University of Cape Town

EEE3097S

Engineering Principles: Electrical And Computer Engineering

Paper Design

Authors:
David Young
Caide Spriestersbach

Student Numbers:
YNGDAV005
SPRCAI002

August 2022

Table of Contents

Table of Figures

List of Tables

Administrative Details

1 Individual Contributions

Both members worked equally on the project and contributed to each section. However, some sections were focussed more heavily on by a certain member. These sections are listed below.

Name	Sections	Pages
David Young YNGDAV005	1.1, 1.3, 1.5, 2.1.4, 2.1.6, 3.1, 3.2, 3.3	4, 6, 7, 8, 11, 12, 15, 16, 17
Caide Spriestersbach SPRCAI002	Intro, 1.1, 1.2, 1.4, 2.1.1, 2.1.2, 2.1.3, 2.1.5, 2.2, 2.3	3, 4, 5, 7, 9, 10, 11, 12, 13, 14

Table I – Table of individual contributions made by each member

2 Project Management Tool

Below is a screenshot of the project management tool front page as of Friday, 19th Aug.

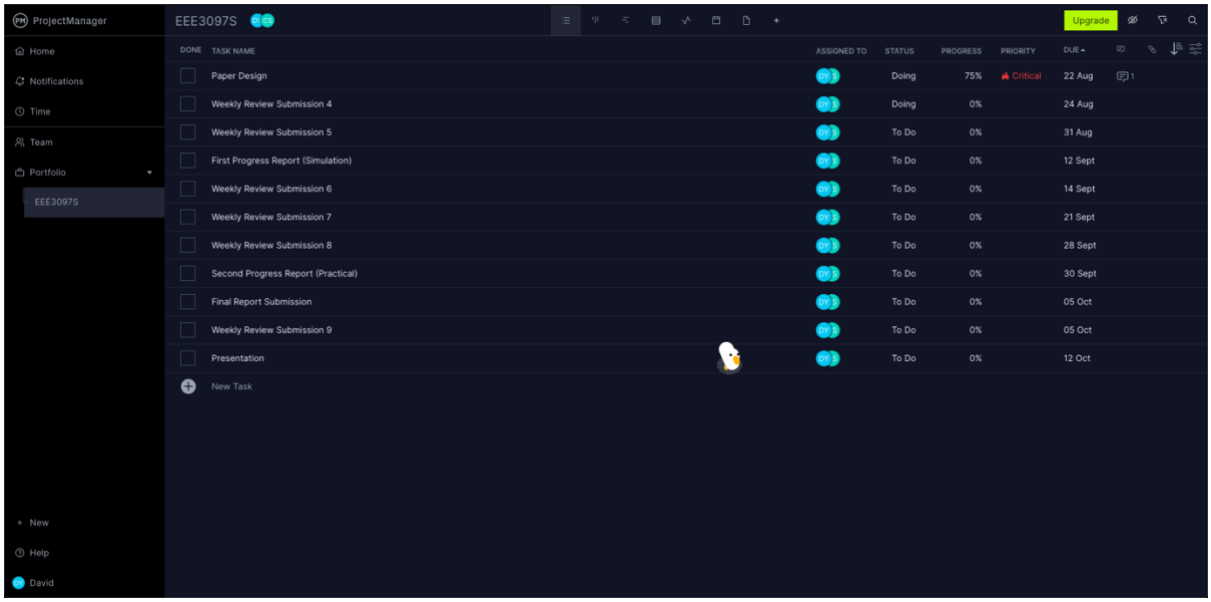

Figure 1 – Screenshot of project management tool front page

3 Development Timeline

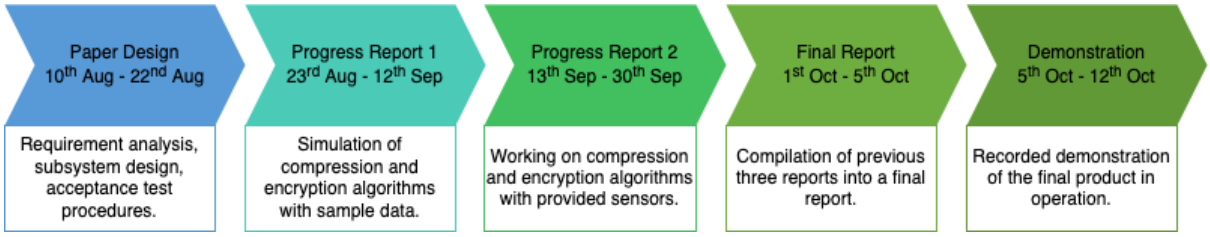

Figure 2 – Development timeline for project

Introduction

As a part of studying the movement and atmosphere of the Antarctic pancake ice, the SHARC buoy in Antarctica consists of multiple sensors and other processors. The following report will outline the requirements and design procedure for the compression and encryption of data received by the Inertial Measurement Unit (IMU) – one of the crucial sensors onboard the buoy. The data is compressed to reduce the transmission cost, as Iridium is exceptionally costly.

The compression and encryption will be done on the STM32F051 Discovery Board, which will be connected to the Sense HAT B – which houses the sensors and records the data. The STM32 Discovery Board makes use of the ARM Cortex M0 core. This will be responsible for processing, encrypting, compressing, and transmitting the relevant data packets from the IMU, which will be used to analyse the environment surrounding Antarctica.

The report will analyse the requirements outlined in the project design brief, present the subsystem and the sub-subsystem design, outline the acceptance test procedures, and the development timeline of the overall design. In doing so, the report will summarise each requirement derived from the project brief, compare various compression and encryption algorithms to determine the most suitable algorithm, analyse the feasibility of the overall project, present possible bottlenecks of the system, outline Subsystem and Sub-subsystem requirements and specifications, as well as the Inter-Subsystem and Inter-Sub-subsystems interactions, present the UML diagrams for the overall system. The acceptance test procedures will make use of figures of merit, which will be used to validate the design. The report will present and summarise the experiment design used to test the figures of merit and adumbrate the acceptable performance definition.

1. Requirement Analysis

1.1 Interpretation of the requirements

The following is extrapolated based on the requirements outlined in the design project brief. The requirements below will be used to produce the functional requirements and hence the specifications in the subsequent sections. These requirements are listed in no particular order.

User Requirement ID	Description
UR1	Data must be encrypted.
UR2	Data must be compressed.
UR3	Client requires at least 25% of the lower Fourier coefficients of the data.
UR4	Power usage must be low to increase battery life.
UR5	Software may be written in any language.
UR6	Processing load on the STM board must be low.
UR7	A motion tracking device must be used to collect the data.
UR8	The STM32F051 Discovery Board must be used.

Table II – Interpretation of user requirements

1.2 Comparison of encryption algorithms

Encryption can be broken down into two types, asymmetric and symmetric. Symmetric cryptography uses the same key for encryption and decryption [1]. A unique key is produced for each pair of participants [1]. Since symmetric encryption uses keys that are only known to the two parties making use of the encryption, this brings about a level of authentication. The benefits of using symmetric cryptography are that it is relatively inexpensive for a strong key for ciphering to be produced, the algorithms are computationally cheap, the level of protection dictates the size of the key, and the keys tend to be smaller for the level they afford [1]. The possible downfalls of symmetric encryption are that the privacy of the key must be retained to ensure confidentiality and authentication – this usually means that the key must be encrypted in another key, and the recipient must already have the key for decryption of the encrypted secret-key [1].

Asymmetric cryptography (also known as public-key encryption) uses a pair of keys, known as the public and private keys, which are associated with an individual who needs to encrypt data [1]. Each public key is published while their corresponding private keys are kept secret [1]. All data encrypted with the public key can only be decrypted using the corresponding private key [1]. The possible benefit of using an asymmetric algorithm is that the transmission of data packets will be protected in a public domain environment, i.e., the internet [1]. The

possible downfalls are that if the private key becomes lost or forgotten, the sent data cannot be decrypted, as authentication cannot be made.

The Advanced Encryption Standard (AES) is a symmetric cryptographic algorithm that can be used to protect data [2]. The AES is a block cypher, meaning it will take in data of a fixed size in bits and produce the exact size of bits of ciphertext – an unintelligible form of data [3]. The algorithm has the capacity of using keys of lengths 128, 192, and 256 bits to encrypt and decrypt blocks of data of size 128 bits. The AES is highly efficient for 128-bit form, can be implemented in hardware and software, has open source code, and is invulnerable to most attacks except brute force.

The RSA algorithm is an asymmetric cryptography algorithm. It has become the standard for encrypting and decrypting data shared via the internet. The algorithm's encryption strength depends on the key's size, and a larger key size means a more secure algorithm. The algorithm is known to be robust and reliable but also known to be computationally intensive.

Blowfish is a symmetric block cypher cryptography algorithm. The most notable features of this algorithm are that the cypher block size is 64 bit, variable key length from 32 to 448 bits, known to be much faster than other algorithms, royalty-free (in the public domain), and no license required to use [4]. The algorithm is a quick, reliable, flexible, and unbreakable [4]. The biggest downfall is the 64-bit block, which the algorithm encrypts individually and is not as secure as other algorithms.

Twofish is the predecessor to blowfish and is the same as blowfish, with the difference that it is faster, more flexible, and more secure. It, too, is license free and in the public domain. It has a 128-bit data block, encrypts in 16 rounds, and is ideal for hardware. It was the fastest algorithm across all CPUs, competing to become the AES, and it fits into a few gates in the hardware [5]. There is no other algorithm available with the same flexibility in implementation, the ability of Twofish to trade off key-setup time and RAM and ROM for encryption speed [5].

Although AES is widely used and open-sourced, the code is not as small in size and length as Twofish. Twofish has the same benefits as AES but has shorter and faster code. RSA is asymmetric, which is not the preferred method of key encryption as it will be computationally intensive, which goes against UR006. Blowfish is faster than most algorithms but is not as secure as Twofish, while it does share the same benefits. The two encryption algorithm has a unique combination of flexibility, speed, and conservative design [5].

Although AES is widely used and open-sourced, the code is not as small in size and length as Twofish. Twofish has the same benefits as AES but has shorter and faster code. RSA is asymmetric, which is not the preferred method of key encryption as it will be computationally intensive, which goes against UR006. Blowfish is faster than most algorithms but is not as secure as Twofish, while it does share the same benefits. The two encryption algorithm has a unique combination of flexibility, speed, and conservative design [5].

1.3 Comparison of compression algorithms

Compression can be either lossy or lossless. In lossless compression, the physical file size is reduced while the integrity of the data is retained. This means that the original data can be perfectly recovered at the decompression stage without any depreciation of data quality. In lossy compression, the compression stage comprises the data quality by eliminating data which is not noticeable. This means that the original data can never be recovered after the compression is complete. Lossy compression compresses images, audio, and video, whereas lossless compression compresses text, audio, and images [6].

One of the foundational compression algorithms, LZ77, was created by Abraham Lempel and Jakob Ziv in 1977 [7]. Some notable compression software used today that are based on LZ77 are ZIP and GZIP [8]. The main idea behind LZ77 is to replace multiple occurrences of the same sequence of bytes with an index reference to the first occurrence of that sequence. This is achieved by using a sliding window to inspect the source sequence of bytes and to maintain a ledger of historical data that serves as a dictionary [9]. The sliding window comprises of a look-ahead buffer and a search buffer. The search buffer consists of the dictionary and the recently encoded data. The lookahead buffer contains the next portion of the input data sequence to be encoded [9]. The dictionary contains three values; Offset (distance between the start of sequence and beginning of the file), run-length (number of characters in the sequence), and the deviating characters (markers indicating a new sequence) [7]. The sliding window size is one of the major factors which affects the performance of the compression [9]. The shorter the sliding window size is, the faster the compression will take. However, the likelihood of finding repeated sequence occurrences will decrease; hence, the resultant compressed file will increase in size. Therefore, there is a trade-off to be made between the speed of compression and the effectiveness of the compression. In practice, the effectiveness of the compression is correlated to the data being compressed, and the sliding window size can be from several kilobytes to several megabytes [9].

LZR is a modification of LZ77 that was invented by Michael Rodeh et al. in 1981 [10]. LZR was designed to be a linear time alternative to LZ77; however, the encoded pointers can point to any offset in the file [7]. This results in the memory requirements of LZR increasing as the size of the input string increases [10]. Combined with its poor compression ratio, LZ77 is often superior; LZR is an impractical variant.

Lempel-Ziv-Storer-Szymanski [11], or LZSS, is another algorithm designed as a variant of LZ77. It was first published in 1982 by James Storer and Thomas Szymanski. LZSS manages to improve upon LZ77 by detecting whether a given substitution will decrease the compressed file size or not [7]. If it doesn't, the input is left in its original form; otherwise, the sequence is substituted with an offset-length pair [7]. LZSS also improves upon LZ77 by eliminating the use of deviating characters. This decreases the size of the dictionary and hence improves the efficiency of the compression. LZSS is commonly used for archive formats such as RAR [7].

Deflate (stylised as DEFLATE) was invented by Phil Katz in 1993 [12] and has become the basis for most compression tasks today. Deflate combines the LZ77 (or LZSS pre-processor) algorithm with Huffman coding. Huffman coding is a lossless entropy (smallest number of bits needed, on average, to represent a symbol) encoding compression method which assigns codes based on the frequency of a symbol occurrence [7]. The codes are stored in a Code Book,

constructed for the input string. Each of these codes is variable-length, where the length of the code is based on the frequency of the corresponding symbols [9]. There are two main properties behind Huffman coding; codes for more frequently occurring symbols are shorter than codes for less regularly occurring characters, and each code can be uniquely decoded [9]. Due to the last property, each of the codes must be prefix codes, meaning that the code for a symbol cannot be a prefix of a code for any other character. Huffman encoding has two steps. The first step is to build a Huffman tree (type of binary tree) of the original symbols from the input string. Then the Huffman tree is traversed, and codes are assigned to each character in the tree.

The final lossless compression algorithm to be evaluated is Brotli. Brotli is developed by Google. Brotli uses both LZ77 and Huffman coding like Deflate but introduces 2nd order context modelling to increase the effectiveness of the compression [13] while retaining similar speeds to Deflate. Context modelling uses preceding events to model the next event.

1.4 Feasibility analysis

The microcontroller uses a fixed-point ARM processor, which means it cannot process floating point values. This may affect the compression and encryption algorithms, leading to readdressing the choice of the algorithm used.

The algorithm used, either for compression or encryption, could be too computationally intensive for the microcontroller – which may result in a revision in which a slower but less computationally intensive algorithm must be used. The algorithm for compression may have a high loss rate, failing to meet the requirement UR003.

The IMU has different modes of operation, which will allow for a low power consumption mode, increasing the battery's life span. Other operation modes could lead to a short life span of the battery, which could hinder the entire system.

Another issue which could hinder the project is that the Discovery Board only has 64kb flash memory and 8 bytes of RAM, which means that a complex method will be needed to run the compression and encryption algorithm on the STM Discovery Board itself.

Filtering the data so that 25% of the IMU’s coefficients are extracted may be a challenge to implement. It is also tough to process and work with the STM Discovery Board due to our lack of experience and knowledge of the hardware. Further research and learning may be required.

Despite the above challenges, we have the time and resources to ensure that all user requirements are met.

1.5 Possible bottlenecks

Some trade-offs and restrictions have already been detailed in the sections above. Some are yet to be documented. The bottlenecks due to trade-offs expected to occur are described in the following paragraphs.

In terms of data collection, the size of each packet of data that needs to be transferred must be considered. The data link between the STM discovery board and the sensor HAT could create congestion if the transfer speed is too slow or the size of packets is too large.

The next bottleneck to consider is the processing time. This bottleneck is affected by the collection of data bottleneck detailed above. The larger the packet of data to be compressed and encrypted, the longer it will take to be processed and the longer it will take to be transmitted.

The STM storage is limited to 64 kilobytes of flash memory. Initial estimations place the size of the software around 34 kilobytes. This means there are only 30 kilobytes of ‘free’ storage for data collection before compression and encryption.

The choice of compression and encryption algorithms could become a significant issue. The stronger the compression, the longer it will take to compress a given file, as well as increasing the amount of RAM used for the compression software. In terms of encryption, every encryption algorithm will operate at different speeds depending on the input data type. Some algorithms work better with finite data lengths, whereas others work better with varying data lengths.

The collection, compression or data encryption processing time may require significant data processing time. In that case, the power usage of the STM will increase significantly. Depending on the physical set-up of the buoy-STM system, this could be either a minor or major issue.

2. Subsystem Design

2.1 Subsystem and Sub-subsystems Requirements and Specifications

2.1.1 Retrieval & Storage of Data

This subsystem focuses on retrieving the data generated by the IMU and storing said data on the STM32F0 Discovery board (further referred to as STM). We are constrained only to use 64 kilobytes of flash memory and 8 kilobytes of RAM due to the design of the STM. The STM cannot add external storage space like an SD card.

Functional Requirement ID	Description	User Requirement Addressed
FR1	The IMU is a 9-axis MEMS Motion Tracking device is used.	UR7
FR2	Data cannot be stored in large quantities on the STM.	UR8
FR3	The processing on the STM must not be computationally intensive.	UR6

Table III – Functional requirements of the retrieval and storage of data

Design Specification ID	Description	Functional Requirement ID
DS1	The ICM-20948 MEMS motion tracking is used.	FR1

Table IV – Design specifications for the retrieval and storage of data.

2.1.2 Data Processing

UR003 states that at least 25% of the lower Fourier Coefficients must be extracted. This reduces the strain on the STM as it has limited onboard storage, and as there is no possibility of increasing that storage, this will play a vital role in the entire system. The reduction in the size of the collected data will also increase the efficiency of the compression and encryption algorithm. The IMU cannot record the data in the frequency domain; it measures in the time domain – it is essential for a system to be designed to transform the data to the frequency domain. It is desired to use the Fast Fourier Transform (FFT) to do such a transform for efficiency. To use the FFT, the correct sample time must be used, and the amount of data collected must also be compatible with the onboard storage of the STM. This will allow the subsequent Fourier coefficients to be collected and processed. Based on research from the STM IDE page, it is recommended that the programming language to be used is either C, C++, Pascal, or Java. In the case of this project, we will use a combination of programming languages on the computer but predominantly use C on the STM.

Functional Requirement ID	Description	User Requirement Addressed
FR4	Some of the Fourier coefficients must be retained after compression	UR3
FR5	The programming language used must be compatible with the STM Discovery Board without modifications.	UR8 & UR5

Table V – Functional requirements for data processing

Design Specification ID	Description	Functional Requirement ID
DS2	At least 25% of the Fourier Coefficients must be retained, this can be done by filtering the data using code.	FR4 & FR5

Table VI – Design specifications for data processing

2.1.3 Encryption of data

The encryption will be done using the Twofish algorithm. This is one of the world's fastest, most flexible and most secure algorithms. Twofish is also compatible and easy to implement on the STM32F051 Discovery Board. Ideally, the encryption will be run on the STM, but this depends on the onboard memory. Encrypting data packets that are transmitted is good practice for ensuring the confidentiality of the information. Twofish will provide secure encryption for the data from the SHARC buoy to the researchers.

Functional Requirement ID	Description	User Requirement Addressed
FR6	All data received by and from the STM must be suitably encrypted.	UR1
FR7	The encryption algorithm must be light and computationally inexpensive.	UR6 & UR4

Table VII – Functional requirements for encryption of the data

Design Specification ID	Description	Functional Requirement ID
DS3	Twofish algorithm to be used as it is lightweight and is suitable for all types of encryption.	FR6 & FR7

Table VIII – Design specifications for encryption of the data

2.1.4 Compression of Data

Based on the comparisons made in section xxx between various compression algorithms, the LZSS compression algorithm was chosen. This algorithm was selected due to its high compression ratio, speed, and relative lack of complexity. The data compression will occur before the data's encryption. This is because, after encryption, the data becomes primarily random; hence, the compression algorithm is unlikely to find enough matching sequences of characters to compress the data file effectively.

The SHARC buoy requires a compressed file due to the use of a satellite link to transmit the data. This method of communication has high data transfer costs and inconsistent transmission and bandwidth. Therefore, any efforts to reduce the transmission file size are critical.

Functional Requirement ID	Description	User Requirement Addressed
FR8	All data received by and from the STM must be suitably compressed.	UR2
FR9	The compression algorithm must be effective and computationally inexpensive.	UR6 & UR4

Table IX – Functional requirements for compression of data

Design Specification ID	Description	Functional Requirement ID
DS4	LZSS algorithm is to be used as it is effective and is suitable for text based compression.	FR8 & FR9

Table X – Design specifications for compression of data

2.1.5 Transmission of Data

Once the data has been compressed, encrypted, and checked if the contents are sufficient – have at least 25% of the lower Fourier coefficients, the data will be transmitted from the SHARC buoy via Iridium. Iridium is a satellite communications company whose only service is providing voice and data services to everywhere on the earth, even the remote Antarctic. Iridium is extremely costly, so compression and encryption are vital in ensuring the project's feasibility.

Functional Requirement ID	Description	User Requirement Addressed
FR9	All compressed and encrypted data must be received by the researchers.	UR1 & UR2 & UR3

Table XI – Functional requirements for the transmission of data

Design Specification ID	Description	Functional Requirement ID
DS5	All encrypted and compressed data will be transmitted via the Iridium network from SHARC buoy to researchers.	FR9

Table XII – Design specifications for the transmission of data

2.1.6 Checksum

Before the compression and encryption of the data, a CRC-32 checksum of the data file will be created. After the data is transmitted, decrypted, and decompressed on the client side, another CRC-32 checksum will be made. These two checksums can then be compared to verify whether the data file has changed during any of the stages between the data collection and the data decompression.

Functional Requirement ID	Description	User Requirement Addressed
FR10	The data received by the researchers must be identical to the data recorded on the STM.	UR3

Table XIII – Functional requirements for the checksum subsystem

Design Specification ID	Description	Functional Requirement ID
DS6	The integrity of the data will be verified by a CRC-32 checksum.	FR10

Table XIV – Design specifications for the checksum subsystem

2.2 Inter-Subsystem and Inter-Sub-subsystems Interactions

All the above submodules form the primary system where the compression, encryption and transmission of the data received from the IMU will occur. Although the STM itself is a submodule of the overall SHARC buoy, this paper design treats the STM as if it were the main module – all of the above submodules form a part of the larger STM submodule. The IMU-20948 is another submodule apart from the SHARC buoy, which will interact with the STM, allowing for the transmission of data from the sensor to the STM. This data transmission between the IMU and STM is a sub-subsystem of the overall buoy system.

Once the STM has filtered the data, a CRC-32 checksum of the data file will be created. The data will be compressed using the LZSS algorithm. This task is performed by the 4th submodule outlined above. Following compression, the Twofish algorithm encrypts the data before it is transmitted to the computer – this is the 3rd submodule described above. Following encryption, the data processing is complete, and the encrypted data packet is now ready for transmission via Iridium to the researchers.

It should be noted that the entire subsystem depends on the lifespan, battery life, and storage on the STM Discovery board and is susceptible to the IMU-20948, which will determine the sample rate. This dictates the efficiency of submodule one above and will impact the overall SHARC buoy. The processes described above are explained in the diagram below.

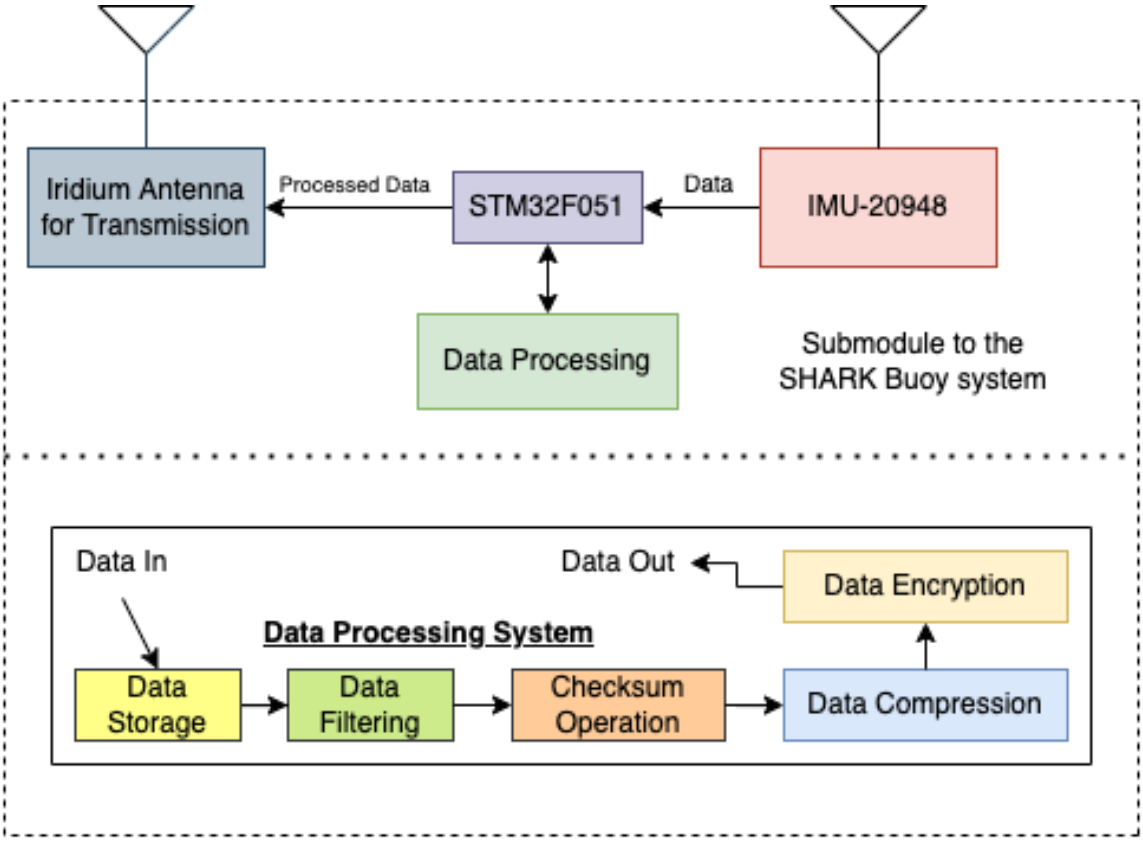

Figure 3 - Diagram of the system in operation

2.3 UML or OP Diagrams

The following UML diagram depicts how all the submodules interact to make up the overall SHARC buoy system and the flow in which the submodules operate. Note that this is a sequential process – a submodule must complete its operation before the next submodule can proceed with its task.

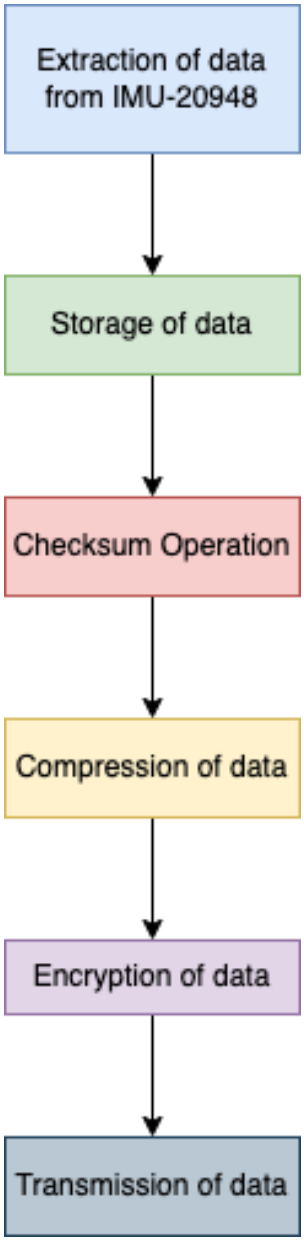

Figure 4 - UML showing the operation of the STM module of the SHARC buoy System

3. Acceptance Test Procedure

The acceptance test procedures used to evaluate the performance of the design and whether it meets the minimum requirements are analysed in the following subsections below.

3.1 Figures of Merit

Subsystem	Figures of Merit
1: Retrieval & Storage of Data	The sample time of sensor values should be correct. The sampled data should not exceed the storage restrictions due to the size of the code files.
2: Data Processing	Determine the execution speeds, efficiency, and correct processing of the STM.
3: Encryption of Data	The execution time of the algorithm should not exceed 10 seconds per 10 kilobytes of data. The algorithm should be secure as well as unlikely to be decrypted.
4: Compression of Data	The compression algorithm should be efficient and take no longer than 5 seconds per 10 kilobytes of data. No data must be lost during the compression of the data file. The compression ratio of the original file size and the compressed file size should be at least 1.5.
5: Checksum	The checksum of the original data file should be created successfully on the STM and it should match the checksum of the decompressed data file on the client-side.
6: Transmission of Data	No packet loss should occur during transmission of the encrypted data file.

Table XV– Figures of merit for each subsystem

3.2 Experiment Design of ATPs

The compression and encryption experiment design handle the data processing subsystem ATP.

3.2.1 Experiment Design to Test the Compression ATPs

Various sample IMU datasets are available for testing the compression figures of merit. The LZSS compression algorithm will be used to compress the data file. The LZSS decompression algorithm will also be used to decompress the compressed file. The resultant compressed file can be compared to the original data file to determine the compression ratio. The decompressed file can then be compared to the original data file to determine whether the data was corrupted. The execution time of the compression (and decompression) algorithm can be recorded to create a benchmark to determine the mean speed of compression.

3.2.2 Experiment Design to Test the Encryption ATPs

Various sample IMU datasets are available for testing encryption figures of merit. The Twofish algorithm will be used to encrypt the data file and decrypt the encrypted file. The quality of the encryption algorithm can be determined by attempting to decrypt the file using different keys. If any key besides the key generated by the encryption algorithm manages to decrypt the file, then the encryption is not successful. The execution time of the encryption (and decryption) algorithm can be recorded to create a benchmark to determine the mean speed of encryption. The last test will compare the original file to the decrypted file to determine whether it is human-readable.

3.2.3 Experiment Design to Test the Checksum & Transmission of Data ATPs

A CRC-32 checksum of the data file can be created before compression and encryption. The data file will then be compressed, encrypted, and transmitted to the user. The file can then be decrypted and decompressed on the users' end, and a CRC-32 checksum of the resultant file can be created. The users' checksum and the STM's checksum can then be compared. If the two checksums are different, then the original data file was likely compromised during any stages between the creation of the two checksums.

3.3 Acceptable Performance Definition

The design needs to be able to compress the data sufficiently enough to be efficiently transferred by the Iridium satellite network. However, no data should be lost during the compression process; therefore, lossless compression must be used. The design also needs to encrypt the compressed data to ensure that the transmission is confidential. The processing handled on the buoy must consume as little battery power as possible to ensure that the design does not incur frequent battery recharging.

3.3.1 Compression subsystem

ATP	Description
Compression time	The compression algorithm must take no longer than 5 seconds per 10 kilobytes of data.
Data loss	No data must be lost during the compression of the data file.
Compression effectiveness	The compression ratio of the original file size and the compressed file size must be at least 1.5.

Table XVI – ATPs for the compression subsystem

3.3.2 Encryption subsystem

ATP	Description
Encryption time	The execution time of the encryption and decryption should not exceed 10 seconds per 10 kilobytes of data.
Data loss	No data must be lost during the encryption and decryption of the data file.
Encryption security	The encryption must be strong enough to prevent decryption .
Encryption integrity	The original data file's contents must be identical to the decrypted data file's contents.

Table XVII – ATPs for the encryption subsystem

3.3.3 Checksum subsystem

ATP	Description
STM checksum	The STM must create a checksum of the original data file successfully.
Client checksum	The client's device must create a checksum of the decrypted and decompressed file successfully.
Checksum comparison	The client's and STM's checksums must be identical.

Table XVIII – ATPs for the checksum subsystem

References

[1] IBM, “Transaction Processing Facility Enterprise Edition,” 05 March 2021. [Online]. Available: https://www.ibm.com/docs/en/ztpf/1.1.0.14?topic=concepts-symmetric-cryptography. [Accessed 17 August 2022].

[2] National Institute of Standards and Technology (NIST), “NIST Technical Series Publications,” 26 November 2001. [Online]. Available: https://nvlpubs.nist.gov/nistpubs/fips/nist.fips.197.pdf. [Accessed 18 August 2022].

[3] K. Meghna, “Geeks for Geeks,” Geeks for Geeks, 24 February 2022. [Online]. Available: https://www.geeksforgeeks.org/block-cipher-modes-of-operation/?ref=lbp. [Accessed 18 August 2022].

[4] B. Schneier, “Schneier on Security,” [Online]. Available: https://www.schneier.com/academic/blowfish/. [Accessed 18 August 2022].

[5] B. Schneier, “Schneier on Security,” December 1998. [Online]. Available: https://www.schneier.com/academic/archives/1998/12/the_twofish_encrypti.html. [Accessed 18 August 2022].

[6] Geeks For Geeks, “Difference between Lossy Compression and Lossless Compression,” Geeks For Geeks, 8 June 2020. [Online]. Available: https://www.geeksforgeeks.org/difference-between-lossy-compression-and-lossless-compression/. [Accessed 17 August 2022].

[7] L. F. Buenavida, “Crunch Time: 10 Best Compression Algorithms,” DZone, 28 May 2020. [Online]. Available: https://dzone.com/articles/crunch-time-10-best-compression-algorithms. [Accessed 17 August 2022].

[8] D. Budhrani, “How data compression works: exploring LZ77,” Towards Data Science, 12 September 2019. [Online]. Available: https://towardsdatascience.com/how-data-compression-works-exploring-lz77-3a2c2e06c097. [Accessed 17 August 2022].

[9] E. Chen, “Understanding zlib,” January 2019. [Online]. Available: https://www.euccas.me/zlib/#deflate_lz77. [Accessed 17 August 2022].

[10] M. Rodeh, V. R. Pratt and S. Evan, “Linear Algorithm for Data Compression via String Matching,” *Association for Computing Machinery,* vol. 28, no. 1, pp. 16-24, 1981.

[11] J. A. Storer and T. G. Szymanski, “Data compression via textual substitution,” *Association for Computing Machinery,* vol. 29, no. 4, pp. 928-951, 1982.

[12] L. P. Deutsch, “DEFLATE Compressed Data Format Specification version 1.3,” May 1996. [Online]. Available: https://datatracker.ietf.org/doc/html/rfc1951#section-4. [Accessed 17 August 2022].

[13] Google, “GitHub Brotli,” GItHub, 11 October 2013. [Online]. Available: https://github.com/google/brotli. [Accessed 18 August 2022].