6.2 Specific objective 4
1.7 Significance of the study 5
1.8 Scope of the study 5
1.9 Delimitation of study 5
1.10 Structure of Thesis 5

CHAPTER 2: LITERATURE REVIEW

2.1 Introduction 7
2.2 General overview 7
2.2.1 Vessel to satellite data transfer 7
2.2.2 Satellite to ground station data transfer 9
2.2.3 Ground station to satellite data transfer 11
2.2.4 Sample AIVDM sentence 12
2.3 Related study 13
2.4 Limitations 20

CHAPTER 3: ANALYSIS AND DESIGN

3.1 Introduction 21
3.2 Analysis 21
3.2.1 Payload analysis 22
3.2.2 Algorithm Analysis 23
3.2.3 Simulation hardware analysis 24
3.2.4 Simulation software analysis 25
3.3 Design 27
3.3.1 Algorithm design 27
3.3.2 Hardware design 29
3.3.3 Software design 30
3.4 Summary 30

CHAPTER 4: IMPLEMENTATION AND SOFTWARE OUTPUT

4.1	Introduction	31
4.2	Tools and materials	31
4.3	Hardware connectivity	31
4.4	Driver installation and COM port	32
4.5	Algorithm development (Satellite/Sender)	33
4.6	Algorithm development (Ground Station/Receiver)	37
4.7	Application development (Sender and Receiver)	41
4.8	Output of the sending software	41
4.9	Output of the receiving software	44
4.10	Summary	44

CHAPTER 5: TESTING AND RESULTS

5.1	Introduction	45
5.2	Software and hardware testing	45
5.3	Algorithm testing	49
5.4	Results	53
5.5	Summary	53

CHAPTER 6: CONCLUSION AND FUTURE STUDIES

6.1	Conclusion	54
6.2	Future studies	54

REFERENCES **55**

APPENDIX A **59**
SENDER SOFTWARE SOURCE CODE 59

APPENDIX B **67**
RECEIVER SOFTWARE SOURCE CODE 67

APPENDIX C **72**
GROUND STATION SAMPLE LOG FILE 72

LIST OF FIGURES

Figure 1.1: ZACUBE-2 nanosatellite 1
Figure 1.2: Plan for sending data from satellite to ground station 3
Figure 1.3: Plan for receiving data with algorithm for integrity validation 4
Figure 2.1: Vessel to satellite data transfer 8
Figure 2.2: Cubesat SDR-based AIS Receiver 9
Figure 2.3: Satellite orbital period at 550km altitude 10
Figure 2.4: Satellite to ground station data transfer 11
Figure 2.5: CubeSat VHF/UHF Transceiver 11
Figure 2.6: Payload data decoded 12
Figure 3.1: satellite and ground station communication 21
Figure 3.2: 100 AIVDM messages 22
Figure 3.3: Single AIVDM message 23
Figure 3.4: Steps of data transfer between satellite and ground station 24
Figure 3.5: nRF24L01+ 2.4 GHz Transceiver module 25
Figure 3.6: USB Adapter for NRF24L01+ 25
Figure 3.7: Ordered firmware functionalities for satellite 26
Figure 3.8: Ordered software functionalities for ground station 27
Figure 3.9: ASCII table 28
Figure 3.10: Algorithm calculations 29
Figure 3.11: Hardware connectivity diagram 29
Figure 4.1: nRF24L01+ connected a USB adapter 32
Figure 4.2: Serial driver installation setup 32
Figure 4.3: Test hardware device detected 33
Figure 4.4: The execution flow for transmitting data from a satellite to a ground station 34
Figure 4.5: Algorithm code for payload packaging (satellite) 35
Figure 4.6: Algorithm code for response processing (satellite) 36
Figure 4.7: The procedural sequence for ground station data processing and response generation 38
Figure 4.8: Algorithm code for payload processing (ground station) 40
Figure 4.9: Development studio and source code 41
Figure 4.10: Sender software output (DataProcessor.exe) 42
Figure 4.11: Sender software – settings tab 43
Figure 4.12: Sender software – logs tab 43
Figure 4.13: Receiver software output (DataReceiver.exe) 44
Figure 5.1: Sending device is detected and COM is allocated 45
Figure 5.2: Sender software selects sending device 45
Figure 5.3: Sender software connectivity test 46
Figure 5.4: Data loaded and prepared for sending 47
Figure 5.5: Receiving device is detected and COM is allocated 48

Figure 5.6: Receiver software selects receiving device 48
Figure 5.7: Receiver software connectivity test 49
Figure 5.8: Simulation test passed 49
Figure 5.9: Received response flag P 50
Figure 5.10: Lost character reconstructed 50
Figure 5.11: Errors detected on payload and cannot be reconstructed 51
Figure 5.12: Multiple attempts on erroneous payload 52

LIST OF TABLES

Table 2.1: Literature Survey 15
Table 5.1: Results 53

ABBREVIATIONS AND ACRONYMS

Abbreviation/Acronym

AIS	Automatic Identification System
ASCII	American Standard Code for Information Interchange
COM	Component Object Model
FIFO	First In First Out
LED	Light Emitting Diode
LEO	Low Earth orbit
MDA	Maritime Domain Awareness
OS	Operating System
RF	Radio frequency
SDR	Software defined radio
UART	Universal Asynchronous Receiver-Transmitter
UHF	Ultra High Frequency
USB	Universal Serial Bus
VHF	Very High Frequency

GLOSSARY

Term	Explanation
AIS Receiver	The AIS Receiver is a compact Software Defined Radio (SDR) designed for CubeSat missions and capable of receiving AIS messages.
AIS Transponder	A device designed to be capable of providing position, identification, and other information about the ship to other ships and to coastal authorities automatically (AIS transponders, 2022).
AIVDM message	An encoded ASCII string containing information about a ship intended to be decoded by AIS Decoder.
Algorithm	Any well-defined procedure for solving a given class of problems (Dembski, 2022).
Computing	The use of computer operations to generate output from given input(s)
Embarcadero Rad Studio	A software development tool owned by Embarcadero Technologies
Firmware	Firmware is a type of software that is etched directly into a piece of hardware. It operates without going through APIs, the operating system, or device drivers—providing the needed instructions and guidance for the device to communicate with other devices or perform a set of basic tasks and functions as intended (Kuntal, 2022).
nRF24L01+ Transceiver	A radio frequency transceiver operating at a frequency band of 2.4GHz
nRF24L01+ USB Adapter	A USB adapter allowing the connection of a nRF24L01+ Transceiver to be used as a serial communication device
Payload	Data packaged and sent from one device to another over a network.
Program	A series of coded software instructions to control the operation of a computer or other machine (Computer Program | Encyclopedia.com, 2022).
Receiver software	A simulation software built and used to represent a ground station.
Receiving device	An electronic device responsible for receiving and processing data sent by another device.

<table>
<tr><td>Receiving software</td><td>A simulation software built and used to represent a ground station.</td></tr>
<tr><td>Sender software</td><td>A simulation software built and used to represent a satellite.</td></tr>
<tr><td>Sending device</td><td>An electronic device responsible for packaging and signing and sending data to another device.</td></tr>
<tr><td>Sending software</td><td>A simulation software built and used to represent a satellite.</td></tr>
<tr><td>Signature</td><td>A combination of characters appended into a payload for data integrity verification</td></tr>
<tr><td>Signed payload</td><td>A payload with both the sum and the length appended as a header separated by pipe</td></tr>
<tr><td>Software</td><td>computer programs that are designed by a computer programmer or, more likely, a team of computer programmers, to perform a particular function (Smith, 2022).</td></tr>
<tr><td>Transceiver</td><td>A device that can both transmit and receive signals on a communication medium (transceiver | Encyclopedia.com, 2022).</td></tr>
<tr><td>Transponder</td><td>a device for receiving a radio signal and automatically transmitting a different signal (transponder | Encyclopedia.com, 2022).</td></tr>
<tr><td>Windows</td><td>An operating system owned by Microsoft</td></tr>
<tr><td>Wireless</td><td>Wireless technology provides the ability to communicate between two or more entities over distances without the use of wires or cables of any sort (Wireless Technology | Encyclopedia.com, 2022).</td></tr>
</table>

CHAPTER 1
BACKGROUND TO THE RESEARCH PROBLEM

1.1 Introduction

The chapter starts with the background of the study on nanosatellites and data transmission. In subsequent sections, the problem statement, objective of the study, specific objectives, research question, justification of the study, significance of the study, and outline of the thesis are discussed.

1.2 Background of the study

Nanosatellites are a special class of satellites with a mass ranging from 1kg to 10kg (A Basic Guide to Nanosatellites, 2008). Nanosatellites were introduced because they are both portable and inexpensive to develop compared to conventional satellites. However, the absence of sufficiently small or inexpensive launch vehicles for the delivery of nanosatellites to orbit presents a significant barrier to the development of small satellite missions given their typically smaller budgets and development timescales (Smith et al., 2014). Nanosatellites are typically launched and deployed into space from a standardized container or canister, most frequently hitching a ride into space as a secondary or tertiary payload along with one or more larger spacecraft (Millan et al., 2019).

Nanosatellites are useful for Maritime Domain Awareness (MDA). The maritime domain is defined as all areas and things of, on, under, relating to, adjacent to, or bordering on a sea, ocean, or other navigable waterway, including all maritime-related activities, infrastructure, people, cargo, and vessels and other conveyances (The National Strategy for Maritime Security, 2005). When nanosatellites are used for MDA, they are monitoring vessels that are equipped with the Automatic Identification System (AIS) which is vital for vessel navigation, vessel tracking, vessel monitoring, searching, and rescuing missions. Figure 1.1 shows the ZACUBE-2 nanosatellite which is also demonstrating the MDA use case.

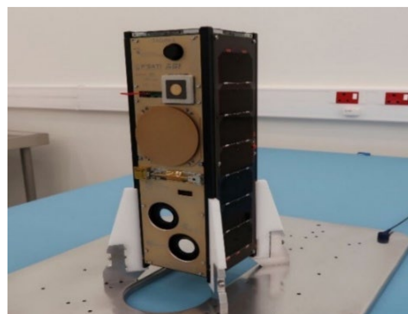

Figure 1.1: ZACube-2 nanosatellite

(SA’s ZACube-2 nanosatellite transmits first images from space - defenceWeb, 2022)

Because nanosatellites are small by design, they tend to have small and limited resources such as batteries, solar panels, onboard computers, and more. The satellite uses a Software Defined Radio (SDR) called AIS Receiver to detect vessels in the ocean. The AIS Receiver can detect any vessel equipped with an AIS Transponder. Once detected by the satellite, such data is stored on the onboard storage for later transmission to the ground station when the connection is established (when the ground station is visible).

1.3 Research questions

How can we use ASCII to detect errors in data transferred from the satellite to the ground station?

What are the effects of implementing error detection and correction mechanisms in the space industry?

Is it possible to simulate data transmission using the same baud rate as the spacecraft without using the spacecraft?

1.4 Problem statement

As the satellite may not always be in the line of sight of the ground station, it becomes crucial for it to store important data onboard until it can be successfully transmitted to the ground station. However, this can be a challenging task as the satellite's storage capacity is limited. The issue at hand is to determine whether the data received by the ground station from the satellite is complete and accurate before instructing the satellite to delete such a payload. This problem arises when a satellite loses its connection to the ground station while transmitting the data, leading to the corruption and unreliability of the data. Space radiations also cause single bit errors (Hillier et al., 2019)

1.5 Proposed methodology

The proposed methodology for this project is engineering problem solving which consists of the following phases.

- Define the problem
- List possible solutions
- Develop a plan for the most attractive solution
- Implement the solution
- Check the results

Define the problem: In the process of transferring data between a satellite and a ground station, communication can sometimes be interrupted, leading to corrupted data on one end. To ensure that the data received by the ground station is both accurate and complete, they need to be notified of any issues. Similarly, the satellite also needs to be informed of the successful reception of data by the ground station so that it can free up storage space for new data acquisition.

List possible solutions: During the study, a potential solution was proposed for evaluation. The solution involves exploring and implementing an algorithm that utilizes ASCII for signing and verifying each payload transmitted by a satellite to the ground station. The proposed solution should have the capability of performing 8-bit error correction on a corrupt payload.

Develop a plan for the most attractive solution: To develop a full plan for the most attractive solution, some learning in outcome needed to be completed first. The following modules were completed before the solution was developed, satellite mission analysis and design, satellite applications, management of space technology satellite subsystems, and research methodology. The plan takes advantage that every character set is available in the ASCII table (excluding foreign characters), therefore, this is used as a basis of data integrity validation. The plan involves developing an algorithm for sending data, and an algorithm for receiving data. Figure 1.2 shows the flow of the data from the satellite to the ground station with the planned algorithm.

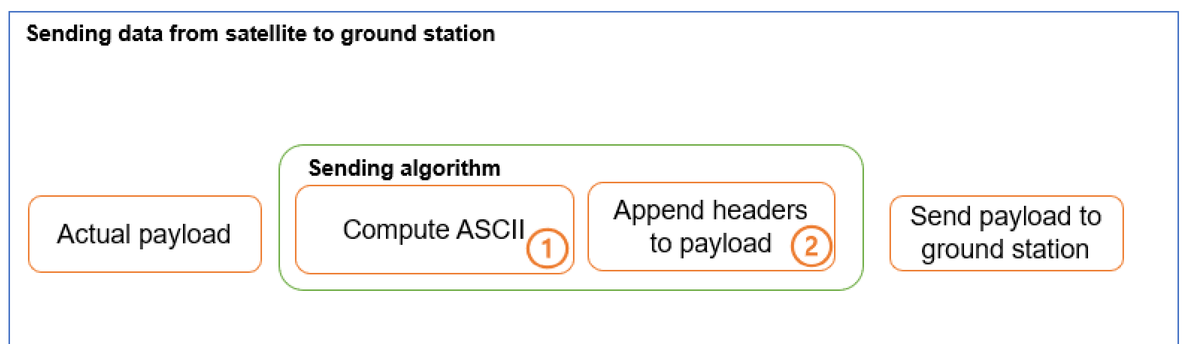

Figure 1.2: Plan for sending data from satellite to ground station

Once the data is received by the ground station, it needs to be validated using the receiving algorithm. This is done by utilizing the headers appended to the payload together with the actual payload, then recomputing the payload with the help of ASCII to determine data integrity. Figure 1.3 illustrates this concept.

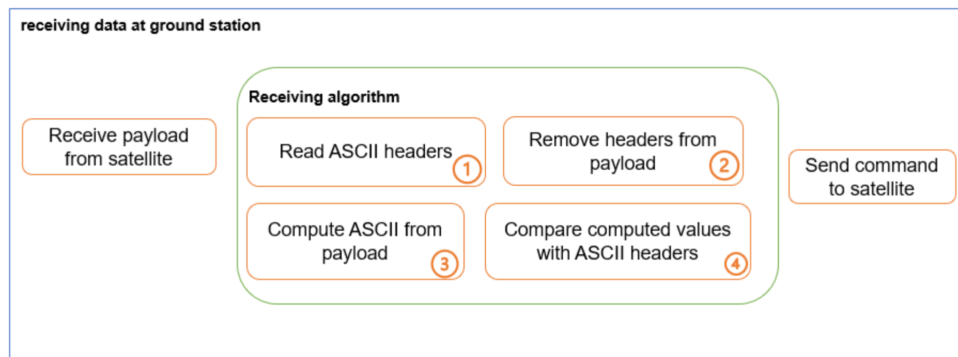

Figure 1.3: Plan for receiving data with algorithm for integrity validation

Implement the solution: This is the stage where the solution is implemented to solve the problem identified in chapter 4.

Check for the results: This is the stage where the solution is analyzed to determine if it solves the problem identified in Chapter 5.

1.6 Objectives of the study

1.6.1 General objective

The main objective of this research is to investigate and develop an algorithm for both error detection and error correction in payloads sent by satellites to the ground station.

1.6.2 Specific objective

- Implement a software that communicates with a transceiver for sending a payload from the satellite to the ground station, with an algorithm capable of appending headers to such payload for data integrity check.
- Implement a software that communicates with a transceiver for receiving a payload from the satellite, with an algorithm capable of validating such payload for data integrity and checking for data recovery possibilities.
- Connect each software with a physical transceiver to begin data transfer.
- Transfer data between both transceivers to perform a data integrity check.
- Perform a simulation for an incomplete payload transfer.
- Establish how useful this algorithm would be to the wireless communication space.

1.7 Significance of the study

This study has the following significance:

- Shows the use of ASCII to sign and validate data transmitted between devices.
- Improve error detection in data transmitted by satellite to the ground stations.
- Provide 8-bit error correction in data transmitted by satellite to the ground stations.
- Contribute to development in the space industry.

1.8 Scope of the study

To enhance data transfer between transceivers, a software will be developed using a flexible algorithm for data integrity validation and error correction. The software will be created using C++ programming language in Embarcadero RAD Studio. The nRF24L01+ transceiver will be used to simulate data transfer between satellite and ground station. Data will be transferred between both transceivers using custom-built software. This study explores error detection, data loss, and data reconstruction simulations.

1.9 Delimitation of study

For the purpose of this study, the developed algorithm will not be demonstrated using a physical real nanosatellite. Additionally, there will be no use of any VHF/UHF Transceiver to demonstrate the algorithm. Moreover, auto-connect after a connection was lost between two devices will not be demonstrated.

1.10 Structure of Thesis

Chapter 1 offers a general description of nanosatellites and their applications. It also presents ZACUBE-2, a nanosatellite employed for Marine Domain Awareness purposes. The chapter outlines the research objectives and provides a brief overview of the proposed solution.

Chapter 2 discusses related studies to this research. This chapter also provides a general overview of nanosatellites and transceivers.

Chapter 3 presents an analysis and design of the proposed solution. The chapter also covers AIVDM messages.

Chapter 4 details the implementation of the proposed solution and showcases the output. The chapter provides a comprehensive discussion of the implementation.

Chapter 5 presents the results of testing the developed solution, and the research concludes with future studies in the final chapter.

Chapter 6 concludes the research and highlights future studies and recommendations.

CHAPTER 2
LITERATURE REVIEW

2.1 Introduction

This chapter provides an overview of the process of data acquisition by the satellite from the ships, as well as the transmission of data from the satellite to the ground station. Additionally, it explains the method of data transfer between two transceivers. The chapter concludes with related studies in the same field.

2.2 General overview

2.2.1 Vessel to satellite data transfer

When a nanosatellite is deployed to an orbit, it is deployed to perform a specific task. In the case of a Maritime Domain Awareness (MDA) mission, the satellite is deployed to monitor an area of interest for vessels, which is a certain portion of an ocean or sea. A ship equipped with a functional AIS Transponder will emit an AIVDM sentence which is the crucial part of the payload. In software development and Application Programming Interfaces (API), payload refers to the data that is sent in a request or received in a response. It can also be defined as data being packaged and transmitted between devices. AIVDM is a sentence code, not an acronym. The AIVDM sentence is an encoded array of characters that defines the attributes of the vessel. In this context, attributes refer to the latitude, longitude, speed, heading, and other basic information that describes the type, location, and direction of the ship. The average size of this AIVDM sentence can range from 47 to 52 Bytes. AIVDM payload is referred to as the AIVDM sentence combined with the headers before being transmitted to the ground station. This is because the data packaged and sent to the ground station only consists of AIVDM sentences and calculated headers.

While the satellite orbits, it will eventually establish a connection with the area of interest. This is when the communication devices onboard are turned on and ready for data acquisition. In this case, the area of interest is the ocean where the ships are located. The ships are always transmitting, but the satellites are not always listening. This is because the satellite needs to save battery power, and only turns on the onboard communication devices when necessary (when close to the area of interest). Once the communication link is established, the satellite will receive a lot of messages (sentences) from the ship but will only filter and save the messages of interest (AIVDM sentences). It is important to avoid saving duplicate messages due to the limited onboard storage. The AIS Transponder can transmit the same AIVDM sentence continuously, especially if the ship is stationary. The satellite must save the data on

board in plain format unless such data is sensitive and encryption is implemented. By saving in plain format, you are saving the processing power of the satellite. Figure 2.1 shows the steps on how the data is acquired by the satellite. I composed the image shown in Figure 2.1 as part of the presentation for one of the modules of this course hence no source was supplied.

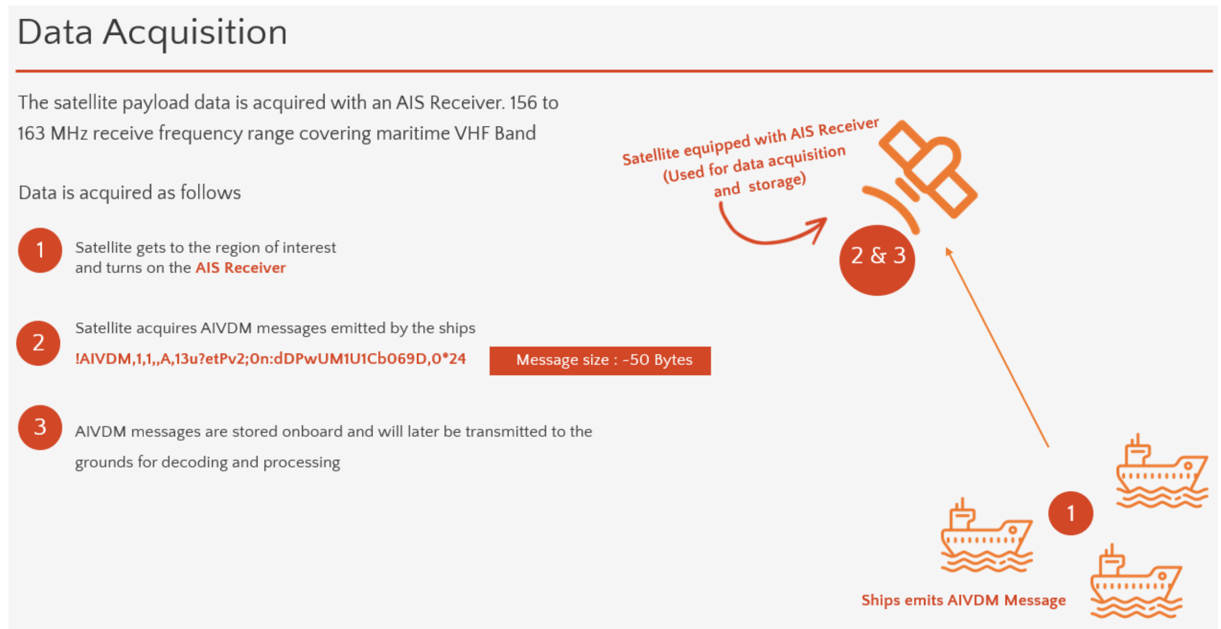

Figure 2.1: Vessel to satellite data transfer

Once the data has arrived at the satellite, it gets stored in the onboard storage. The communication window between the satellite and the ship depended on multiple variables such as the size of the area of interest, orbital altitude, inclination angle, weather conditions, and positioning of the antennas to name a few. Most satellites orbiting the Earth do so at altitudes between 160 and 2,000 kilometres which is known as Low Earth Orbit (LEO) due to the satellites' relative closeness to the Earth. Satellites in LEO typically take between 90 minutes and 2 hours to complete one full orbit around the Earth and are ideally situated for remote sensing missions, including Earth observation and reconnaissance due to its low altitudes in combination with short orbital periods.

Figure 2.2 shows the actual Software Defined Radio (SDR)-based AIS Receiver used in actual satellites to detect ships in the ocean or sea. Such a receiver has a 156 to 163 MHz receive frequency range covering the maritime VHF Band. It also has multiple interfaces including I2C bus which is used for telemetry, commands, and user data transfer. Other interfaces include UART, SPI bus, and 100baseT ethernet. This AIS Receiver is designed modular, portable, and plug-and-play, which is why is mostly used in nanosatellites. The SDR-based AIS Receiver specification was published by CPUT

in collaboration with FSATI and has a dual-core ARM9 processor that operates at 600MHz. It also has a mass of approximately 130 grams and supports input voltage ranging from 3.7VDC to 12VDC. This transceiver was demonstrated on the ZACube-2 satellite mission for a ship tracking.

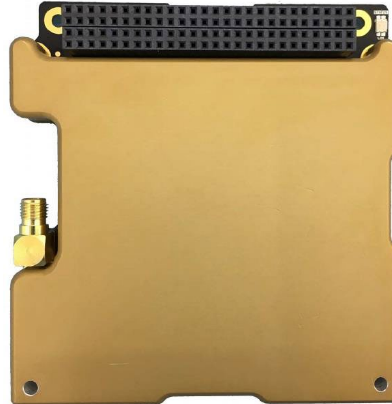

Figure 2.2: Cubesat SDR-based AIS Receiver (BR-01-00150 Rev A, CPUT, 2022)

2.2.2 **Satellite to ground station data transfer**

For any satellite to send data to the ground station must first establish a communication link. Since nanosatellites are mostly in low Earth orbit (LEO), the satellite will not always see the ground station. Even if it does, it will only see it for a short period of time (a few minutes). The visibility period depends on the inclination angle of the satellite, the altitude of the satellite, the location of the ground station, and other parameters. Figure 2.3 below shows both the velocity and the orbital period (~96 minutes) of the satellite based on the altitude the satellite is orbiting on. This is important as it indicates that while the satellite is orbiting, the ground station may be visible for a few minutes or be invisible. This implies that while there is no communication between the satellite and the ground station, the data acquired by the satellite needs to remain on the onboard storage until such data is successfully transferred to the ground station. In this context, when the ground station is invisible, it implies that there is no communication happening between the communication device at the ground station and the satellite. The orbital location of the LEO is not the only reason for the variable visibility period, but the ground station positioning of antennas also plays a role in the communication window. Figure 2.3 also shows formulas for both the velocity and the orbital period. The visibility period can be simulated using a space mission simulation software called Timeloop VTS as we have done one of the modules of this course. This paper will mostly focus on error detection and correction during data transfer.

It is important to ensure that data is delivered from the ground station with no room for error. Data integrity verification needs to happen, that is, error detection and error correction where possible. Noting that the satellite has limited storage capacity, the ground station needs to issue a command back to the satellite informing it to erase the data that was sent successfully to the ground station. This command allows the satellite to free the storage so that it can store new data that is acquired from the vessels. The erase commands can be imagined as sending a letter 'e' from the ground station to the satellite immediately after the data is received, but it can also be any acknowledgment code pre-defined on the firmware or software of the spacecraft allowing it to flag data for deletion.

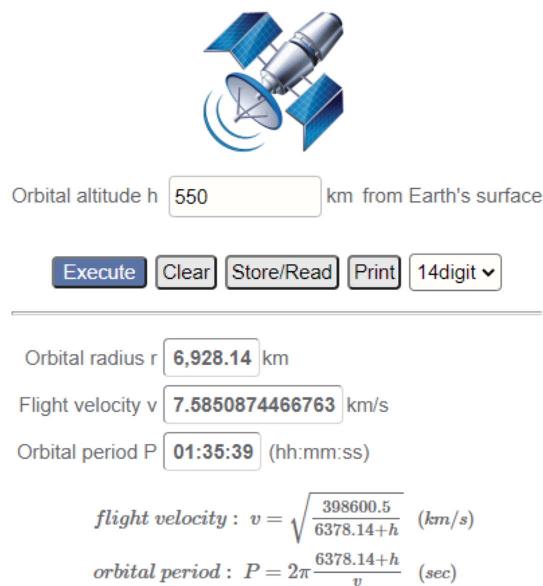

Figure 2.3: Satellite orbital period at 550km altitude
(Orbit of a satellite Calculator, 2022)

Figure 2.4 illustrates the basic steps of data transmission to the ground station and data processing. The ground station needs to be visible to the satellite. Then a communication link is established. Data is then transferred from the satellite to the ground station. The ground station receives the data and decodes it before storing it in the database. In this case, the data is transferred from the satellite at a baud rate of 9600bps. For simplicity, this figure does not show the ground station sending back the erase command to the satellite. Also, the baud rate used is the same as the one used for the simulation and testing of the algorithm proposed later in this study.

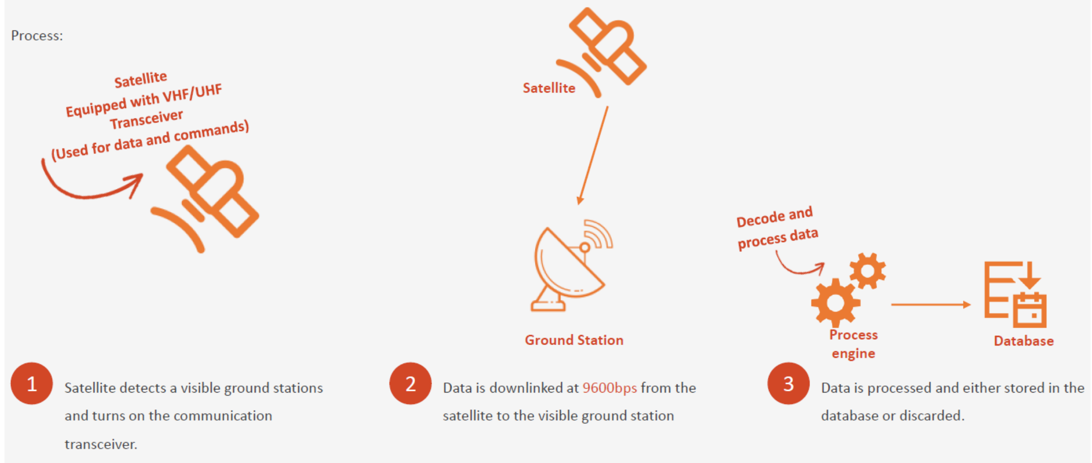

Figure 2.4: Satellite to ground station data transfer

Figure 2.5 shows the actual transceiver used by a nanosatellite to send data to the ground station, and to receive data from the ground station. This transceiver uses VHF for uplink and UHF for downlink. The receive frequencies are from 140 to 150 Mhz while the transmit frequencies are from 400 to 420 MHz and 430 to 440 MHz covering commercial and amateur bands**.**

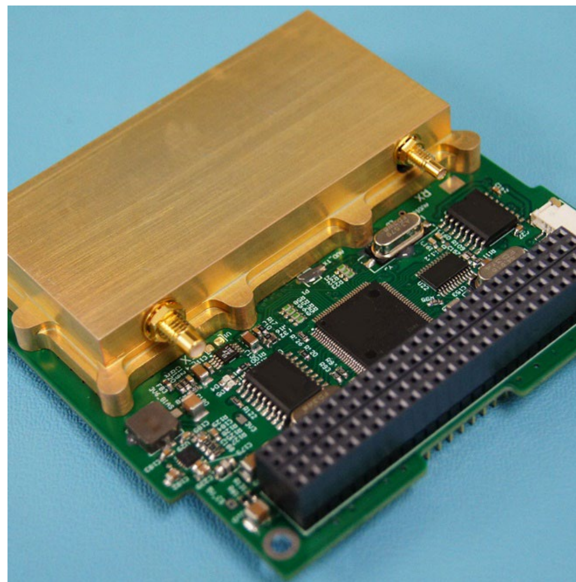

Figure 2.5: CubeSat VHF/UHF Transceiver
(Cubesat VHF/UHF Transceiver, 2022)

2.2.3 Ground station to satellite data transfer

The same transceiver displayed in Figure 2.5 is used to send commands to the satellite. That is, informing the satellite that the payload was received and validated successfully. If the payload is computed and found to be accurate, the ground station will inform the satellite to erase such payload on the onboard storage, allowing new data to be stored.

If the satellite still has more payloads to send, it will attempt to send the next one to the ground station. The process continues until the communication link is broken or the satellite does not have more data to send. The link used to send data to the ground station is called a downlink. The link used to send data to the satellite is called an uplink.

2.2.4 Sample AIVDM sentence

Figure 2.6 shows the sample AIVDM sentence transmitted by a ship to the satellite, later transmitted to the ground station. The average size of this AIVDM sentence is 50 bytes. From the sample AIVDM sentence, we can see that it can be decoded to provide information that is easy to read and understand by an ordinary person.

Data Format (Decoded)

Satellite acquires AIVDM messages emitted by the ships

!AIVDM,1,1,,A,13u?etPv2;0n:dDPwUM1U1Cb069D,0*24

Message size : ~50 Bytes

```
!AIVDM,1,1,,A,13u?etPv2;0n:dDPwUM1U1Cb069D,0*24

Message sent (UTC) : 17:21:53
MMSI               : 265547250
Latitude           : 57.660353°
Longitude          : 11.832977°
Speed              : 13.9 knots
Heading            : 41°
Course over ground : 40°
Rate of turn       : -2°/min
Navigational status: 0
```

Figure 2.6: Payload data decoded

2.3 Related study

To better solve the problem discussed in the previous section, we need to first look at previous related studies in this field of research.

There is little to no research about error detection and error correction, especially for VHF/UHF Transceivers in the nanosatellite space industry. Here we look at previous studies related to error detection and error correction.

Aiswarya et al., (2017) proposed a single bit error correction method based on Cyclic Redundancy Check (CRC) but implemented on a Field Programmable Gate Arrays (FPGAs) as a single chip solution. The discussion in Aiswarya et al., 2017 indicated that conventionally serial transceivers in FPGA uses CRC for block level data, but it consumes more area and have large complexity. CRC is one of the best error control methods (Aiswarya et al., 2017). Its computation involves a long division operation, where remainder has treated as the result and quotient is discarded. CRC encoder consists of check bits which are appended to the data bits. Error detection and correction is archived by the check bits attached to the data bits as headers. The authors successfully designed a fixed latency transceiver with single bit error correction using Verilog programming and implemented on virtex 5 FPGA. However, it is challenging to use such a solution in nanosatellite for long distance communication because it means such a spacecraft need to make use of this additional chip while the power resources are limited. Also, it means both the sending and receiving devices must implement using this chip, which limits the coverage scope of error detection and correction.

In this proposal, we propose a mechanism of error detection and correction at the application level, that is, a satellite in space can compute and send a data (payload) to a ground station without the need of special or additional IC (integrated circuit) chip, and without the need of a special operating system.

Achmad et al., (2017), studied the use of Hamming code in the detection and correction of errors in data transmission due to ease in the detection and correction of bit damaged. Hamming code operates at a bit level, meaning data received remotely as a string needs to be converted into binary before it can be used for Hamming code computation. Also, Hamming code is an error-detection method that can detect some errors, but it is only capable of single error correction (Fitriani et al., 2016).

Hamming Code method was invented by Richard W. Hamming in the 1940s (Hamming, 1950). It is an error correction system that uses parity bits for data correction. Hamming Code Method is one of the error detection methods that can detect some errors, but it is only capable of correcting one error (Deepika et al., 2016). Hamming Code computation involves the use of both data bits and check bits. The check bits are sent along with the data bits as checksums to ensure accurate data transmission. However, if the receiver detects more than one mistake, the data cannot be repaired using the Hamming Code. The proposed solution eliminates the need for converting received data into binary for calculations and verification.

The well known error detection and correction methods are Hamming Code and CRC. Both use equations to calculate and append headers to the actual data, which is then used later to determine if the packet was modified in transit. In this thesis, the proposed solution utilizes ASCII due to its universality across programming languages and operating systems, even at the firmware or microcontroller level. Any data transmission protocol can transmit data signed by the error detection and correction algorithm proposed here. While most error detection algorithms work at the hardware and bit level, the proposed algorithm operates at the application and byte level. This algorithm can recover 8 bits at once if only a single character was lost in transit. At an application level, a single character is 1 byte, which is 8 bits.

Most of the error detection and correction studies are proposed based on a single device with sub-modules, while this proposed study is based on two different devices exchanging data wirelessly over a long distance. Some of the work conducted includes the following:

Table 2.1 shows the literature survey of various papers on error detection and error correction.

Table 2.1: Literature Survey

S. No	Summary	Technology/Achitecture/ Platform	Drawback	References
1	Cyclic Redundancy Check is used to detect errors on data transmitted. This method involved a long division operation where remainder has treated as the result and quotient is discarded	Xilinx Spartan6 Field Programmable Gate Arrays (FPGAs)	An additional integrated circuit is required. This solution is unsuitable for long-range communication.	(Aiswarya et al., 2017)
2	Single Error Correcting (SEC) and Double Error to detect six error bits and correct double bit error in first and second block and single bit error in the third block using 15-bits of parity.	Verilog in Xilinx Vivado® Design Suite V14.7	Due to the ARINC429 protocol, this solution is not suitable for software application-level use. It detects up to 5 error bits.	(Santos et al., 2018)
3	A 2mm Aluminium shield was used to minimize the single event effect because	Aluminium sheet, OMERE Software and TRIM	The spacecraft now contains aluminium shields, which have increased its weight and	(Hillier et al., 2019)

	of the space radiation. This reduces the possibility of errors in the electronic circuit of the satellite.		led to higher deployment costs.	
4	Single error recovered by Foreword Error Correction (FEC) using Hamming code while burst error would be recovered by retransmitted.	VHDL, Cyclic Redundancy Check, Hybrid Automatic Repeat-Request	This solution requires a dedicated integrated circuit as it is based on VHDL	(Saleh et al., 2018)
5	Single bit error correction and double bits error detection performed on a custom electronic circuit.	VHDL, XOR	This solution requires a dedicated integrated circuit as it is based on VHDL	(Mitrych et al., 2003)
6	Tested the feasibility and accuracy of three qubit bit flip and phase flip error correcting code in quantum computer provided by International Business Machine Quantum Experience (IBM	IBM Quantum Computing	Higher computation cost for a nanosatellite with limited hardware and power resources	(Sangat et al., 2022)

	QX) cloud platform and found to have highest average accuracy 77.9% ± 3.09% on all qubits simultaneously.			
7	Implementation of iterative error detection and correction for BAN transceiver systems. The analytical results indicate that a coding gain of 2.4 dB was achieved when the decoder bit error rate (BER) was 10–4, providing a significantly superior random and burst error correction capability.	Body Area Network (BAN) transceiver	Uses a 0.18 nanometre chip resulting in high development cost.	(Kuang-Hao and Fu-Chi, 2023)
8	Checksum Error Detection. Most checksum algorithms use cryptography hash functions that take input data and generate a string which can be a	XOR checksum	Dual transmission and hashing require multiple transfers for data verification, resulting in higher	(Rihab, 2022)

	mix of letters and numbers. Sometimes checksum can also be named “hashes”.		bandwidth costs.	
9	Error Detection and Correction for Processing in Memory (PiM). An efficient error detection and correction pipeline and analyze the complex performance-area-coverage trade-off space, using three representative PiM technologies	Processing in memory (PiM) technology	This solution is suitable for in-device data transfer, not between two devices communicating wirelessly	(Hüsrev et al., 2022)
10	Efficient Method for Error Detection and Correction in In-Memory Computing Based on Reliable Ex-Logic Gates. Although detection and correction are demonstrated in NOR and NAND logic gates with	NOR and NAND logic gates	According to the researcher, this method is only effective in In-Memory computing.	(Taegyun et al., 2023)

	the memristor model, the other logic gates can be applied with the same algorithm with the appropriate module-enable signal and input-checker bits. Memristor-based stateful logic gates suffer from high error rates in logic operations due to the variations of a memristor.			
11	Automatic Error Detection in Integrated Circuits Image Segmentation: A Data-driven Approach. by adapting existing CNN-based approaches of image classification and image translation with additional pre-processing and post-processing techniques, we are able to achieve	Integrated Circuit (IC) computation. In-circuit error detection	According to the researcher, this solution can only work with a dedicated integrated circuit.	(Zhikang et al., 2023)

	recall/precision of 0.92/0.93 in wire error detection and 0.96/0.90 in via error detection, respectively.			
12	Study of Different Types of Error Detection and Correction Code in Wireless Communication. It is observed that Cyclic Redundancy Check (CRC) codes showed better performance, and hence, they are chosen for further study.	Cyclic Redundancy Check (CRC) codes	Higher computation cost due to hardware incompatibility.	(Sudha et al., 2023)

2.4 Limitations

Many studies on error detection and correction deal with single-bit errors occurring within a single electronic device. However, this particular study focuses on detecting and correcting errors that may occur when two independent devices communicate with each other over a long-distance wireless link.

CHAPTER 3
ANALYSIS AND DESIGN

3.1 Introduction

This chapter discusses data transfer between the satellite and ground station, including simulation and proposed solution design.

3.2 Analysis

We have a satellite orbiting in space, which sends important data to the ground station. However, the satellite can only transmit the data when it establishes a communication link with the ground station, which happens only when the ground station is visible. As the satellite has limited capacity to store data, it can only save the data that has not yet been transferred to the ground station. Hence, it is crucial for the ground station to inform the satellite about the successful receipt of each payload, so that the satellite can delete such payload from its onboard storage. This frees up storage space and allows the satellite to store new data. In Figure 3.1, you can see the illustration of the data transfer process between the satellite and the ground station. The ground station sends an acknowledgment message for every payload received from the satellite.

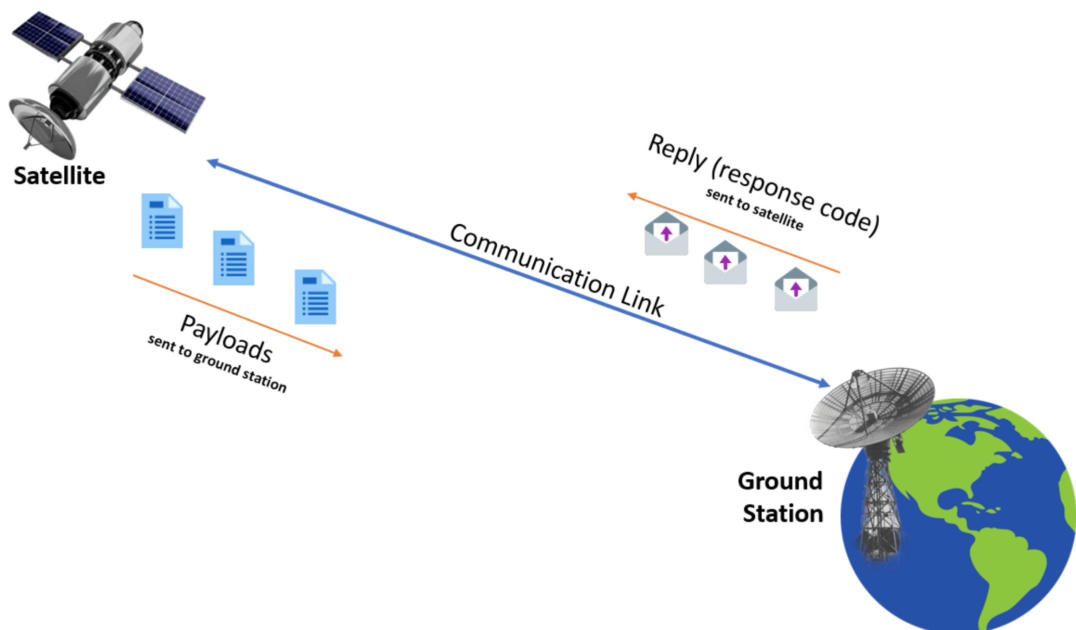

Figure 3.1: satellite and ground station communication

3.2.1 Payload analysis

For this project, our main objective is to send the payload, which is an AIVDM sentence emitted by the AIS Transponder. The AIVDM message is received by the satellite's AIS Receiver. However, since the satellite has limited battery capacity that needs to be preserved for operation, the received AIVDM message is stored in the onboard storage instead of being processed by the satellite. The satellite will send the data it receives to the ground station for processing. This means that the satellite does not have to spend processing power on the received payloads. Figure 3.2 presents a sample of 100 AIVDM messages stored on a hard disk. It is the responsibility of the satellite to remove duplicate entries when it receives these AIVDM messages from the ships, as the onboard storage has limited capacity.

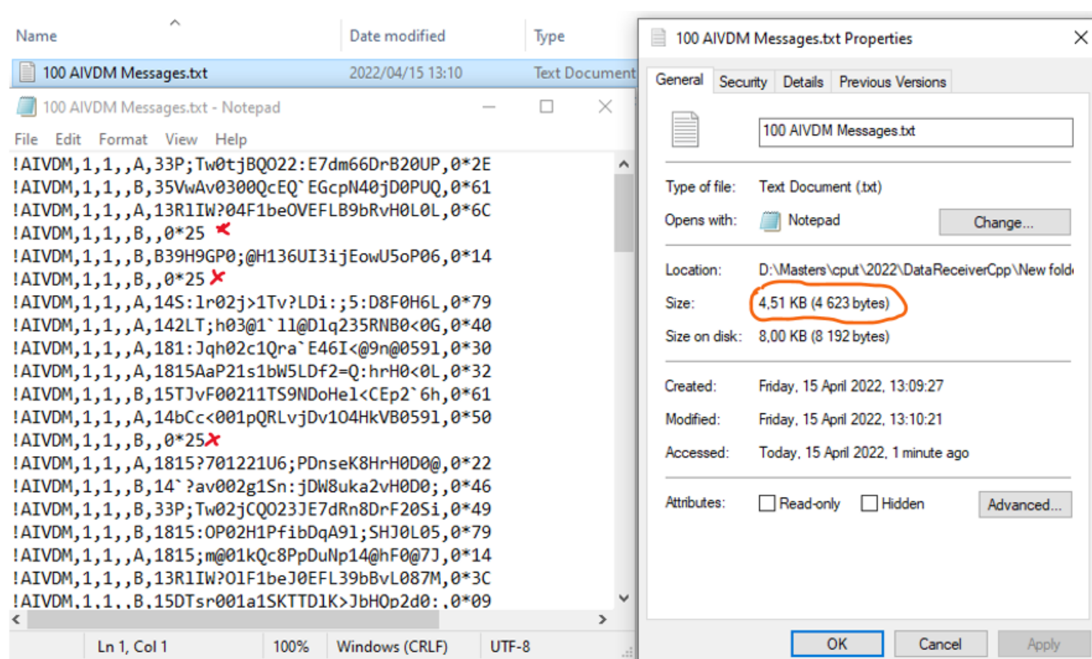

Figure 3.2: 100 AIVDM message samples

It is important to note that AIVDM is just one type of message broadcasted by the AIS transponder. We are interested in AIVDM because according to the NMEA 0183 protocol (Raymond, 2021), the AIVDM sentence contains the following data which is crucial…

- Time (UTC)
- MMSI Number
- Latitude
- Longitude
- Speed Knots
- Heading

- Course over Ground
- Rate of turn
- Ship Type
- Ship Name
- Ship Callsign
- Ship length, beam and draft
- Navigation status

Each AIVDM message is encoded and averages 47 characters. The byte size is equivalent to the number of characters as shown in Figure 3.3.

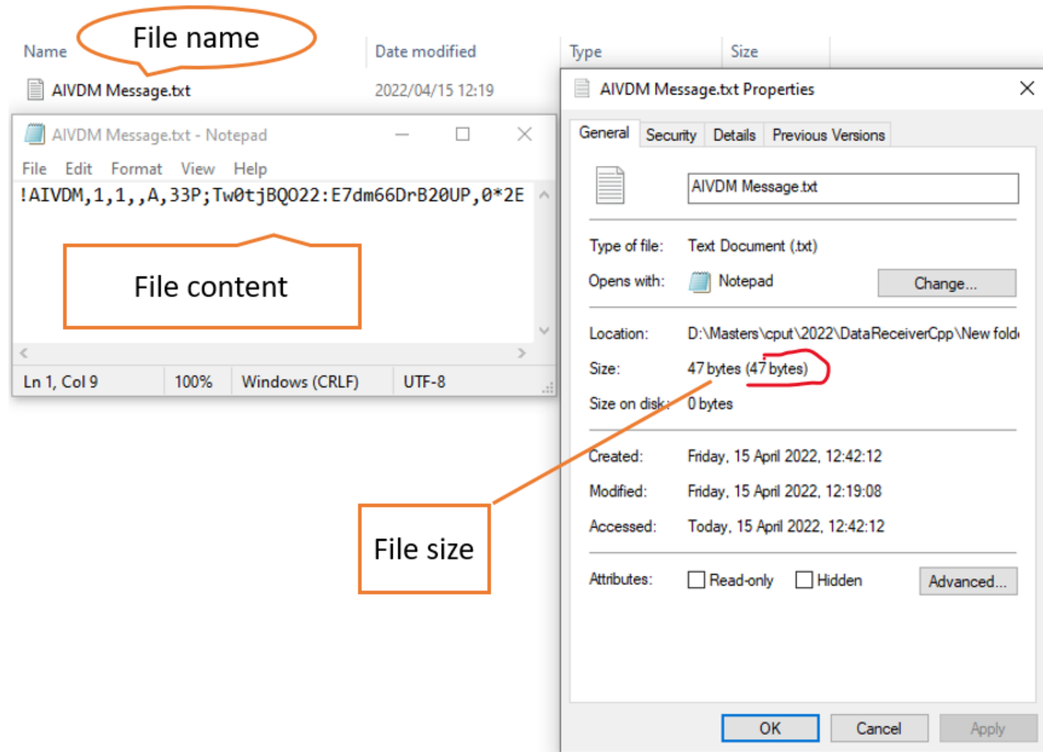

Figure 3.3: Single AIVDM message

In this project, we will simulate data exchange between two parties using a list of AIVDM messages. It is crucial for the satellite to send data in a calculated frame since the ground station is only visible for a limited time period.

3.2.2 Algorithm analysis

Two devices are communicating with each other even though they are located hundreds of kilometers apart. However, during data transmission, errors may occur, and the established link may be cut off at any time. To solve this problem, an algorithm has been proposed in this thesis. The algorithm makes use of ASCII to sign and validate the payloads received from the satellite. Upon validating a payload, the algorithm sends a feedback message to the satellite indicating whether the payload was successfully validated or not. If the satellite receives such notification, it decides

to delete the successfully sent payload and retain the ones that were not successful. When the satellite attempts to send payloads to the ground station, some of them may not be successful. In such cases, the satellite will make another attempt to send those payloads during the next ground station visibility timeframe. The process of transferring data between the satellite and the ground station is explained in Figure 3.4 below. The algorithm's implementation will be discussed in the upcoming chapter.

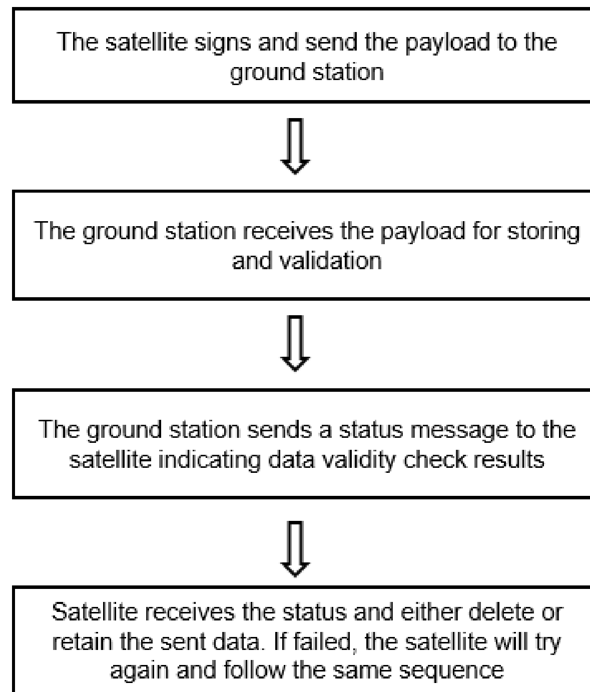

Figure 3.4: Steps of data transfer between satellite and ground stations

3.2.3 Simulation hardware analysis

A nanosatellite requires a transceiver that enables two-way communication. One such transceiver is illustrated in figure 2.5 of chapter 2, known as the CubeSat VHF/UHF Transceiver. This device can provide full-duplex communication, with AFSK offering a data rate of 1200 bps and GMSK providing 9600 bps. Since we don't have an actual nanosatellite for testing, we have identified a hardware device called nRF24L01+ that can facilitate wireless data transmission. Figure 3.5 illustrates a transceiver device that uses 2.4 GHz unlicensed frequency to transmit and receive data. This device, provided by Nordic Semiconductor, supports on-air data rates of 250 kbps, 1 Mbps, and 2 Mbps.

Figure 3.5: nRF24L01+ 2.4 GHz Transceiver module
(NRF24L01 Module - ITEAD Wiki, 2022)

For the simulation to be effective, it should closely match the communication speed used by the satellite and ground station. Additionally, the simulation device should be connected to a computer to perform its function accurately. In order to connect the nRF24L01+ to a computer, a second hardware device is needed. Figure 3.6 displays the USB adapter that was chosen for this project. Both hardware devices allow the transfer speed of data to be set. For this project, the default baud rate speed of 9600 bps was selected for data transfer. This speed is also used by some nanosatellites.

Figure 3.6: USB Adapter for NRF24L01+
(USB Adapter for NRF24L01+, 2022)

3.2.4 **Simulation software analysis**

Both the satellite and the ground station require the ability to send, receive, and process data. Simply having a transceiver is not enough to send data. A software or firmware is required to instruct the transceiver what to send. In the case of a satellite, this can be achieved through a firmware, while a software may be used at the ground station. A firmware operates on limited hardware, providing specific functions with the advantage of lower power consumption as it does not require a full operating system. This is especially important for a satellite, which has limited power sources and battery capacity. Figure 3.7 shows the functions that need to be performed by the firmware of the satellite.

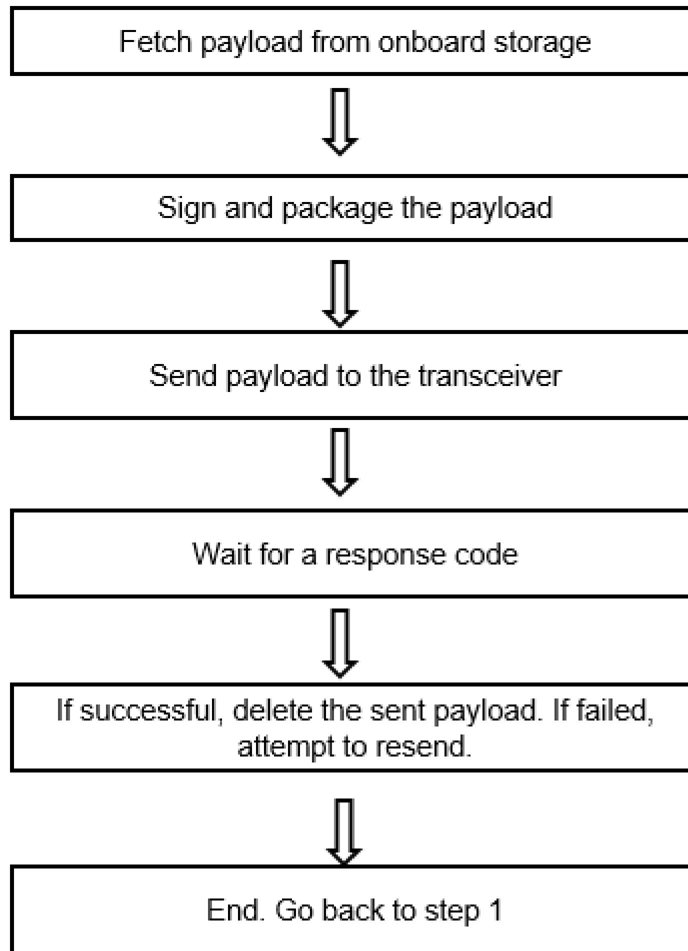

Figure 3.7: Ordered firmware functionalities for satellite

Two programs have been developed for a project, with one program used at the sending device and the other at the receiving device. Both programs run on Windows operating system, but on different computers. The proposed algorithm is used by both programs and they can be developed in any programming language. In this project, C++ was used to develop both the programs. The decision to develop these programs instead of using existing software was made after it was observed that the proposed algorithm needed to be implemented. Figure 3.8 shows the processes or tasks that a software needs to perform at the ground station.

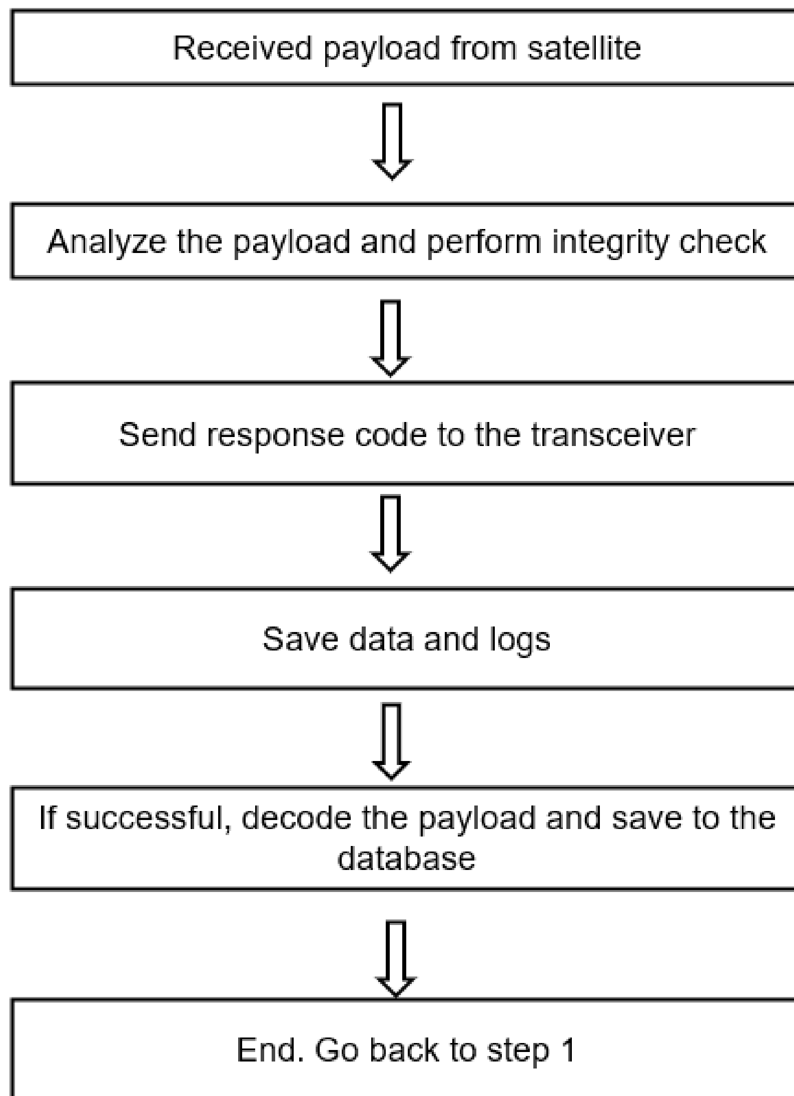

Figure 3.8: Ordered software functionalities for ground station

3.3 Design

3.3.1 Algorithm design

The proposed algorithm takes advantage that each character sent between devices is available on the ASCII table. Figure 3.9 shows the ASCII table for reference. The algorithm needs to be able to compute a signature which is appended as a header to the payload sent to the ground station. This method was chosen because ASCII is the standard understood by any programmable chip and programming language. This gives us the ability to design an algorithm that runs from minimal hardware to a full operating system like Windows. It must be noted that the proposed algorithm can be developed in any programming language and can run on any operating system.

ASCII TABLE

Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char
0	0	[NULL]	32	20	[SPACE]	64	40	@	96	60	`
1	1	[START OF HEADING]	33	21	!	65	41	A	97	61	a
2	2	[START OF TEXT]	34	22	"	66	42	B	98	62	b
3	3	[END OF TEXT]	35	23	#	67	43	C	99	63	c
4	4	[END OF TRANSMISSION]	36	24	$	68	44	D	100	64	d
5	5	[ENQUIRY]	37	25	%	69	45	E	101	65	e
6	6	[ACKNOWLEDGE]	38	26	&	70	46	F	102	66	f
7	7	[BELL]	39	27	'	71	47	G	103	67	g
8	8	[BACKSPACE]	40	28	(	72	48	H	104	68	h
9	9	[HORIZONTAL TAB]	41	29	)	73	49	I	105	69	i
10	A	[LINE FEED]	42	2A	*	74	4A	J	106	6A	j
11	B	[VERTICAL TAB]	43	2B	+	75	4B	K	107	6B	k
12	C	[FORM FEED]	44	2C	,	76	4C	L	108	6C	l
13	D	[CARRIAGE RETURN]	45	2D	-	77	4D	M	109	6D	m
14	E	[SHIFT OUT]	46	2E	.	78	4E	N	110	6E	n
15	F	[SHIFT IN]	47	2F	/	79	4F	O	111	6F	o
16	10	[DATA LINK ESCAPE]	48	30	0	80	50	P	112	70	p
17	11	[DEVICE CONTROL 1]	49	31	1	81	51	Q	113	71	q
18	12	[DEVICE CONTROL 2]	50	32	2	82	52	R	114	72	r
19	13	[DEVICE CONTROL 3]	51	33	3	83	53	S	115	73	s
20	14	[DEVICE CONTROL 4]	52	34	4	84	54	T	116	74	t
21	15	[NEGATIVE ACKNOWLEDGE]	53	35	5	85	55	U	117	75	u
22	16	[SYNCHRONOUS IDLE]	54	36	6	86	56	V	118	76	v
23	17	[ENG OF TRANS. BLOCK]	55	37	7	87	57	W	119	77	w
24	18	[CANCEL]	56	38	8	88	58	X	120	78	x
25	19	[END OF MEDIUM]	57	39	9	89	59	Y	121	79	y
26	1A	[SUBSTITUTE]	58	3A	:	90	5A	Z	122	7A	z
27	1B	[ESCAPE]	59	3B	;	91	5B	[	123	7B	{
28	1C	[FILE SEPARATOR]	60	3C	<	92	5C	\	124	7C	|
29	1D	[GROUP SEPARATOR]	61	3D	=	93	5D	]	125	7D	}
30	1E	[RECORD SEPARATOR]	62	3E	>	94	5E	^	126	7E	~
31	1F	[UNIT SEPARATOR]	63	3F	?	95	5F	_	127	7F	[DEL]

Figure 3.9: ASCII table

(File:ASCII-Table-wide.svg - Wikipedia, 2022)

Figure 3.10 shows the design of the proposed algorithm and its calculations. A signature is computed and appended as a header into the actual payload. The receiving device must be able to use the header to the computer and determine if the payload is valid. The proposed algorithm assumes that data is sent on a FIFO (First In First Out) sequence as it is the UART buffer that forces each byte of your serial communication to be passed on in the order received (Yang, 2009). With this knowledge, we can be certain that the data does not get scrambled around while in transit. It also guarantees that the header will always be the first to arrive. The receiving device will analyse the headers, and if valid, it will then start to unpack the actual payload and compute using the same method used to sign it. The computed signature on the receiving device need to match the one sent as a header to the actual payload. If they match, the receiving device needs to send back a response status code ‘P’ indicating ‘Passed’, meaning no errors were detected. If they don’t match, one of the other two status codes will returned. ‘R’ and ‘F’ are the other possible status codes that can be returned by the receiving device. ‘R’ for ‘Reconstructing’, and ‘F’ for ‘Failed’. If no status code is received by the satellite, it needs to default to Failed, that is, do not erase the payload because no confirmation message received from the ground station indicating that the payload successfully arrived.

Actual Payload

!AIVDM,1,1,,A,33P;Tw0tjBQO22:E7dm66DrB20UP,0*2E

Signed Payload (ready to send)

!AIVDM,1,1,,A,33P;Tw0tjBQO22:E7dm66DrB20UP,0*2E|3088|47

Signature

|3088|47

47 = Number of characters

3088= A sum of all character's ASCII equivalent decimal value

Example 1	**Example 2**
ABC = 65 + 66 + 67 = 198	abc = 97 + 98 + 99 = 294
Signed payload	**Signed payload**
ABC|198|3	abc|294|3

Figure 3.10: Algorithm calculations

3.3.2 Hardware design

nRF24L01+ is an electronic hardware device capable of performing real-time wireless data transfer and was chosen in this project for simulation. This device will be connected to a USB adapter, which is then connected to a computer running windows operating system. Figure 3.11 shows the hardware connectivity used in this project for simulation. Device A will be transmitting data at a baud rate of 9600bps into device B. Device B will receive the payload process it and determine its integrity. Device B will save the received data into a hard disk and return a status message to device A indicating if the data is complete and valid. In this project, both devices will operate at a baud rate of 9600bps.

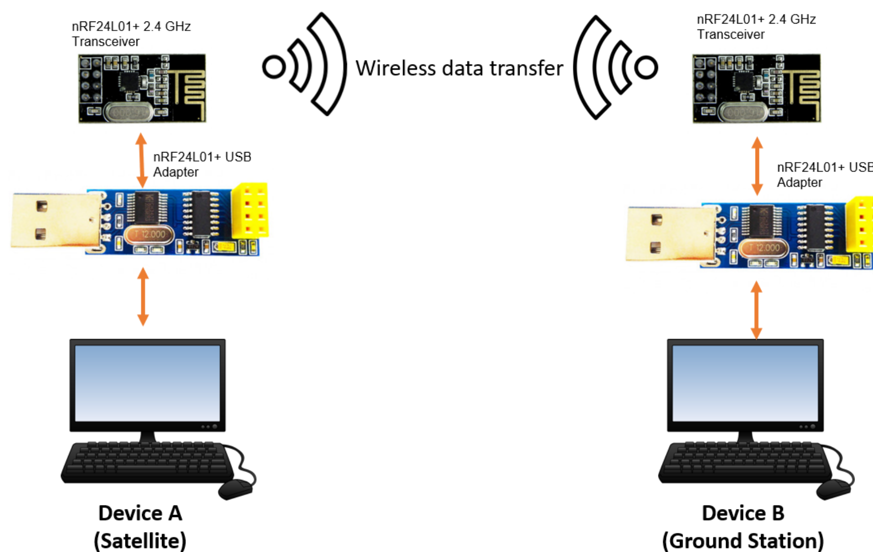

Figure 3.11: Hardware connectivity diagram

3.3.3 Software design

The real-time simulation software will be designed using Embarcadero Rad Studio and the programming language chosen is C++. Embarcadero Rad Studio is a licensed product and a 1-year license to use the product is already obtained. It is, however, also available as a community version with limited features. Two applications will be developed and need to be able to communicate with the hardware (transceiver) to exchange data in real time. The software to be developed will not automatically detect the hardware connected to the computer. The only way to communicate with the hardware is to install drivers compatible with that hardware. The selected USB adapter uses a USB-to-Serial communication chip known as CH340T. This chip is very popular, by installing its drivers, the operating system will be able to generate a new serial communication (COM) port number which represents the connected device.

The sending software will be able to load a file from a local hard drive containing a list of AIVDM messages that are going to be sent as payloads to the receiving software via the transceiver. The sender will be able to simulate data loss by chopping the tail of the data sent to the ground station. The options will be provided as no data loss, 1-byte data loss, and more than 1-byte data loss. The receiving software needs to be able to reconstruct the lost character if a 1-byte data loss occurs. Both sending and receiving software need to have an option of selecting the COM port and the baud rate for data transfer. The sending software needs to have a window to indicate the logs of the status messages received from the ground station. It also needs to have a window to display all the payloads read from the file, together with the status indicating the progress of the data being sent. The receiving software (ground station) will have a window to display the received payload, computation results, and logs.

3.4 Summary

This chapter explained both the analysis and the design of the proposed system and the expected operation sequence of the system. The next chapter will show and explain the implementation of the proposed system.

CHAPTER 4
IMPLEMENTATION AND SOFTWARE OUTPUT

4.1 Introduction

In this chapter, the proposed solution is implemented and presented. A detailed guide of building the project is provided with screenshots and details of each major component.

4.2 Tools and materials

In this project, a development studio Embarcadero Rad Studio is used for both the development and debugging of the solution. This studio was chosen because of its familiarity and ease of use while providing support for C++ programming. Another reason this studio was chosen is the ability to add third-party components for both open-source and commercial use. The version of the studio used in this project is Rad Studio XE10.1 Berlin. A 1-year license for this studio is already acquired at a cost of R28 000 with a student discount already applied.

A Dell OptiPlex 7040 is used for development and testing. This computer runs a Windows 10 operating system. The version of the operating system is Windows 10 Pro 21H1 build 19043.1586. The Rad Studio is installed in this Dell computer and the license is applied upon installation. The Windows license is obtained prior to the registration of this study.
A driver pack called CH341SER is also installed on the Dell computer to enable the drivers for the USB adapter so that the computer will be able to recognize the device and assign a unique port number for operations. This driver pack enables support for CH340T, which is the chip found in the USB adapter.

The hardware used for data transfer is the nRF24L01+ working together with the nRF24L01+ USB Adaptor. In this project, logs will be generated at the receiving end (ground station) in an output format of .csv. A program called Notepad will be used to view the generated logs. A built-in Windows program called Computer Management is used to check the current port number assigned to the connected USB adaptor. This port number is needed for the developed software to connect.

4.3 Hardware connectivity

The first step is to configure the hardware device used for communication. The nRF24L01+ is connected to the USB adapter as shown in Figure 4.1. At this stage, the USB adaptor is then connected to a USB port of a computer. After connecting into

computer, the Windows operating system will try to detect the connected device, but it will fail because no drivers are installed for the connected device. Once the device is connected to a computer, regardless of the drivers, it will emit two blue LEDs (Light Emitting Diodes), one will be steady blue while the other one is blinking at a rate of 500 milliseconds.

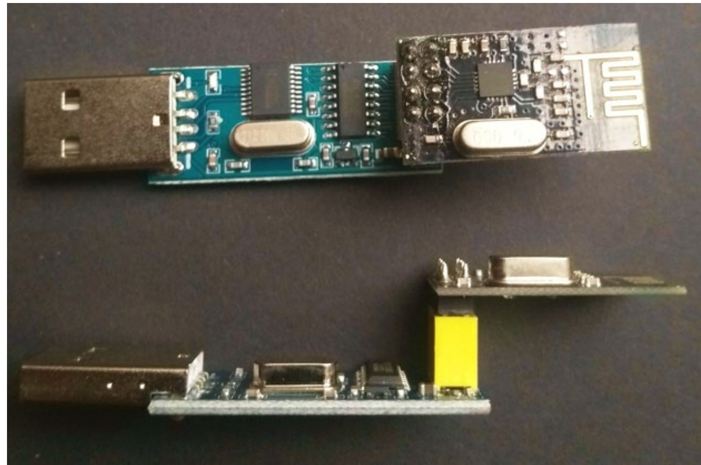

Figure 4.1: nRF24L01+ connected a USB adapter

4.4 Driver installation and COM port

In this step, the drivers are installed. The drivers are provided and recommended by Sparkfun and instruction instructions of installation are well explained here (How to Install CH340 Drivers - learn.sparkfun.com, 2022)**.** Figure 4.2 shows the installation dialog after executing the CH341SER.EXE setup. The INSTALL button needs to be pressed to begin the installation. A popup dialog will appear with a message saying, 'Driver install success!', which guarantees that the installation was a success. If no popup with the message above, then assume the installation failed. It is worth noting that driver installation requires elevated privileges (Administrator Mode) to successfully install the drivers.

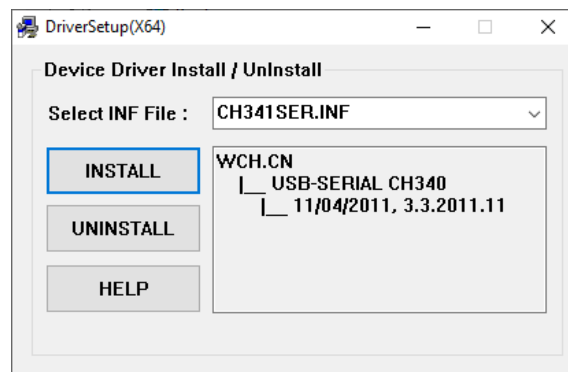

Figure 4.2: Serial driver installation setup

Once the drivers are installed, the USB adapter is then connected to the computer. Figure 4.3 shows the COM port allocated to the device after it got detected by the operating system. Windows reserves the right to change the COM port number assigned to a USB device depending on the USB port used. That is, connecting the adapter to another USB port may result in another different port number being assigned to the device. It is important to monitor the ports listed on the Computer Management software while plugging in the USB adapter to identify the port assigned to the device. The developed software needs to connect to the correct port and Windows may already have had other ports in the list which may create confusion if not noted. Figure 4.3 also illustrates the scenario where more COM ports already exist in Windows OS.

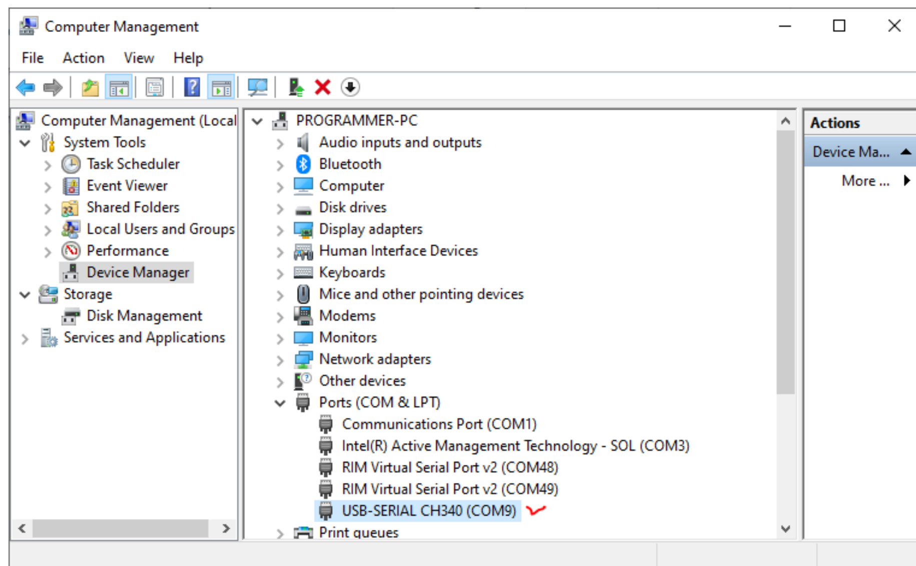

Figure 4.3: Test hardware device detected

4.5 Algorithm development (Satellite/Sender)

Figure 4.4 illustrates the execution flow for transmitting data from a satellite to a ground station. Initially, data is loaded from the local disk and prepared for transmission once the communication link is established. Subsequently, two fundamental arithmetic operations (summation and counting) are performed after converting each character to its corresponding integer value from the ASCII table. The results of these operations are appended to the payload as a header. The signed payload is then transmitted to the ground station, which responds to confirm the success of the transmission. If the data is successfully received by the ground station, it is deleted from the local storage. Conversely, if no response is received or a failure code is returned, the local data is retained and retransmitted.

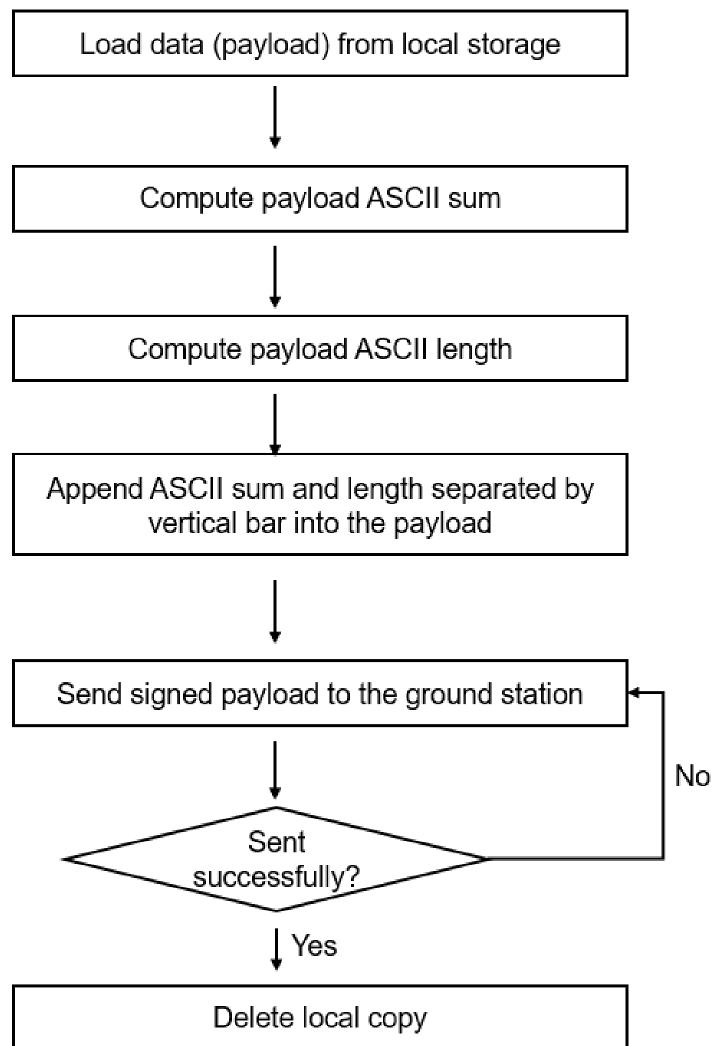

Figure 4.4: The execution flow for transmitting data from a satellite to a ground station

Figure 4.5 shows the code of the algorithm used to send data from the sender, that is, simulating the satellite. A class called *TfrmMain* was developed and has three methods named *sendNextData*, *get_ASCII_sum* and *getDataLength* respectively.

The *get_ASCII_sum* is accepting the payload as a parament and returns an integer value which is the sum of all the decimal values found on each character of the payload given. The sum is then computed by extracting the equivalent decimal value of a character from the ASCII table and adding them all. The calculated value is then used as a header into the actual payload followed by the data length.

The *getDataLength* is accepting the payload as a parament and returns an integer value of the quantity of the characters found on the payload. The length of the characters is then appended into the actual payload as a header. In addition to the sum and the length, two more characters are also added to the payload which is used to

split the data. The character | (also known as Pipe or Vertical Bar) is used to separate the values from the payload.

```
void TfrmMain::sendNextData()
{
    finalLine= lstRecords->Strings[lstData->ItemIndex].Trim()+"|"+
                            IntToStr(get_ASCII_sum(lstRecords->Strings[lstData->ItemIndex].Trim()))+"|"+
                            IntToStr( getDataLength(lstRecords->Strings[lstData->ItemIndex].Trim()) );
    mydata_ready->Clear();
    while(finalLine.Length()>25)
    {
        mydata_ready->Add(finalLine.SubString(1,25));
        finalLine=finalLine.Delete(1,25);
    }
    mydata_ready->Add(finalLine);

    // Check Dataloss simulation
    if(rdgDataLoss->ItemIndex==1)   // loose 1st character
    {
         mydata_ready->Strings[0]=mydata_ready->Strings[0].Delete(1,1);
    }
    else if(rdgDataLoss->ItemIndex==2)   // loose +=2 characters
    {
         mydata_ready->Strings[0]=mydata_ready->Strings[0].Delete(1,Random(5)+2);
    }

    for(int a=0;a<mydata_ready->Count;a++)
    {
        com->WriteText(mydata_ready->Strings[a]);
        Sleep(1500);
        Application->ProcessMessages();
    }
    com->WriteText("\r\n");
    lstData->Items->Item[lstData->ItemIndex]->SubItems->Strings[1]="YES";
}
//---------------------------------------------------------------------------
int TfrmMain::get_ASCII_sum(String data)
{
     int sum=0;
     for(int a=1;a<=data.Length();a++)
     {
         sum+= int(data[a]);
     }
     return sum;
}
int TfrmMain::getDataLength(String data)
{
    return data.Trim().Length();
}
```

Figure 4.5: Algorithm code for payload packaging (satellite)

The *sendNextData* method does not accept any parameters and does not return any value. It uses the variable and the methods declared in the same class to package and send the data. A variable called *finalLine* is used to store each signed payload. In this context, a signed payload is a payload with both the sum and the length appended as a header separated by a pipe. Because the signed payload can be too long for the UART buffer to handle, a variable called *mydata_ready* was introduced which allows sending of the signed payload a chunk of 25 characters. That is, if the signed payload has a total of 55 characters, the last 25 characters will be sent first, followed by the middle 25 character, then the first 5 characters are sent last. This sequence ensures that the header is the first thing to arrive at the receiver.

Once all characters are sent, a carriage return and a newline character is sent to indicate the end of the payload. In C++, the carriage return and newline characters are

indicated by \r\n. This explanation is also shown on Figure 4.5 and can be implemented in any programming language. Figure 4.6 shows the code for processing the response based on the character passed as a parameter.

```
void TfrmMain::processResponse(String data)
{
    // process reponse
    if(data.Trim().Length()==1)
    {
        if(data.Trim()=="P")
        {
            // passed
            lstData->Items->Item[lstData->ItemIndex]->SubItems->Strings[2]="Passed";
            lstData->Items->Item[lstData->ItemIndex]->SubItems->Strings[3]="YES";

            if(lstData->ItemIndex<lstData->Items->Count-1)
            {
              lstData->ItemIndex=lstData->ItemIndex+1;
              continue_sending=true;
            }
            else
            {
                // End timer
                tmrCounter->Enabled=false;
            }
        }
        else if(data.Trim()=="F")
        {
            // Failed
            lstData->Items->Item[lstData->ItemIndex]->SubItems->Strings[2]="Failed";
            lstData->Items->Item[lstData->ItemIndex]->SubItems->Strings[3]="NO";

            lstData->ItemIndex=lstData->ItemIndex;
            continue_sending=true;
        }
        else if(data.Trim()=="R")
        {
            // Reconstructing
            lstData->Items->Item[lstData->ItemIndex]->SubItems->Strings[2]="Reconstructing";
            lstData->Items->Item[lstData->ItemIndex]->SubItems->Strings[3]="YES";

            if(lstData->ItemIndex<lstData->Items->Count-1)
            {
              lstData->ItemIndex=lstData->ItemIndex+1;
              continue_sending=true;
            }
            else
            {
                // End timer
                tmrCounter->Enabled=false;
            }
        }
    }
}
```

Figure 4.6: Algorithm code for response processing (satellite)

A new method called *processResponse* was introduced to process the response from the receiver (ground station). This method can only process three possible response codes flagged as ‘P’, ‘F’ and ‘R’. The ‘P’ flag indicates that the signed payload arrived at the desired destination, and the verification was successful. At this stage, the satellite can safely delete the sent payload.

The ‘F’ flag indicates that either a portion or all the data of the signed payload arrived at the desired destination, but the verification process failed because errors were detected. That is, the integrity that determines the validity of the payload failed.

At this stage, the satellite does not need to erase the payload because it was not successfully sent to the ground station. The satellite may need to resend the signed payload until a success or reconstruction code is returned.

The ‘R’ flag indicates that the sent payload was discovered to be missing a single character (8 bits), and it was recovered by the receiving algorithm. When the satellites receive the ‘R’ flag, it knows that the payload sent lost a single character along the way, and the lost character was re-constructed at the ground station level. The process of single character reconstruction is declared as error correction in this thesis. The method *processResponse* uses the flags to mark the statuses of the payloads displayed on the sending software. The word ‘Passed’ is displayed when the flag ‘P’ is received. The word ‘Failed’ is displayed when the flag ‘F’ is received. The word ‘Reconstructing’ is displayed when the flag ‘R’ is received. In this implementation, the word ‘YES’ is used to mark the payload for erasing. The word ‘NO’ is used to mark the payload as do-not-erase.

The satellite must only delete the payload if either of the status ‘P’ or ‘R’ is received. There is a chance of not receiving any flag from the receiver. That is when the connection is lost between the two devices. In this project, the focus is on error detection and correction. The device connectivity mechanism is not implemented. That is, auto-connecting and sending payload when the ground station is visible, or when the connection was lost is not implemented. In this implementation, the connection is done manually between both devices.

4.6 Algorithm development (Ground Station/Receiver)

Figure 4.7 illustrates the procedural sequence for ground station data processing and response generation. Initially, a signed payload is received from the satellite. This payload is then decomposed into three components: the unsigned payload, the ASCII sum, and the character count. At the ground station, a new ASCII sum and character count are computed for the unsigned payload. These computed values are then compared with the corresponding values received from the satellite. Based on this comparison, the ground station system generates a response code, which is transmitted back to the satellite. The received payload is subsequently stored in the database, and a log file is generated to document the transmission transaction.

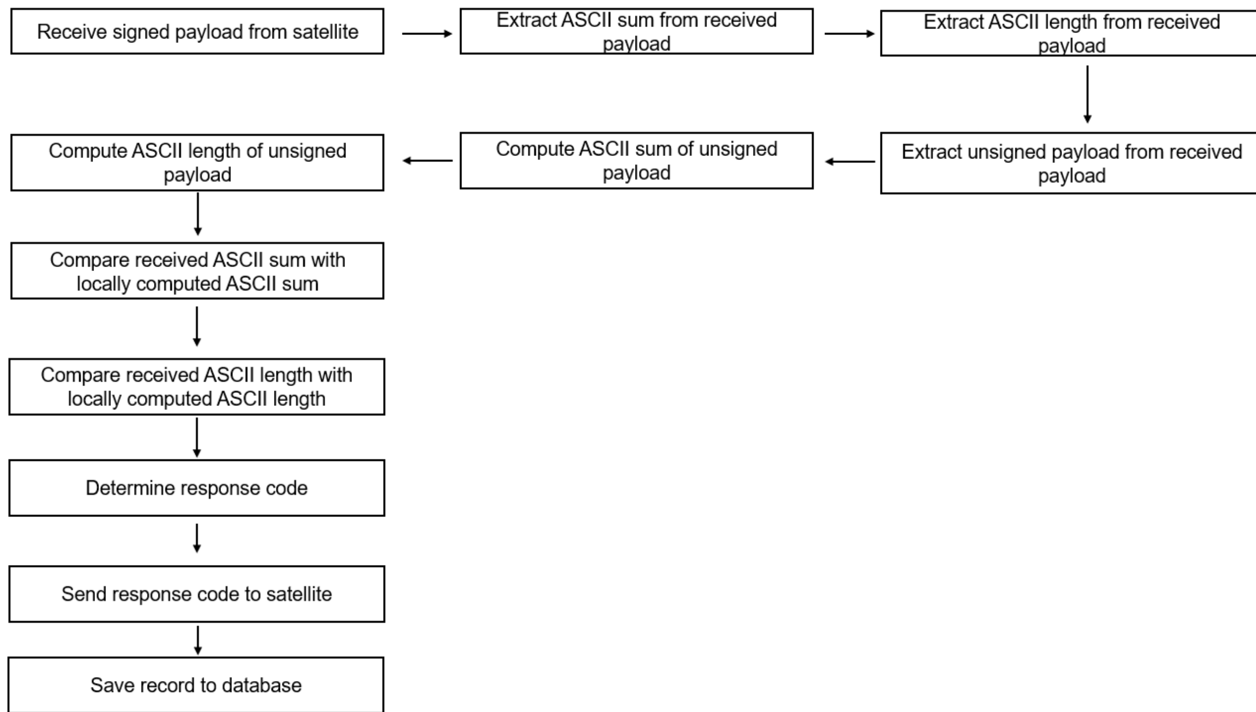

Figure 4.7: The procedural sequence for ground station data processing and response generation

Figure 4.8 shows the actual code of the algorithm used by the receiver's program, that is, simulating the ground station. The receiving software also has a class called *TfrmMain*, with the same naming and implementation of the methods *get_ASCII_sum* and *getDataLength*. Refer to figure 4.5 for the implementation of these methods. An additional method named *validateData* is implemented by the receiver software which is responsible for unpacking the signed payload. This method is also responsible for generating and sending the response flag. This method is also responsible for generating logs and saving them to the hard disk.

The *validateData* method accepts one parameter which is a string, a complete signed payload sent by the sending software.
The received payload is checked if it contains a pipe (vertical bar) character which is expected to be in the header of the payload separating the payload, the sum, and the length. The payload is considered to be invalid if the expected sequence of characters is not found. If the payload was found to be containing the correct signature, the process of unpacking begins. A dynamic array variable called *validate* is used to store chunks of data after it was split by the separating character (pipe).

After the split operation, the variable validate[0] will contain the received payload without the signature. The variable validate[1] will contain the sum of the payload that

was appended by the sender as a header. The variable validate[2] will contain the length (number of characters) that was appended by the sender as a header. The data stored on validate[0] , validate[1] and validate[2] is coming from the sender. This information is then save saved as logs in case is needed later.

The variable ASCII_Sum is then used to store the computed ASCII sum of the payload, which is computed by the receiving software. The computed sum (ASCII_Sum) is then compared with validate[1], which is the sum sent as a header in the payload. Another variable Character_Count is used to store the computed length of the payload, which is computed by the receiving software. The computed length (Character_Count) is then compared with validate[2], which is the length sent as a header in the payload.
For a payload to be declared valid, two conditions need to be satisfied. The first condition is that ASCII_Sum and validate[1] must be equal. The second condition is that Character_Count and validate[2] must also be equal. If both these conditions are met, the payload is declared to be valid, and the status code P is returned to the sender notifying that the payload was received, and the verification was a success. If the difference between Character_Count and validate[2] is an integer value 2 or more, the payload is declared to have errors, and the status code F is sent as a response to the sender notifying that the payload was received but the verification failed because errors were detected. If the difference between Character_Count and validate[2] is exactly 1, it indicates that a single character was lost, that is, 8 bits of data were lost.
The lost character is re-constructed by subtracting ASCII_Sum from validate[1], which is a formula (validate[1] - ASCII_Sum), and the resulting number is used to retrieve its corresponding character from the ASCII table. The retrieved character is exactly the one that was lost.

```
void TfrmMain::validateData(String data)
{
    String response="F";
    if(data.Trim().Pos("|")>0 && data.Trim().LastDelimiter("|")>data.Trim().Pos("|"))
    {
        TStringDynArray validate=SplitString(data.Trim(),"|");

        logsLine=FormatDateTime("dd-mm-yyyy hh:nn:ss",Now())+"#";

        logsLine=logsLine+data+"#";
        mmLogs->Lines->Add("Actual Payload: \n\t"+validate[0]);
        mmLogs->Lines->Add("Header ASCII Sum: \t"+validate[1]);
        mmLogs->Lines->Add("Header Character Count: \t"+validate[2]);

        logsLine=logsLine+validate[0]+"#";
        logsLine=logsLine+validate[1]+"#";
        logsLine=logsLine+validate[2]+"#";
        mmLogs->Lines->Add("------Validation Begin------\n");

        String ASCII_Sum=IntToStr(get_ASCII_sum(validate[0]));
        mmLogs->Lines->Add("Computed ASCII Sum: \t"+ASCII_Sum);

        String  Character_Count=IntToStr(getDataLength(validate[0]));
        mmLogs->Lines->Add("Computed Character Count: \t"+Character_Count);

        logsLine=logsLine+ASCII_Sum+"#";
        logsLine=logsLine+Character_Count+"#";

        if( validate[1].ToInt()== get_ASCII_sum(validate[0]) )
        {
           mmLogs->Lines->Add("ASCII Sum Match: \tTrue");
           logsLine=logsLine+"TRUE#";
        }
        else
        {
            mmLogs->Lines->Add("ASCII Sum Match: \tFalse");
            logsLine=logsLine+"FALSE#";
        }

        if( validate[2].ToInt()== getDataLength(validate[0]) )
        {
           mmLogs->Lines->Add("Character Count Match: \tTrue");
           logsLine=logsLine+"TRUE#";
        }
        else
        {
            mmLogs->Lines->Add("Character Count Match: \tFalse");
            logsLine=logsLine+"FALSE#";
        }
        if( validate[1].ToInt()== get_ASCII_sum(validate[0]) &&
             validate[2].ToInt()== getDataLength(validate[0]) )
        {
            response="P";
            logsLine=logsLine+"P#NONE";
            mmLogs->Lines->Add("Return status: \t"+response+ " ( Passed )");
            com->WriteText(response.Trim()+"\r\n");
        }
        else if( validate[1].ToInt() != get_ASCII_sum(validate[0])
             && getDataLength(validate[0]) == validate[2].ToInt()-1  )
        {
            response="R";
            String reconstructed=String( Char( validate[1].ToInt()-
                                  get_ASCII_sum(validate[0]) ) )+validate[0];
            logsLine=logsLine+"R#"+ reconstructed;
            mmLogs->Lines->Add("Return status: \t"+response+ " ( Reconstructing )");
            mmLogs->Lines->Add("Reconstructed: \t"+reconstructed);

            com->WriteText(response.Trim()+"\r\n");
        }
        else
        {
            response="F";
            logsLine=logsLine+"F#NONE";
            mmLogs->Lines->Add("Return status: \t"+response+ " ( Failed )");
            com->WriteText(response.Trim()+"\r\n");
        }
        mmLogs->Lines->Add("------Validation End------\n");

        logs->Add(logsLine);
        logs->SaveToFile(logsFilename);
    }
}
```

Figure 4.8: Algorithm code for payload processing (ground station)

4.7 Application development (Sender and Receiver)

Rad studio Is used to develop the proposed solution. Figure 4.9 shows the development software opened with two projects named DataProcessor.exe and DataReceiver.exe respectively. DataProcessor.exe represents the sender while DataReceiver.exe represents the receiver. The sender is the software sending the payload to the ground station, which is the software used by the satellite to package and send data. The receiver is the software at the receiving end (ground station) which is processing the data coming from the sender and proving a feedback message. Both sending and receiving software can be developed in any programming language to run on any physical device and on any architecture.

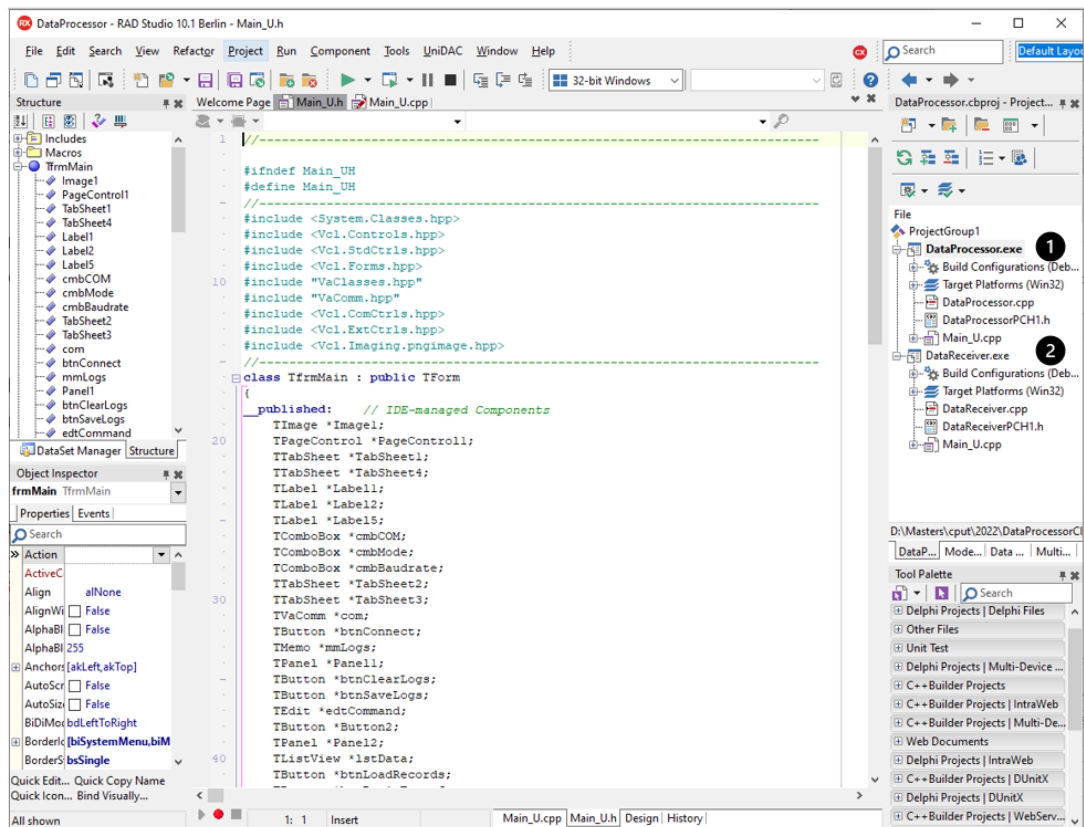

Figure 4.9: Development studio and source code

4.8 Output of the sending software

Figure 4.10 shows the output of the sending software which is responsible for acting as a satellite. It has the option to simulate data loss with three options. The software also has the 'Data' tab which is responsible for loading the records from the local disk (onboard storage) and preparing them for sending to the receiving device.

Figure 4.10: Sender software output (DataProcessor.exe)

The same 'Data' tab is used to display the payload with their respective statuses. Another tab named 'Settings' is used to configure and connect to the COM port, which is the device intended to be used for wireless communication. This tab has a connectivity mode, and by default, the 'Satellite' mode is selected to indicate that this software will represent the satellite. The baud rate is pre-selected to 9600bps as it will be used to negotiate the rate at which the data is transferred between the devices. The 'Connect' button is used to connect to the device. Once connected, bidirectional data transfer can happen. A test group box is also provided to test data transfer after the connection was established between both devices. Figure 4.11 shows the 'Settings' tab with the functionalities mentioned. Please see Appendix A for a full source code of this program.

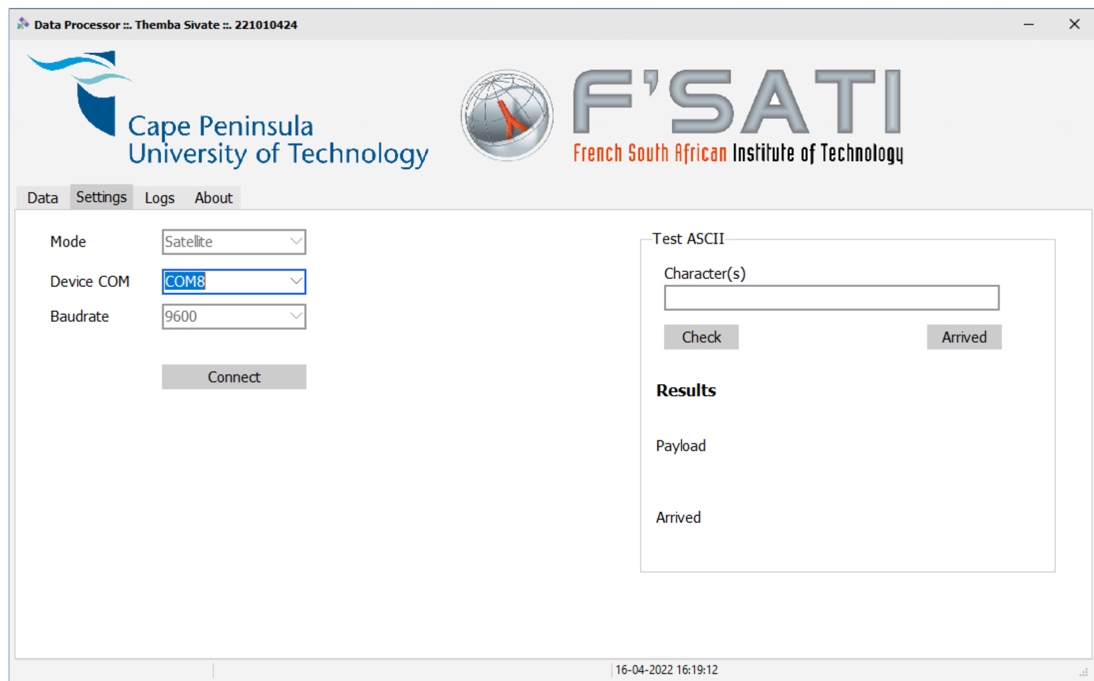

Figure 4.11: Sender software – settings tab

Another useful tab is the 'Logs', which is responsible for displaying data such as responses. The 'Logs' tab is shown on figure 4.12 and has a function to export the logs into file by clicking the Save to file button. Another button is provided to clear the logs. Another useful feature is the ability to send a test message and observe the feedback coming through the 'Logs' tab.

The tabs named 'Data', 'Settings', and 'Logs' provide the necessary functionalities to perform real-time data transfer, which is sending and receiving data through the nRF24L01+ transceiver via the USB adapter. This software runs on Windows and supports both 32bit and 64bit architecture.

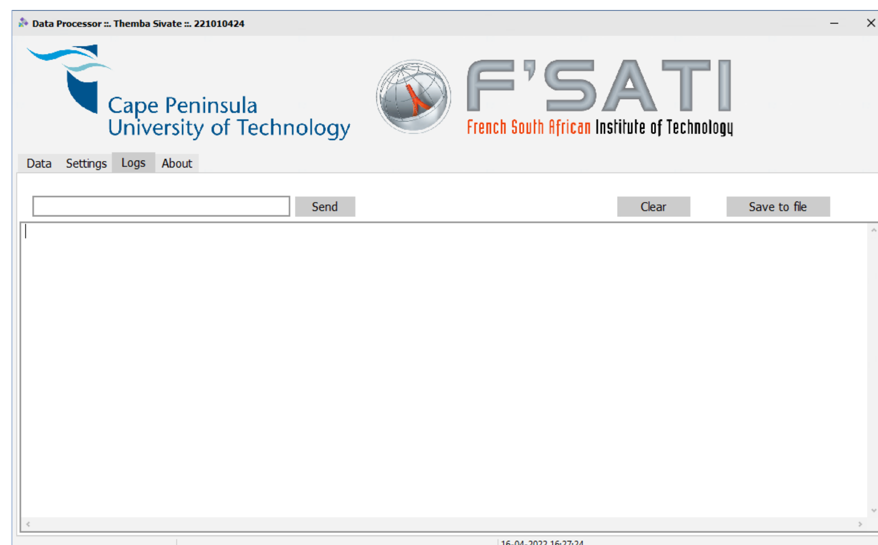

Figure 4.12: Sender software – logs tab

4.9 Output of the receiving software

Figure 4.13 shows the output of the receiving software which is responsible for acting as a ground station. It has the option to select the device to connect to. The baud rate is pre-selected to 9600bps. The connect button is used to connect the device and begin data transfer. A test group box is also provided to send a test message. An open area is provided to display any data received from the device. This software also generates and saves logs in a .csv file format. Once the software is connected to the hardware via the COM port, it will listen to any incoming data and determine if such data is a valid payload or not. Please see Appendix B for a full source code of this program.

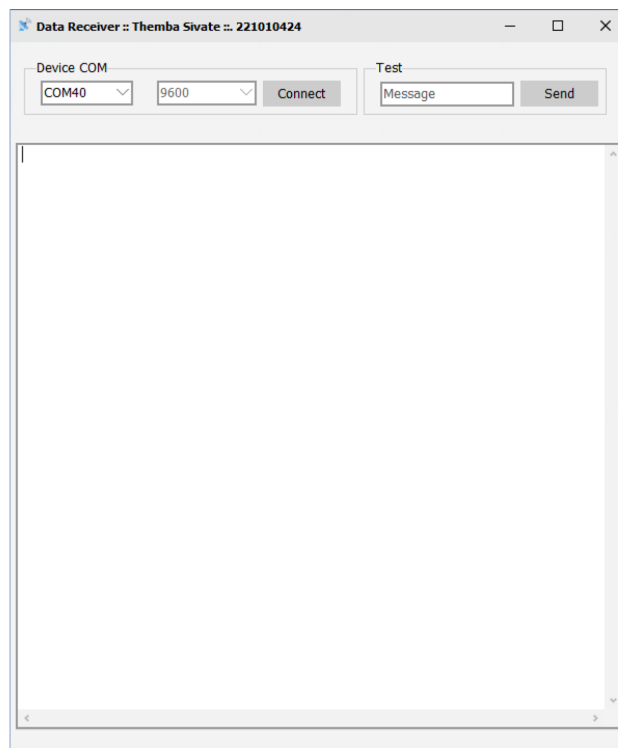

Figure 4.13: Receiver software output (DataReceiver.exe)

4.10 Summary

Both sending and receiving software were implemented in this chapter and the outputs of both programs were shown. Both programs are running as expected on windows operating system. The next chapter will show both programs being used to simulate the scenario and the results will be presented.

CHAPTER 5
TESTING AND RESULTS

5.1 Introduction

In this chapter, the implemented solution is tested, and the results are being provided and discussed. The screenshots of the cases will be presented too.

5.2 Software and hardware testing

The nRF24L01+ transceiver is connected to the nRF24L01+ USB adapter. The nRF24L01+ USB adapter is then connected to a computer running Windows. The computer allocates a COM port number for the connected device, in this case, COM8 as shown in Figure 5.1.

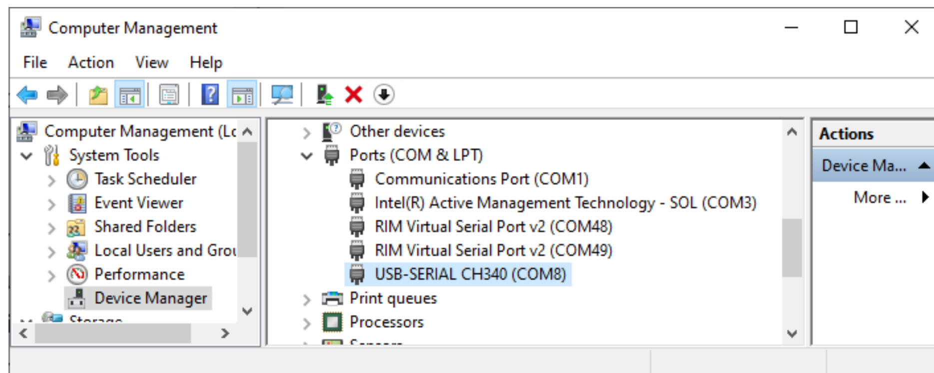

Figure 5.1: Sending device is detected and COM is allocated

The sending software imitating the satellite (DataProcessor.exe) is launched. The correct COM port is selected as shown in Figure 5.2. The baud rate is left to default (9600) and the ‘Connect’ button is clicked. Once connected, the caption of the button will change to ‘Disconnect’. At this stage, the system is ready to send data to the ground station.

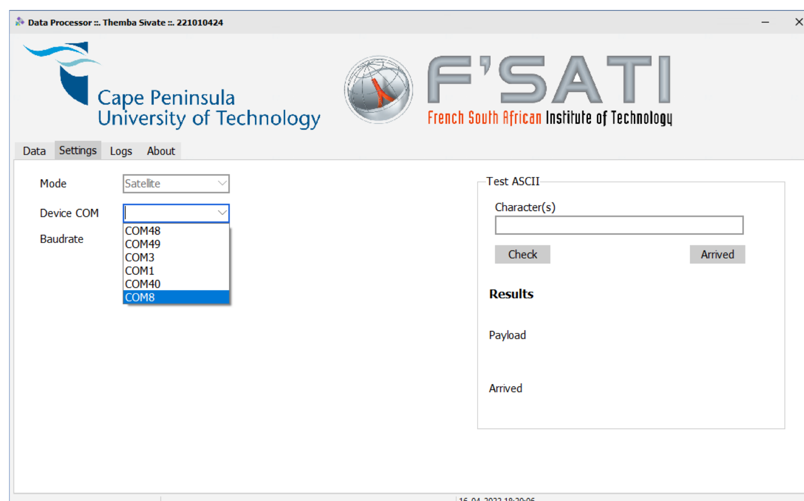

Figure 5.2: Sender software selects sending device

Switching to the ‘Logs’ tab to send a test command that can verify that the software is now connected to the hardware. The command ‘AT?’ is entered into the text box followed by a click of the ‘Send’ button. Figure 5.3 illustrates this action of the connectivity test. A response code ‘OK’ is returned by the nRF24L01+ transceiver followed by the device ID, and then the frequency of 2.400GHz is displayed. The language showing this information is not English and that is the nature of the device and its default settings. These language settings do not affect the operation of the device.

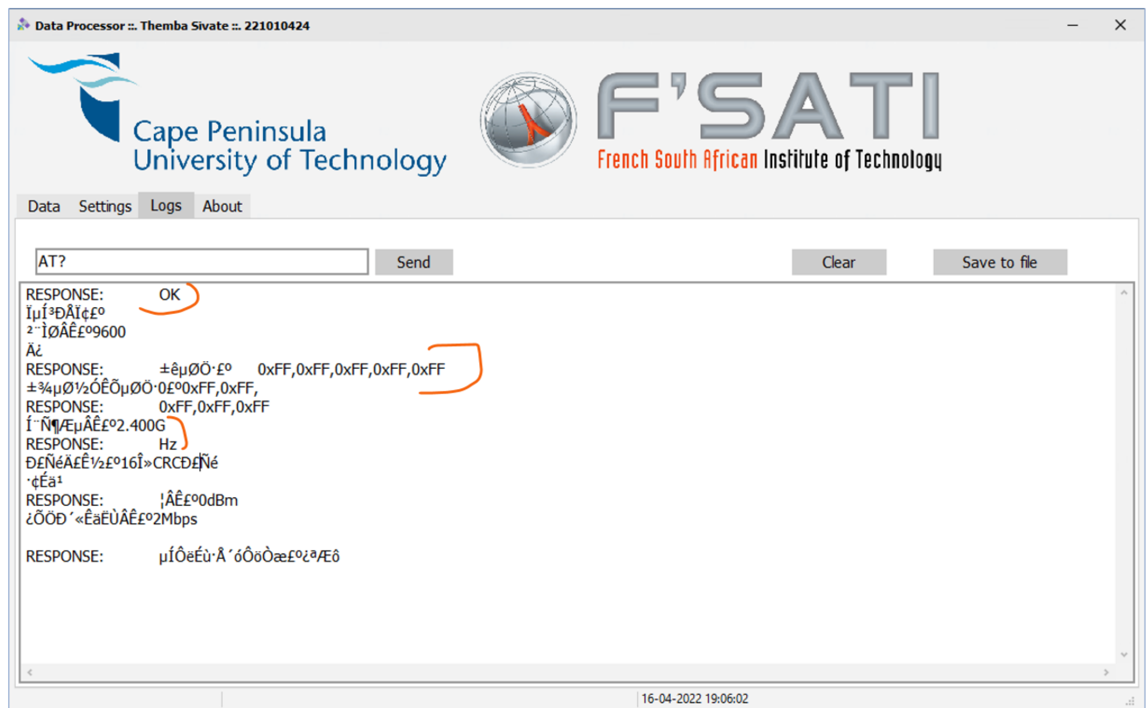

Figure 5.3: Sender software connectivity test

Switching to the Settings tab to perform the actual simulation. The ‘Load Record(s)’ button is clicked which looks for a specific file on the hard drive containing the payloads to send to the ground station. The file’s content is then displayed on the list as shown in Figure 5.4. Three rows are shown on the list view indicating the status of the payload if it was ‘Sent,’ ‘Verified’ or ‘Marked Erase’. All three indicators are defaulted to the value ‘NO’. The payload column is displaying all the payloads (AIVDM sentences) which were retrieved from the sample file. The data loss radio group defaulted to ‘None’ indicating that no data loss. The current date and time are displayed at the bottom of the list. The total number of payloads is also shown at the bottom of the list. The payloads are not signed yet. The signing of the payload is done when the connection is established to the ground station and is done per record. The developer of the firmware or the system has the option to either sign all the records before being stored

into the drive or sign each record before sending it. In this project, it was decided to sign each record before sending it.

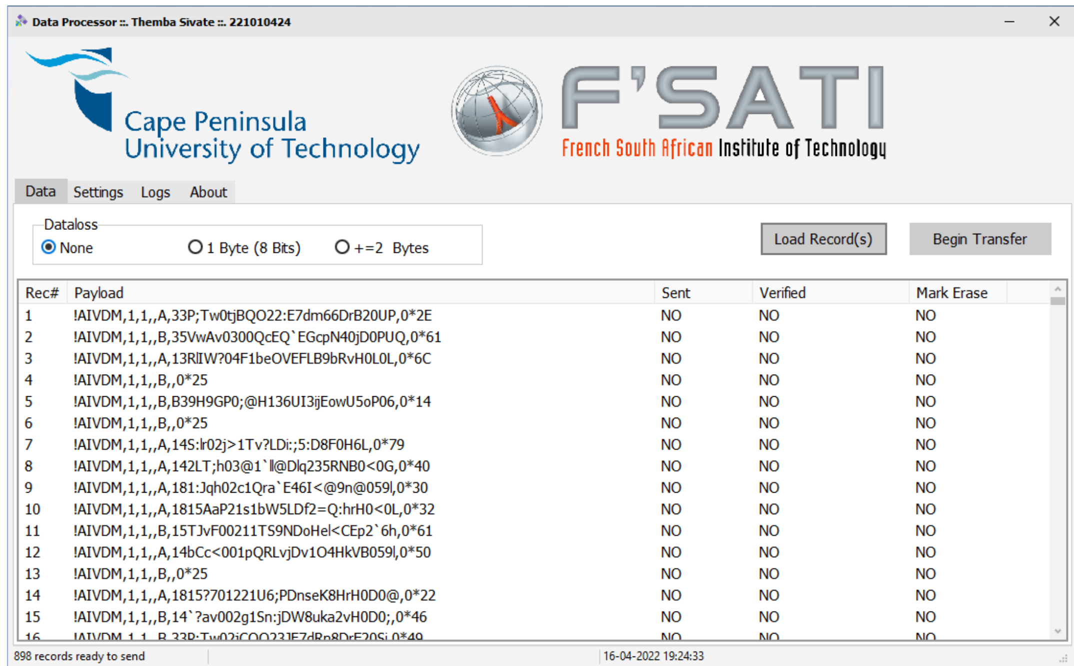

Figure 5.4: Data loaded and prepared for sending

At this stage, the sender software (satellite) is ready to send data. Because the ground station software is not running yet, the sender will not get any feedback. This is because, even if the satellite is ready to send data, the ground station needs to be visible, and the communication link needs to be established. The second nRF24L01+ transceiver is connected to the USB adapter which is then connected to the ground station's computer. Figure 5.5 shows the receiving device detected by the computer and the COM port number 5 is allocated to the device. In this simulation, the solution is tested using both single-computer and multi-computer modes. Single computer mode allows both devices to be connected to one computer for simulation. Multi-computer mode allows each device to be connected to its dedicated computer for simulation. In both modes, data is transferred wirelessly between both devices at the baud rate negotiated.

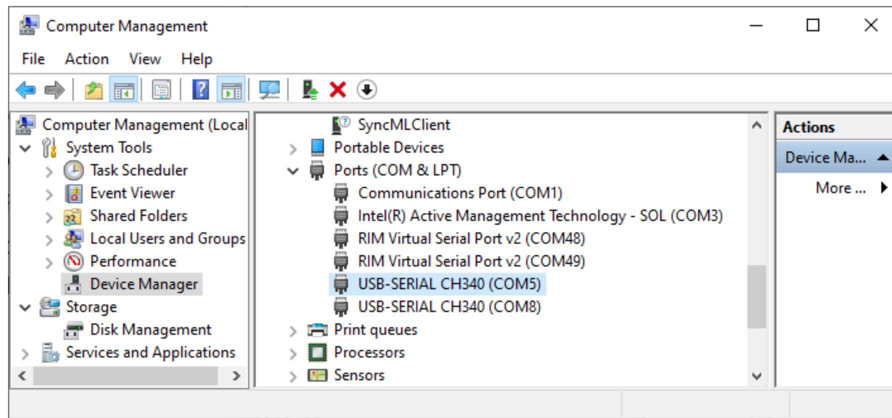

Figure 5.5: Receiving device is detected and COM is allocated

The receiving software imitating the ground station (DataReceiver.exe) is launched. The correct COM port is selected as shown in Figure 5.6. The baud rate is left to default (9600) and the 'Connect' button is clicked to connect to the transceiver. Once connected, the 'Connect' button will change its caption to 'Disconnect'.

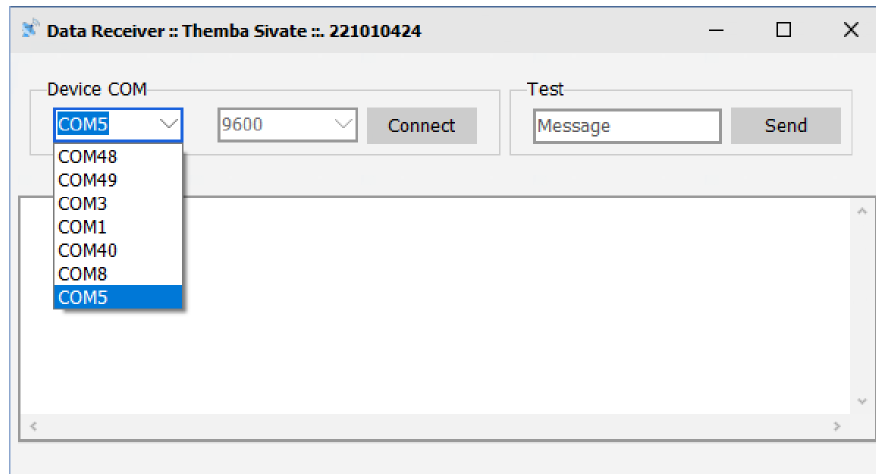

Figure 5.6: Receiver software selects receiving device

The test group box is used to input data or commands to send to the device. Figure 5.7 shows a connectivity test where a command 'AT?' was sent and the response 'OK' was returned by the device together with the ID of the device and the frequency of 2.400GHz.

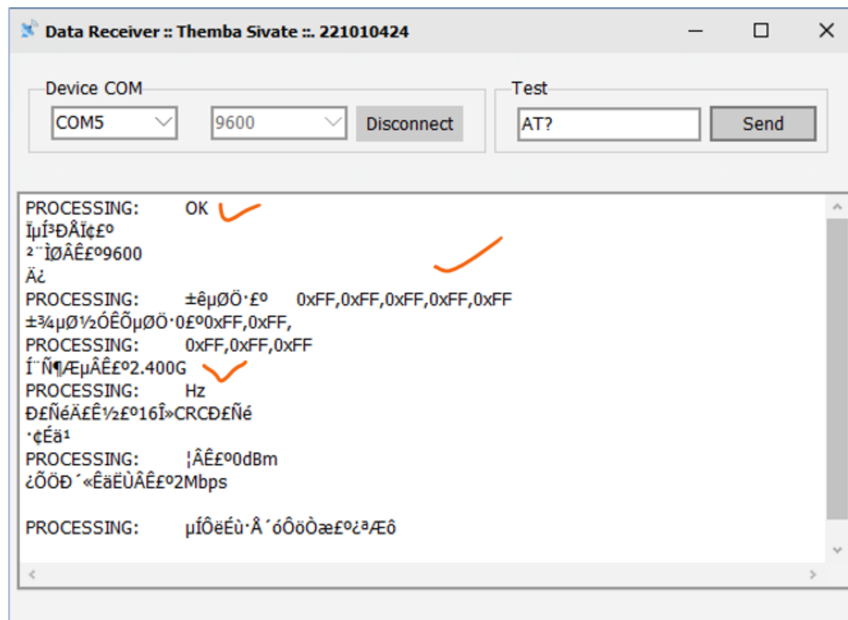

Figure 5.7: Receiver software connectivity test

5.3 Algorithm testing

Both the satellite and the ground station programs are ready to send and receive data. Clicking the 'Begin Transfer' on the sender software initiates the process of sending the payloads. Figure 5.8 shows the payload being sent by the sender software and validated by the receiver software, which returns the flag after validation. Because the data loss is set to 'None', the test passed.

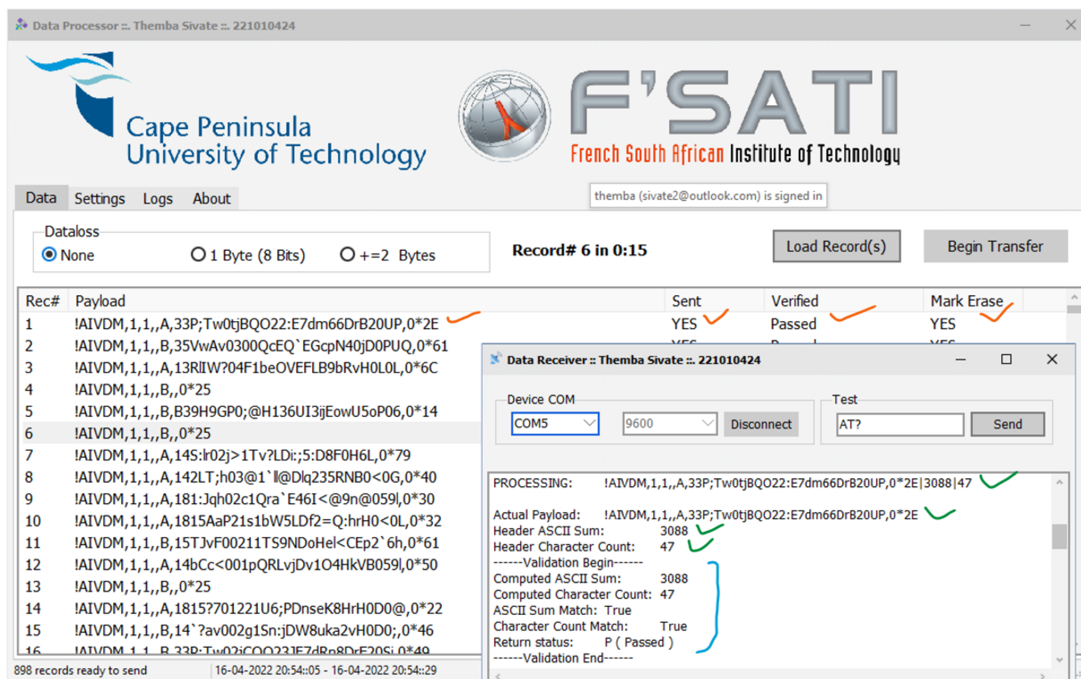

Figure 5.8: Simulation test passed

Figure 5.9 shows the received flags for the first five payloads in the Logs tab. This P flag was used to update the 'Verified' column as '*Passed'* and the 'Mark Erase' column as 'YES' as shown in Figure 5.8.

Figure 5.9: Received response flag P

A test was also performed where the last character of the payload was lost. The algorithm can detect a lost character and recover it by computing and retrieving it from the ASCII table. Figure 5.10 illustrates this scenario where the character '!' was recovered after it got chopped during data transfer.

Figure 5.10: Lost character reconstructed

A third test was performed where more than one characters were lost during data transfer and the system was able to send the F flag as a response indicating that the payload had errors that were unable to recover. Figure 5.11 illustrates this scenario.

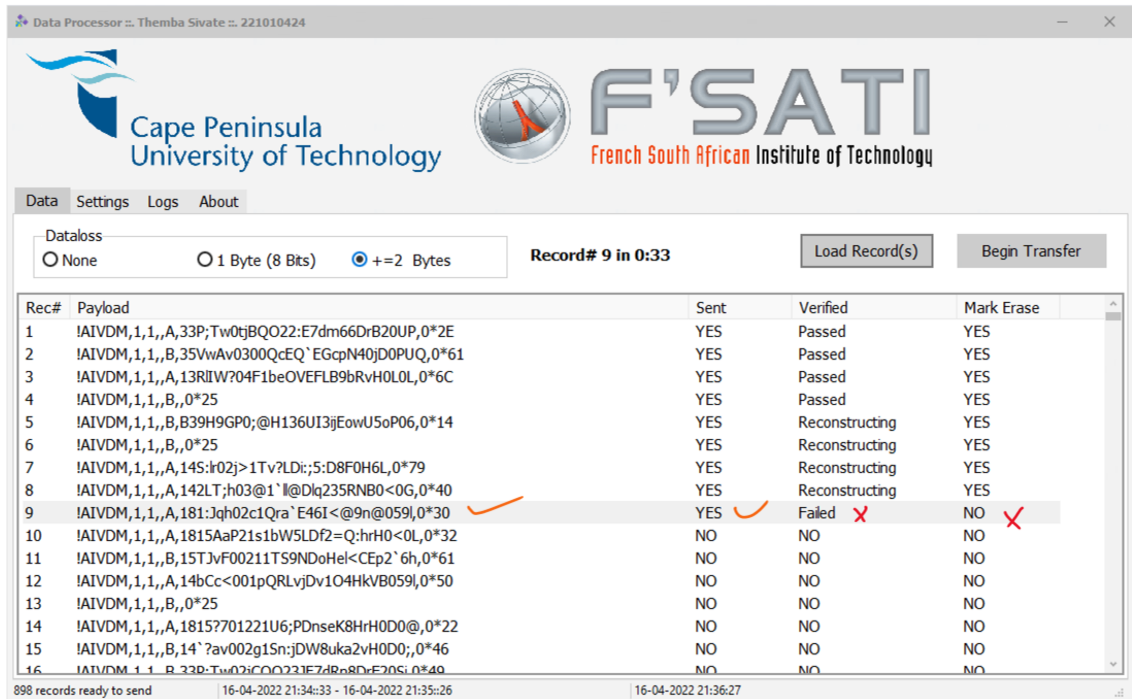

Figure 5.11: Errors detected on payload and cannot be reconstructed

An interesting fact about test case number three is that the satellite attempts to send the payload again but keeps failing. This can be noted in the receiving software displayed in Figure 5.12. This is important as it ensures that correct data needs to arrive at the ground station and is validated before it gets erased on the onboard storage of the satellite.

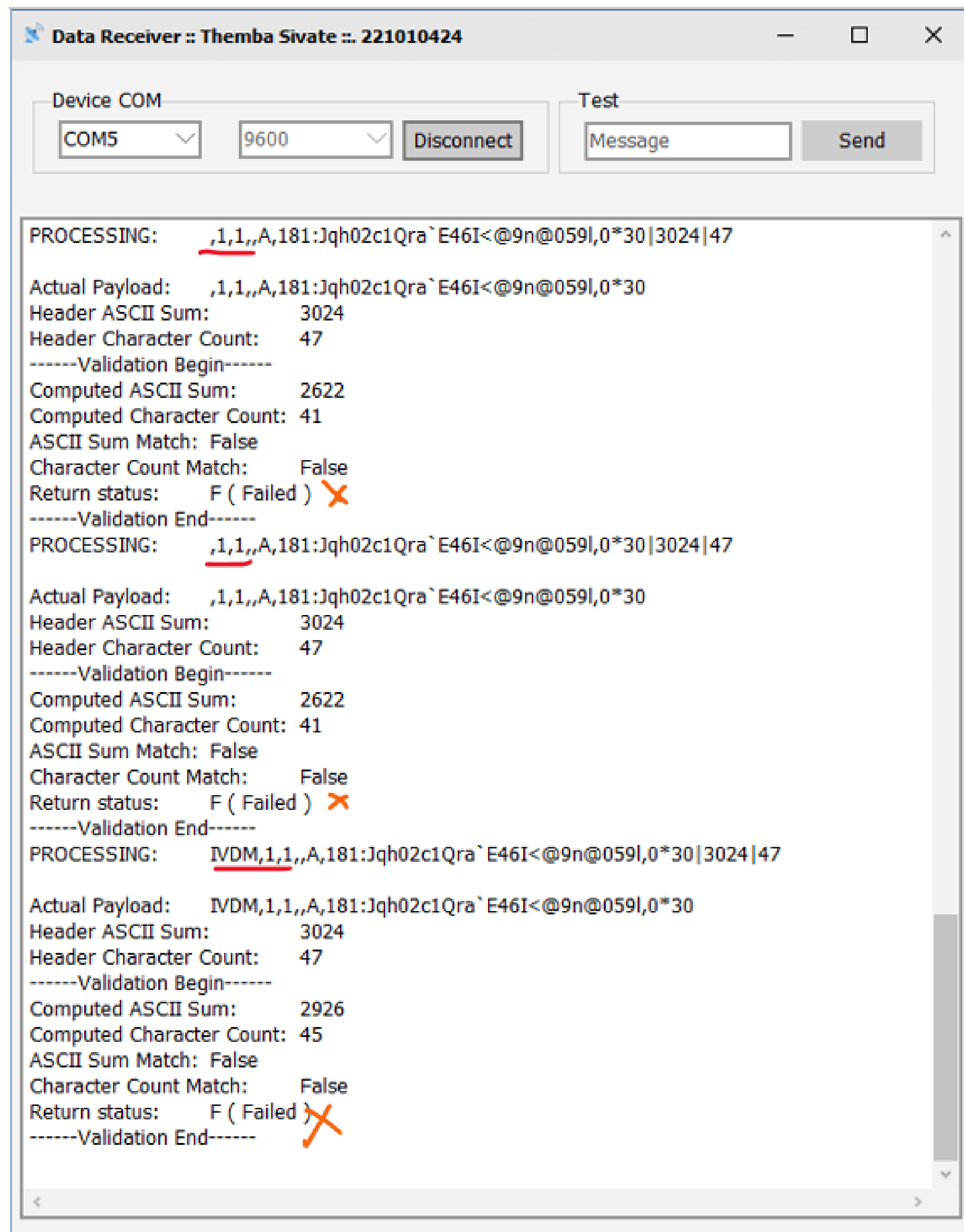

Figure 5.12: Multiple attempts on erroneous payload

A final test was performed where the actual physical device was disconnected while data was transferred from the satellite to the ground station. Since no response coming from the ground station, the payload remains on board, the verified status remains as NO, the Mark Erase status also remains as NO while the sent status changed to YES. This indicates that the satellites do not erase the content as no confirmation indicating it has arrived successfully at the ground station.

5.4 Results

Table 5.1 show the test results of the simulation

Table 5.1: Results

Test #	Description	Results
1	The sending algorithm computes and signs the payload.	Test Passed
2	The receiving algorithm unpacks and validates both the signature and the payload.	Test Passed
3	Sending software communicates with the hardware and performs data transfer.	Test Passed
4	Receiving software communicates with the hardware and performs data transfer.	Test Passed
5	Simulation of sending a complete payload received a flag P indicating a Passed.	Test Passed
6	Simulation of sending a single chopped character payload received a flag R indicating Reconstructing.	Test Passed
7	Simulation of sending an erroneous payload received a flag F response indicating Failed.	Test Passed
8	Cutting the communication link during data transfer results in no response	Test Passed

5.5 Summary

In this chapter, we have conducted tests on both the algorithm and the programs that were developed for simulation purposes. We have tested both the sending and receiving algorithms and they have been proven to be working as proposed. Similarly, the sending and receiving programs that were developed for simulation have also been tested and found to be working as expected. Based on the results, we have concluded that it is possible to use ASCII for signing and verifying text data that is sent between devices. Furthermore, we have also concluded that ASCII can be used for error detection and correction in wireless data transmission. The next chapter will provide the concluding remarks for this research.

CHAPTER 6
CONCLUSION AND FUTURE STUDIES

6.1 Conclusion

The purpose of this study was to address the issue faced by satellite operators in detecting errors during data transfer and providing error correction for data sent from the satellite to the ground station. To achieve this, an algorithm was developed using ASCII to detect errors in payloads sent by the satellite. Additionally, another algorithm was created to provide 8-bit error correction. Two Windows applications were developed to test the effectiveness of both algorithms. To simulate data transfer, one system acted as the satellite while another system acted as the ground station. The data was transferred wirelessly between both systems at a baud rate of 9600bps using the nRF24L01+ transceiver.

The algorithm was successful in detecting errors that occurred during data transfer, and it was also able to perform error correction of up to 8 bits. Based on these results, it was concluded that ASCII can be used to sign and verify packets sent over a network. Furthermore, it was concluded that any application can be developed to utilize this algorithm to detect and correct errors in data sent over radio frequency. Because ASCII is a character encoding standard used worldwide for electronic communication, it was recommended that the algorithm be implemented and compiled as firmware to run on a programmable microcontroller, specifically for bare metal firmware development.

It has been observed that, due to the algorithm's design, an extra piece of data must be included as a header in the payload. This causes the payload size to increase, resulting in longer transit times. This is because larger payloads take more time to transfer.

6.2 Future studies

In practical scenarios, a satellite uses a firmware which is not the same as a regular operating system like Windows. The purpose of a firmware is to carry out specific tasks with the aim of minimizing power usage. In the future, researchers plan to develop the algorithm on a firmware level without an operating system or graphical user interface for interaction. This means to create a firmware that can run on a programmable microcontroller and still have the ability to detect and correct errors. Additionally, future studies will involve experimenting with different baud rates, transceivers, and hardware.

REFERENCES

Aiswarya, S. & George, A. 2017. Fixed latency serial transceiver with single bit error correction on FPGA. International Conference on Trends in Electronics and Informatics (ICEI) : 902-907

Santos, M.S. &. d'Amore, R. 2018. Error detection method for the ARINC429 communication. 2018 IEEE 19th Latin-American Test Symposium (LATS). 1-6. doi: 10.1109/LATW.2018.8349687.

Hillier, C. & Balyan, V. 2019. Effect of Space Radiation on LEO Nanosatellites. Journal of Engineering and Applied Sciences, 14: 6843-6857.

Hillier, C. & Balyan, V. 2021. Review Paper: Error Detection and Correction onboard Nanosatellites. Advances in Intelligent Systems and Computing, Volume. 1334.

Leopold, L. N., Bayendang, N. P. & Balyan V. "A Nanosatellite Receiver Downconverter C-Band to L-Band Bandpass Filters—Review and Design," 2023 International Conference on Electrical, Computer and Energy Technologies (ICECET), Cape Town, South Africa, 2023, pp. 1-7, doi: 10.1109/ICECET58911.2023.10389197.

Owoade, A.A. & Ogundile, O.O. & Babalola, P.O. & Balyan, V. 2023. Efficient hybrid enhanced genetic algorithm and ant colony system model for rerouting multimedia message in multiple node–link failures within wireless network. International Journal of Communication Systems, Volume. 36.

Saleh, A.H., Abed, W.N.A. & Taha, A.M. 2018. A New Combination Method to Error Detection and Correction Using VHDL. Journal of Engineering and Applied Sciences. 13: 6167-6172

Millan, R.M., Steiger, R., Ariel, M., Bartalev, S., Campagnola, S., Castillo-Rogez, J., Mitrych, J., Musil, V. & Vlcek, K. 2003. The VHDL Model of Error Detection and Correction Subsystem for High Reliable Microcomputer Control. IFAC Proceedings Volumes. 36: 1474-6670

Sharma, S., Basnet, S. & Khanal, R. 2022. Implementation of Error Correction on IBM Quantum Computing Devices. Journal of Nepal Physical Society. 8. 7-15. 10.3126/jnphyssoc.v8i1.48278.

Lin, K. & Chen, F. 2023. Implementation of iterative error detection and correction for BAN transceiver systems. Wireless Networks. 1-16. 10.1007/s11276-022-03222-3.

Rihab, A. 2022. Checksum Error Detection. 10.13140/RG.2.2.19235.94247.

Hüsrev, C., Salonik, R., Zamshed, C., Masoud, Z., Yang, L., Brandon, Z., Jian-Ping, W., Sachin, S. & Karpuzcu U.R. 2022. Error Detection and Correction for Processing in Memory (PiM). 10.48550/arXiv.2207.13261.

Park, T., Kim, Y., Shin, D., Lee, B. & Hwang, C. 2023. Efficient Method for Error Detection and Correction in In-Memory Computing Based on Reliable Ex-Logic Gates. Advanced Intelligent Systems. 10.1002/aisy.202200341.

Zhikang, Z., Bruno, T., Michael, G., Zifan, Y., Christopher, P. & Fengbo, R. 2022. Automatic Error Detection in Integrated Circuits Image Segmentation: A Data-driven Approach. 10.48550/arXiv.2211.03927.

Sudha, K. & Laxminarayan, G. 2022. Study of Different Types of Error Detection and Correction Code in Wireless Communication. International Journal of Scientific Research in Science, Engineering and Technology. 448-455. 10.32628/IJSRSET2293138.

Rao, V.S., Schilling, K., Stephens, G., Title, M.A. & Wu, J. 2019. Small satellites for space science: A COSPAR scientific roadmap, Advances in Space Research. 64: 1466-1517

Smith, K., Crisp, N., & Hollingsworth, P. (2014). Launch and Deployment of Distributed Small Satellite Systems. In Proceedings of the International Astronautical Congress (IAC) International Astronautical Federation, IAF.

Fauzi, A. & Rahim, R. 2017. Bit Error Detection and Correction with Hamming Code Algorithm.

Fitriani, W. & Siahaan, A. P. U. "Single-Bit Parity Detection and Correction using Hamming Code 7-Bit Model", International Journal of Computer Applications, vol. 154, no. 2, pp. 12-16, 2016.

Deepika, S.S., Kumar, A.,& Gurusiddayya, H. "A Study on Error Coding Techniques," International Journal for Research in Applied Science & Engineering Technology (IJRASET), vol. 4, no. 4, pp. 825-828, 2016.

Hamming , R. "Error Detecting and Error Correcting Codes", The Bell System Technical Journal, vol. 29, no. 2, 1950.

2005. *The National Strategy for Maritime Security*. [ebook] Available at: <https://www.hsdl.org/?view&did=456414> [Accessed 02 February 2022].

2022. *Cubesat VHF/UHF Transceiver*. [online] Available at: <https://www.cput.ac.za/preview/research2/innovations/cubesat-vhf-uhf-transceiver> [Accessed 19 April 2022].

defenceWeb. 2022. *SA's ZACube-2 nanosatellite transmits first images from space - defenceWeb*. [online] Available at: <https://www.defenceweb.co.za/aerospace/aerospace-aerospace/sas-zacube-2-nanosatellite-transmits-first-images-from-space/> [Accessed 18 April 2022].

Dembski, W. 2022. *Algorithm | Encyclopedia.com*. [online] Encyclopedia.com. Available at: <https://www.encyclopedia.com/education/encyclopedias-almanacs-transcripts-and-maps/algorithm> [Accessed 18 March 2022].

En.m.wikipedia.org. 2022*. File:ASCII-Table-wide.svg - Wikipedia*. [online] Available at: <https://en.m.wikipedia.org/wiki/File:ASCII-Table-wide.svg> [Accessed 18 February 2022].

Encyclopedia.com. 2022. *Computer Program | Encyclopedia.com*. [online] Available at: <https://www.encyclopedia.com/humanities/dictionaries-thesauruses-pictures-and-press-releases/program-0> [Accessed 01 April 2022].

Encyclopedia.com. 2022. *Ground Infrastructure | Encyclopedia.com*. [online] Available at: <https://www.encyclopedia.com/science/news-wires-white-papers-and-books/ground-infrastructure> [Accessed 13 April 2022].

Encyclopedia.com. 2022. *transceiver | Encyclopedia.com*. [online] Available at: <https://www.encyclopedia.com/computing/dictionaries-thesauruses-pictures-and-press-releases/transceiver> [Accessed 07 April 2022].

Encyclopedia.com. 2022. *transponder | Encyclopedia.com*. [online] Available at: <https://www.encyclopedia.com/humanities/dictionaries-thesauruses-pictures-and-press-releases/transponder> [Accessed 18 April 2022].

Encyclopedia.com. 2022. *Wireless Technology | Encyclopedia.com*. [online] Available at: <https://www.encyclopedia.com/computing/news-wires-white-papers-and-books/wireless-technology> [Accessed 18 April 2022].

Farheen , N. & Pande , K.S. 2020. *Error Detection and Correction Using RP SEC-DED. International Conference on Electronics, Materials Engineering & Nano-Technology (IEMENTech)*: 1-6

Fléron, R., Gass, V., Gregorio, A., Klumpar, D.M., Lal, B., Macdonald, M., Park, J.U., *High accuracy calculation for life or science. 2022. Orbit of a satellite Calculator.* [online] Available at: <https://keisan.casio.com/exec/system/1224665242> [Accessed 18 April 2022].

Imo.org. 2022. *AIS transponders*. [online] Available at: <https://www.imo.org/en/OurWork/Safety/Pages/AIS.aspx> [Accessed 18 April 2022].

Kuntal, C., 2022. *What is Firmware? - Definition from Techopedia*. [online] Techopedia.com. Available at: <https://www.techopedia.com/definition/2137/firmware> [Accessed 18 April 2022].

Learn.sparkfun.com. 2022. *How to Install CH340 Drivers - learn.sparkfun.com*. [online] Available at: <https://learn.sparkfun.com/tutorials/how-to-install-ch340-drivers/windows-710> [Accessed 18 April 2022].

Makerfabs. 2022. *USB Adapter for NRF24L01+*. [online] Available at: <https://www.makerfabs.com/usb-adapter-for-nrf24l01.html?search=MCI24L01P> [Accessed 18 April 2022].

Pole Star. 2020. *An Introduction to Maritime Domain Awareness (MDA)*. [online] Available at: <https://www.polestarglobal.com/resources/an-introduction-to-maritime-domain-awareness-mda> [Accessed 17 January 2022].

Raymond, E. 2021. *NMEA 0183 Sentences*. [online] Opencpn.org. Available at: <https://opencpn.org/wiki/dokuwiki/doku.php?id=opencpn:opencpn_user_manual:advanced_features:nmea_sentences> [Accessed 13 April 2022].

Smith, D., 2022. *Software | Encyclopedia.com*. [online] Encyclopedia.com. Available at: <https://www.encyclopedia.com/finance/finance-and-accounting-magazines/software> [Accessed 18 April 2022].

Wiki.iteadstudio.com. 2022. *NRF24L01 Module - ITEAD Wiki.* [online] Available at: <https://wiki.iteadstudio.com/NRF24L01_Module> [Accessed 18 April 2022].

Yang, C. 2009. *nRF24L01+ Single Chip 2.4GHz Transceiver*. [online] Available at: <https://moxa.com.cn/getmedia/16e4924f-f5c2-4103-a133-5ef72c4f7360/moxa-the-secrets-of-uart-fifo-tech-note-v1.0.pdf> [Accessed 20 March 2022].

APPENDIX A
SENDER SOFTWARE SOURCE CODE

Program: DataProcessor
File: Main_U.h

```
//---------------------------------------------------------------------------

#ifndef Main_UH
#define Main_UH
//---------------------------------------------------------------------------
#include <System.Classes.hpp>
#include <Vcl.Controls.hpp>
#include <Vcl.StdCtrls.hpp>
#include <Vcl.Forms.hpp>
#include "VaClasses.hpp"
#include "VaComm.hpp"
#include <Vcl.ComCtrls.hpp>
#include <Vcl.ExtCtrls.hpp>
#include <Vcl.Imaging.pngimage.hpp>
//---------------------------------------------------------------------------
class TfrmMain : public TForm
{
__published:	// IDE-managed Components
	TImage *Image1;
	TPageControl *PageControl1;
	TTabSheet *TabSheet1;
	TTabSheet *TabSheet4;
	TLabel *Label1;
	TLabel *Label2;
	TLabel *Label5;
	TComboBox *cmbCOM;
	TComboBox *cmbMode;
	TComboBox *cmbBaudrate;
	TTabSheet *TabSheet2;
	TTabSheet *TabSheet3;
	TVaComm *com;
	TButton *btnConnect;
	TMemo *mmLogs;
	TPanel *Panel1;
	TButton *btnClearLogs;
	TButton *btnSaveLogs;
	TEdit *edtCommand;
	TButton *Button2;
	TPanel *Panel2;
	TListView *lstData;
	TButton *btnLoadRecords;
	TButton *btnBeginTransfer;
	TStatusBar *stbMain;
	TTimer *Timer1;
	TTimer *tmrTransfer;
	TGroupBox *GroupBox1;
	TLabeledEdit *edtCharacters;
	TButton *btnCheck;
	TLabel *lblResults;
	TLabel *Label3;
	TLabel *lblPayload;
```

```
        TButton *btnArrived;
        TLabel *Label4;
        TLabel *lblArrived;
        TTimer *tmrCounter;
        TLabel *lblTime;
        TRadioGroup *rdgDataLoss;
        TLabel *Label6;
        TLabel *Label7;
        TLabel *Label8;
        TLabel *Label9;
        void __fastcall FormShow(TObject *Sender);
        void __fastcall btnClearLogsClick(TObject *Sender);
        void __fastcall comRxChar(TObject *Sender, int Count);
        void __fastcall btnConnectClick(TObject *Sender);
        void __fastcall comRxFlag(TObject *Sender);
        void __fastcall comRx80Full(TObject *Sender);
        void __fastcall Button2Click(TObject *Sender);
        void __fastcall btnLoadRecordsClick(TObject *Sender);
        void __fastcall Timer1Timer(TObject *Sender);
        void __fastcall btnBeginTransferClick(TObject *Sender);
        void __fastcall btnCheckClick(TObject *Sender);
        void __fastcall btnArrivedClick(TObject *Sender);
        void __fastcall tmrTransferTimer(TObject *Sender);
        void __fastcall tmrCounterTimer(TObject *Sender);
        void __fastcall btnSaveLogsClick(TObject *Sender);
private:        // User declarations
public:         // User declarations
        __fastcall TfrmMain(TComponent* Owner);
        TStringList* coms;
        TStringList* lstRecords;
        TListItem *itm;
        int get_ASCII_sum(String data);
        int getDataLength(String data);
        TStringList* mydata_ready;
        String newDataReceived;
        void processResponse(String data);
        bool continue_sending;
        String finalLine,startTime;
        void sendNextData();
   int secCounter;

};
//---------------------------------------------------------------------------
extern PACKAGE TfrmMain *frmMain;
//---------------------------------------------------------------------------
#endif
```

File: Main_U.cpp

```
//---------------------------------------------------------------------------

#include <vcl.h>
#pragma hdrstop

#include "Main_U.h"
//---------------------------------------------------------------------------
#pragma package(smart_init)
```

```
#pragma link "VaClasses"
#pragma link "VaComm"
#pragma resource "*.dfm"
TfrmMain *frmMain;
//---------------------------------------------------------------------------
__fastcall TfrmMain::TfrmMain(TComponent* Owner)
        : TForm(Owner)
{
  coms=new TStringList();
  coms->Clear();

  lstRecords=new TStringList();
  lstRecords->Clear();

  mydata_ready=new TStringList();
  mydata_ready->Clear();

  newDataReceived="";
  continue_sending=false;
}
//---------------------------------------------------------------------------
void __fastcall TfrmMain::FormShow(TObject *Sender)
{
  com->GetComPortNames(coms);
  cmbCOM->Items->AddStrings(coms);
  continue_sending=false;
}
//---------------------------------------------------------------------------

void __fastcall TfrmMain::btnClearLogsClick(TObject *Sender)
{

        if(MessageDlg("You are about to clear all the logs. Do you want to
continue?",mtWarning,TMsgDlgButtons()<<mbYes<<mbNo,0)==mrYes)
        {
          mmLogs->Lines->Clear();
   }
}
//---------------------------------------------------------------------------

void __fastcall TfrmMain::comRxChar(TObject *Sender, int Count)
{
  newDataReceived=newDataReceived+com->ReadText();
  if(newDataReceived.Pos("\r\n")>0)
  {
                mmLogs->Lines->Add("RESPONSE: \t"+newDataReceived);
                processResponse(newDataReceived);

                newDataReceived="";
  }
 // mmLogs->Lines->Text=mmLogs->Lines->Text+com->ReadText();
}
//---------------------------------------------------------------------------
 void TfrmMain::processResponse(String data)
 {
        // process reponse
        if(data.Trim().Length()==1)
```

```
          {
            if(data.Trim()=="P")
            {
                    // passed
                    lstData->Items->Item[lstData->ItemIndex]->SubItems->Strings[2]="Passed";
                    lstData->Items->Item[lstData->ItemIndex]->SubItems->Strings[3]="YES";

                    if(lstData->ItemIndex<lstData->Items->Count-1)
                    {
                            lstData->ItemIndex=lstData->ItemIndex+1;
                            continue_sending=true;
                    }
          else
                    {
                              // End timer
                              tmrCounter->Enabled=false;
                    }
            }
            else if(data.Trim()=="F")
            {
                    // Failed
                    lstData->Items->Item[lstData->ItemIndex]->SubItems->Strings[2]="Failed";
                    lstData->Items->Item[lstData->ItemIndex]->SubItems->Strings[3]="NO";

                    lstData->ItemIndex=lstData->ItemIndex;
                    continue_sending=true;
            }
            else if(data.Trim()=="R")
            {
                    // Reconstructing
                    lstData->Items->Item[lstData->ItemIndex]->SubItems-
>Strings[2]="Reconstructing";
                    lstData->Items->Item[lstData->ItemIndex]->SubItems->Strings[3]="YES";

                    if(lstData->ItemIndex<lstData->Items->Count-1)
                    {
                            lstData->ItemIndex=lstData->ItemIndex+1;
                            continue_sending=true;
                    }
                    else
                    {
                              // End timer
                              tmrCounter->Enabled=false;
                    }
            }
   }
 }
void __fastcall TfrmMain::btnConnectClick(TObject *Sender)
{



  if(com->Active())
  {
          com->Close();
          btnConnect->Tag=5;
          btnConnect->Caption="Connect";
```

```
  }
  else
  {
     com->DeviceName=cmbCOM->Items->Strings[cmbCOM->ItemIndex];
            com->Open();
            btnConnect->Tag=10;
            btnConnect->Caption="Disconnect";

  }
}
//---------------------------------------------------------------------------

void __fastcall TfrmMain::comRxFlag(TObject *Sender)
{
          mmLogs->Lines->Add("Read Flag available");
}
//---------------------------------------------------------------------------

void __fastcall TfrmMain::comRx80Full(TObject *Sender)
{
  mmLogs->Lines->Add("Read Full");
}
//---------------------------------------------------------------------------
void __fastcall TfrmMain::Button2Click(TObject *Sender)
{
   com->WriteText(edtCommand->Text.Trim()+"\r\n");
}
//---------------------------------------------------------------------------


void __fastcall TfrmMain::btnLoadRecordsClick(TObject *Sender)
{
   lstRecords->Clear();
   lstRecords->LoadFromFile(ExtractFilePath(Application->ExeName)+"data.txt");

   lstData->Items->Clear();
   lstData->Items->BeginUpdate();
   for(int a=0;a<lstRecords->Count;a++)
   {
            itm=lstData->Items->Add();
            itm->Caption=IntToStr(a+1);
            itm->SubItems->Add(lstRecords->Strings[a]);
            itm->SubItems->Add("NO");
            itm->SubItems->Add("NO");
            itm->SubItems->Add("NO");
   }
   lstData->Items->EndUpdate();

   stbMain->Panels->Items[0]->Text= IntToStr(lstData->Items->Count)+" records ready to
send";
}
//---------------------------------------------------------------------------

void __fastcall TfrmMain::Timer1Timer(TObject *Sender)
{
          stbMain->Panels->Items[2]->Text=FormatDateTime("dd-mm-yyyy hh:nn:ss",Now());
```

```
}
//---------------------------------------------------------------------------

void __fastcall TfrmMain::btnBeginTransferClick(TObject *Sender)
{



  if(btnBeginTransfer->Tag==0)
  {
        if(lstData->ItemIndex<0)
        {
              lstData->ItemIndex=0;
        }
        continue_sending=true;
        startTime= FormatDateTime("DD-MM-YYYY HH:NN::SS",Now());
        tmrTransfer->Enabled=true;
        secCounter=0;
        tmrCounter->Enabled=true;
        btnBeginTransfer->Tag=1;

        btnBeginTransfer->Caption="Pause Transfer";
  }
  else  if(btnBeginTransfer->Tag==1)
  {
        if(lstData->ItemIndex<0)
        {
              lstData->ItemIndex=0;
        }
        continue_sending=true;
        startTime= FormatDateTime("DD-MM-YYYY HH:NN::SS",Now());
        tmrTransfer->Enabled=false;
        secCounter=0;
        tmrCounter->Enabled=false;
        btnBeginTransfer->Tag=0;
        btnBeginTransfer->Caption="Begin Transfer";
  }
}
void TfrmMain::sendNextData()
{
  finalLine= lstRecords->Strings[lstData->ItemIndex].Trim()+"|"+
                                                IntToStr(get_ASCII_sum(lstRecords-
>Strings[lstData->ItemIndex].Trim()))+"|"+
                                                IntToStr( getDataLength(lstRecords-
>Strings[lstData->ItemIndex].Trim()) );
  mydata_ready->Clear();
  while(finalLine.Length()>25)
  {
        mydata_ready->Add(finalLine.SubString(1,25));
        finalLine=finalLine.Delete(1,25);
  }
  mydata_ready->Add(finalLine);

  // Check Dataloss simulation
  if(rdgDataLoss->ItemIndex==1)  // loose 1st character
  {
        mydata_ready->Strings[0]=mydata_ready->Strings[0].Delete(1,1);
```

```
  }
  else if(rdgDataLoss->ItemIndex==2)  // loose +=2 characters
  {
     mydata_ready->Strings[0]=mydata_ready->Strings[0].Delete(1,Random(5)+2);
  }

  for(int a=0;a<mydata_ready->Count;a++)
  {
           com->WriteText(mydata_ready->Strings[a]);
           Sleep(1500);
           Application->ProcessMessages();
  }
  com->WriteText("\r\n");
  lstData->Items->Item[lstData->ItemIndex]->SubItems->Strings[1]="YES";
}
//---------------------------------------------------------------------------
int TfrmMain::get_ASCII_sum(String data)
{
         int sum=0;
         for(int a=1;a<=data.Length();a++)
         {
           sum+= int(data[a]);
         }
         return sum;
}
int TfrmMain::getDataLength(String data)
{
  return data.Trim().Length();
}
/*
  computing power exchange

  increase data

  decrease trsafer speed

  data nevr changed from source to destination
  data is complete
  recovery mode available
*/


void __fastcall TfrmMain::btnCheckClick(TObject *Sender)
{
  lblResults->Caption="Results: "+IntToStr(get_ASCII_sum(edtCharacters->Text.Trim()))+" |
"+IntToStr( getDataLength(edtCharacters->Text.Trim()) );
  lblPayload->Caption=edtCharacters->Text.Trim()+"|"+
IntToStr(get_ASCII_sum(edtCharacters->Text.Trim()))+"|"+IntToStr(
getDataLength(edtCharacters->Text.Trim()) );
}
//---------------------------------------------------------------------------

void __fastcall TfrmMain::btnArrivedClick(TObject *Sender)
{
  lblArrived->Caption=edtCharacters->Text.Trim(); //+"|"+
IntToStr(get_ASCII_sum(edtCharacters->Text.Trim()))+"|"+IntToStr(
getDataLength(edtCharacters->Text.Trim()) );
```

```
}
//---------------------------------------------------------------------------

void __fastcall TfrmMain::tmrTransferTimer(TObject *Sender)
{
            if(continue_sending)
            {
              continue_sending=false;
              sendNextData();

    }
}
//---------------------------------------------------------------------------

void __fastcall TfrmMain::tmrCounterTimer(TObject *Sender)
{
  secCounter++;
  lblTime->Caption="Record# "+IntToStr(lstData->ItemIndex+1)+" in "+IntToStr(  (int)
secCounter/60)  +":"+IntToStr(  (int) secCounter%60);
  stbMain->Panels->Items[1]->Text=startTime+" - "+FormatDateTime("DD-MM-YYYY
HH:NN::SS",Now());

  if(!com->Active())
  {
            tmrCounter->Enabled=false;
  }
}
//---------------------------------------------------------------------------

void __fastcall TfrmMain::btnSaveLogsClick(TObject *Sender)
{
          mmLogs->Lines->SaveToFile(ExtractFilePath(ParamStr(0))+"logs
"+FormatDateTime("ddmmyyyy hhmmss",Now())+".txt");
          MessageDlg("Logs saved!",mtInformation,TMsgDlgButtons()<<mbOK,0);
}
//---------------------------------------------------------------------------
```

APPENDIX B
RECEIVER SOFTWARE SOURCE CODE

Program: DataReceiver
File: Main_U.h

```
//---------------------------------------------------------------------------

#ifndef Main_UH
#define Main_UH
//---------------------------------------------------------------------------
#include <System.Classes.hpp>
#include <Vcl.Controls.hpp>
#include <Vcl.StdCtrls.hpp>
#include <Vcl.Forms.hpp>
#include <Vcl.ExtCtrls.hpp>
#include <Vcl.ComCtrls.hpp>
#include "VaClasses.hpp"
#include "VaComm.hpp"
#include <Vcl.Menus.hpp>
#include <STRUtils.hpp>
//---------------------------------------------------------------------------
class TfrmMain : public TForm
{
__published:	// IDE-managed Components
	TPanel *Panel1;
	TMemo *mmLogs;
	TGroupBox *GroupBox1;
	TButton *btnSend;
	TEdit *edtCommand;
	TGroupBox *GroupBox2;
	TButton *btnConnect;
	TComboBox *cmbCOM;
	TComboBox *ComboBox2;
	TStatusBar *StatusBar1;
	TVaComm *com;
	TPopupMenu *ppMemo;
	TMenuItem *ClearLogs1;
	TMenuItem *N1;
	void __fastcall FormShow(TObject *Sender);
	void __fastcall btnConnectClick(TObject *Sender);
	void __fastcall comRxChar(TObject *Sender, int Count);
	void __fastcall btnSendClick(TObject *Sender);
	void __fastcall ClearLogs1Click(TObject *Sender);
private:	// User declarations
public:		// User declarations
	__fastcall TfrmMain(TComponent* Owner);
	TStringList* coms;
	String newDataReceived;
	void validateData(String data);
   int get_ASCII_sum(String data);
	int getDataLength(String data);
	TStringList* logs;
	String logsLine,logsFilename;
```

```
};
//---------------------------------------------------------------------------
extern PACKAGE TfrmMain *frmMain;
//---------------------------------------------------------------------------
#endif
```

File: Main_U.cpp

```
//---------------------------------------------------------------------------

#include <vcl.h>
#pragma hdrstop

#include "Main_U.h"
//---------------------------------------------------------------------------
#pragma package(smart_init)
#pragma link "VaClasses"
#pragma link "VaComm"
#pragma resource "*.dfm"
TfrmMain *frmMain;
//---------------------------------------------------------------------------
__fastcall TfrmMain::TfrmMain(TComponent* Owner)
        : TForm(Owner)
{
  coms=new TStringList();
  coms->Clear();

  newDataReceived="";

  logs=new TStringList();
  logs->Clear();

}
//---------------------------------------------------------------------------
void __fastcall TfrmMain::FormShow(TObject *Sender)
{
  coms->Clear();
  com->GetComPortNames(coms);
  cmbCOM->Items->Clear();
  cmbCOM->Items->AddStrings(coms);

  if(cmbCOM->Items->Count>0)
  {
        cmbCOM->ItemIndex=cmbCOM->Items->Count-1;
  }
  newDataReceived="";
  logs->Clear();

  logsFilename=ExtractFilePath(ParamStr(0))+"logs "+FormatDateTime("ddmmyyyy
hhnnss",Now())+".csv";
  logsLine="DATETIME#RECEIVED DATA#ACTUAL PAYLOAD#HEADER ASCII
SUM#HEADER COUNT#COMPUTED ASCII SUM#COMPUTED COUNT#ASCII SUM
MATCH#COUNT MATCH#RESPONSE CODE#RECONSTRUCTED";
  logs->Add(logsLine);
}
//---------------------------------------------------------------------------
void __fastcall TfrmMain::btnConnectClick(TObject *Sender)
{
```

```
 /* try
  {
          com->DeviceName=cmbCOM->Items->Strings[cmbCOM->ItemIndex];
          com->Open();
           btnConnect->Caption="Disconnect";
  }
  catch(Exception &e)
  {
           btnConnect->Caption="Connect";
  }
 */
  if(com->Active())
  {
           com->Close();
           btnConnect->Tag=5;
           btnConnect->Caption="Connect";

  }
  else
  {
     com->DeviceName=cmbCOM->Items->Strings[cmbCOM->ItemIndex];
           com->Open();
           btnConnect->Tag=10;
           btnConnect->Caption="Disconnect";

  }


}
//---------------------------------------------------------------------------
void __fastcall TfrmMain::comRxChar(TObject *Sender, int Count)
{
            newDataReceived=newDataReceived+com->ReadText();

            if(newDataReceived.Pos("\r\n")>0)
            {
                        mmLogs->Lines->Add("PROCESSING: \t"+newDataReceived);
                        validateData(newDataReceived);
          // Process and send Feedback
                        newDataReceived="";
     }
          //  mmLogs->Lines->Text=mmLogs->Lines->Text+com->ReadText();
}
//---------------------------------------------------------------------------
void __fastcall TfrmMain::btnSendClick(TObject *Sender)
{
           com->WriteText(edtCommand->Text.Trim()+"\r\n");
}
//---------------------------------------------------------------------------
void __fastcall TfrmMain::ClearLogs1Click(TObject *Sender)
{
          if(MessageDlg("You are about to clear all the logs. Do you want to
continue?",mtWarning,TMsgDlgButtons()<<mbYes<<mbNo,0)==mrYes)
          {
            mmLogs->Lines->Clear();
   }
}
```

```
//--------------------------------------------------------------------------
void TfrmMain::validateData(String data)
{
  String response="F";
  if(data.Trim().Pos("|")>0 && data.Trim().LastDelimiter("|")>data.Trim().Pos("|"))
  {
        TStringDynArray validate=SplitString(data.Trim(),"|");

        logsLine=FormatDateTime("dd-mm-yyyy hh:nn:ss",Now())+"#";

        logsLine=logsLine+data+"#";
        mmLogs->Lines->Add("Actual Payload: \n\t"+validate[0]);
        mmLogs->Lines->Add("Header ASCII Sum: \t"+validate[1]);
        mmLogs->Lines->Add("Header Character Count: \t"+validate[2]);

        logsLine=logsLine+validate[0]+"#";
        logsLine=logsLine+validate[1]+"#";
        logsLine=logsLine+validate[2]+"#";
        mmLogs->Lines->Add("------Validation Begin------\n");

        String ASCII_Sum=IntToStr(get_ASCII_sum(validate[0]));
        mmLogs->Lines->Add("Computed ASCII Sum: \t"+ASCII_Sum);

        String  Character_Count=IntToStr(getDataLength(validate[0]));
        mmLogs->Lines->Add("Computed Character Count: \t"+Character_Count);

        logsLine=logsLine+ASCII_Sum+"#";
        logsLine=logsLine+Character_Count+"#";

        if( validate[1].ToInt()== get_ASCII_sum(validate[0]) )
        {
              mmLogs->Lines->Add("ASCII Sum Match: \tTrue");
              logsLine=logsLine+"TRUE#";
        }
        else
        {
               mmLogs->Lines->Add("ASCII Sum Match: \tFalse");
               logsLine=logsLine+"FALSE#";
        }

        if( validate[2].ToInt()== getDataLength(validate[0]) )
        {
              mmLogs->Lines->Add("Character Count Match: \tTrue");
              logsLine=logsLine+"TRUE#";
        }
        else
        {
               mmLogs->Lines->Add("Character Count Match: \tFalse");
               logsLine=logsLine+"FALSE#";
        }

        if( validate[1].ToInt()== get_ASCII_sum(validate[0]) &&
                    validate[2].ToInt()== getDataLength(validate[0]) )
        {
              response="P";
              logsLine=logsLine+"P#NONE";
              mmLogs->Lines->Add("Return status: \t"+response+ " ( Passed )");
```

```
                com->WriteText(response.Trim()+"\r\n");
            }
            else if( validate[1].ToInt() != get_ASCII_sum(validate[0])
                        && getDataLength(validate[0]) == validate[2].ToInt()-1  )
            {
                response="R";
                String reconstructed=String( Char( validate[1].ToInt()-

          get_ASCII_sum(validate[0]) ) )+validate[0];
                logsLine=logsLine+"R#"+ reconstructed;
                mmLogs->Lines->Add("Return status: \t"+response+ " ( Reconstructing )");
                mmLogs->Lines->Add("Reconstructed: \t"+reconstructed);

                com->WriteText(response.Trim()+"\r\n");
            }
            else
            {
                response="F";
                logsLine=logsLine+"F#NONE";
                mmLogs->Lines->Add("Return status: \t"+response+ " ( Failed )");
                com->WriteText(response.Trim()+"\r\n");
            }
            mmLogs->Lines->Add("------Validation End------\n");

            logs->Add(logsLine);
            logs->SaveToFile(logsFilename);
  }
}
int TfrmMain::get_ASCII_sum(String data)
{
          int sum=0;
          for(int a=1;a<=data.Length();a++)
          {
            sum+= int(data[a]);
          }
          return sum;
}
int TfrmMain::getDataLength(String data)
{
  return data.Trim().Length();
}
```

APPENDIX C
GROUND STATION SAMPLE LOG FILE

Sample File: logs 18042022 141141.csv

```
DATETIME#RECEIVED DATA#ACTUAL PAYLOAD#HEADER ASCII SUM#HEADER
COUNT#COMPUTED ASCII SUM#COMPUTED COUNT#ASCII SUM MATCH#COUNT
MATCH#RESPONSE CODE#RECONSTRUCTED
18-04-2022 14:12:16#!AIVDM,1,1,,A,13RIIW?04F1beOVEFLB9bRvH0L0L,0*6C|3092|47
#!AIVDM,1,1,,A,13RIIW?04F1beOVEFLB9bRvH0L0L,0*6C#3092#47#3092#47#TRUE#TRU
E#P#NONE
18-04-2022 14:12:19#!AIVDM,1,1,,B,,0*25|1023|19
#!AIVDM,1,1,,B,,0*25#1023#19#1023#19#TRUE#TRUE#P#NONE
18-04-2022 14:12:23#!AIVDM,1,1,,B,B39H9GP0;@H136UI3ijEowU5oP06,0*14|3022|47
#!AIVDM,1,1,,B,B39H9GP0;@H136UI3ijEowU5oP06,0*14#3022#47#3022#47#TRUE#TRU
E#P#NONE
18-04-2022 14:12:27#!AIVDM,1,1,,B,,0*25|1023|19
#!AIVDM,1,1,,B,,0*25#1023#19#1023#19#TRUE#TRUE#P#NONE
18-04-2022 14:12:32#!AIVDM,1,1,,A,14S:Ir02j>1Tv?LDi:;5:D8F0H6L,0*79|2996|47
#!AIVDM,1,1,,A,14S:Ir02j>1Tv?LDi:;5:D8F0H6L,0*79#2996#47#2996#47#TRUE#TRUE#P#
NONE
18-04-2022 14:12:37#!AIVDM,1,1,,A,142LT;h03@1`Il@Dlq235RNB0<0G,0*40|2977|47
#!AIVDM,1,1,,A,142LT;h03@1`Il@Dlq235RNB0<0G,0*40#2977#47#2977#47#TRUE#TRUE
#P#NONE
18-04-2022 14:12:41#AIVDM,1,1,,A,181:Jqh02c1Qra`E46I<@9n@059l,0*30|3024|47
#AIVDM,1,1,,A,181:Jqh02c1Qra`E46I<@9n@059l,0*30#3024#47#2991#46#FALSE#FALSE
#R#!AIVDM,1,1,,A,181:Jqh02c1Qra`E46I<@9n@059l,0*30
18-04-2022 14:12:46#AIVDM,1,1,,A,1815AaP21s1bW5LDf2=Q:hrH0<0L,0*32|2988|47
#AIVDM,1,1,,A,1815AaP21s1bW5LDf2=Q:hrH0<0L,0*32#2988#47#2955#46#FALSE#FALS
E#R#!AIVDM,1,1,,A,1815AaP21s1bW5LDf2=Q:hrH0<0L,0*32
18-04-2022 14:12:51#AIVDM,1,1,,B,15TJvF00211TS9NDoHel<CEp2`6h,0*61|3089|47
#AIVDM,1,1,,B,15TJvF00211TS9NDoHel<CEp2`6h,0*61#3089#47#3056#46#FALSE#FALS
E#R#!AIVDM,1,1,,B,15TJvF00211TS9NDoHel<CEp2`6h,0*61
18-04-2022 14:12:56#AIVDM,1,1,,A,14bCc<001pQRLvjDv1O4HkVB059l,0*50|3128|47
#AIVDM,1,1,,A,14bCc<001pQRLvjDv1O4HkVB059l,0*50#3128#47#3095#46#FALSE#FALS
E#R#!AIVDM,1,1,,A,14bCc<001pQRLvjDv1O4HkVB059l,0*50
18-04-2022 14:12:59#AIVDM,1,1,,B,,0*25|1023|19
#AIVDM,1,1,,B,,0*25#1023#19#990#18#FALSE#FALSE#R#!AIVDM,1,1,,B,,0*25
18-04-2022 14:13:04#AIVDM,1,1,,A,1815?701221U6;PDnseK8HrH0D0@,0*22|2879|47
#AIVDM,1,1,,A,1815?701221U6;PDnseK8HrH0D0@,0*22#2879#47#2846#46#FALSE#FAL
SE#R#!AIVDM,1,1,,A,1815?701221U6;PDnseK8HrH0D0@,0*22
18-04-2022 14:13:08#M,1,1,,B,14`?av002g1Sn:jDW8uka2vH0D0;,0*46|3151|47
#M,1,1,,B,14`?av002g1Sn:jDW8uka2vH0D0;,0*46#3151#47#2826#42#FALSE#FALSE#F#N
ONE
18-04-2022 14:13:13#IVDM,1,1,,B,14`?av002g1Sn:jDW8uka2vH0D0;,0*46|3151|47
#IVDM,1,1,,B,14`?av002g1Sn:jDW8uka2vH0D0;,0*46#3151#47#3053#45#FALSE#FALSE#
F#NONE
18-04-2022 14:13:18#DM,1,1,,B,14`?av002g1Sn:jDW8uka2vH0D0;,0*46|3151|47
#DM,1,1,,B,14`?av002g1Sn:jDW8uka2vH0D0;,0*46#3151#47#2894#43#FALSE#FALSE#F#
NONE
18-04-2022 14:13:23#DM,1,1,,B,14`?av002g1Sn:jDW8uka2vH0D0;,0*46|3151|47
#DM,1,1,,B,14`?av002g1Sn:jDW8uka2vH0D0;,0*46#3151#47#2894#43#FALSE#FALSE#F#
NONE
18-04-2022 14:13:27#VDM,1,1,,B,14`?av002g1Sn:jDW8uka2vH0D0;,0*46|3151|47
#VDM,1,1,,B,14`?av002g1Sn:jDW8uka2vH0D0;,0*46#3151#47#2980#44#FALSE#FALSE#F
#NONE
```

18-04-2022 14:13:32#DM,1,1,,B,14`?av002g1Sn:jDW8uka2vH0D0;,0*46|3151|47
#DM,1,1,,B,14`?av002g1Sn:jDW8uka2vH0D0;,0*46#3151#47#2894#43#FALSE#FALSE#F#
NONE
18-04-2022 14:13:37#DM,1,1,,B,14`?av002g1Sn:jDW8uka2vH0D0;,0*46|3151|47
#DM,1,1,,B,14`?av002g1Sn:jDW8uka2vH0D0;,0*46#3151#47#2894#43#FALSE#FALSE#F#
NONE