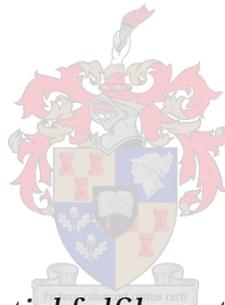


An FPGA-based Adaptive Forward Error Correction Protocol for CubeSats

by

Sandile Mkhali



*Thesis presented in partial fulfilment of the requirements for
the degree of Master of Science in Engineering (Electronic) in
the Faculty of Engineering at Stellenbosch University*

Supervisor: Mnr. A. Barnard

March 2017

Declaration

By submitting this thesis electronically, I declare that the entirety of the work contained therein is my own, original work, that I am the sole author thereof (save to the extent explicitly otherwise stated), that reproduction and publication thereof by Stellenbosch University will not infringe any third party rights and that I have not previously in its entirety or in part submitted it for obtaining any qualification.

Date: March 2017

Copyright © 2017 Stellenbosch University
All rights reserved.

Abstract

CubeSats have become popular due to their simplified model that reduces development time and costs. The standard, however, suffers from limitations imposed by the small form factor. Research is undertaken at different levels to improve the performance of CubeSats, of which one is on the communication subsystem. The question is how the throughput per satellite-to-ground communication session can be improved using modified error correction methods.

Previous work at the ESL proposed a hybrid protocol design of the AX.25 and the FX.25, known as the AFX.25, whose simulation results suggested improved performance over pure protocol implementations. The AX.25 protocol has an error checking functionality but without error correction, so the FX.25 was introduced as a wrapper to the AX.25 to provide for error correction. Inasmuch as the AX.25 is popular among university CubeSat designs, it was necessary that an investigation be done to evaluate if it was the best choice of implementation. The CCSDS Telecommand protocol was chosen for performance evaluation against the AFX.25 due to its functionality which is closer to the FX.25. The evaluation was based on simulation and hardware complexity analysis. SatSim was used as a satellite network simulation environment. The results showed that the AFX.25 is a better choice over the CCSDS TC.

The AFX.25 hardware design and implementation was therefore considered on a Field Programmable Gate Array (FPGA). The FPGAs' parallel processing capability makes them an attractive choice of implementation for error encoding and decoding. The adaptive protocol was designed to switch between no error correction (AX.25) and error correction (FX.25) where the number of correctable errors is 8 using the Reed Solomon code (255, 239). The switching from AX.25 to FX.25 is determined by the packet loss rate while switching from FX.25 to AX.25 is influenced by the packet success rate. The system was implemented on a Fusion M1AFS1500 development board interfaced with a half duplex RF board.

Tests were carried out successfully on a terrestrial testbench which modelled a typical satellite pass.

Uittreksel

CubeSats het populêr geword weens hulle vereenvoudigde model wat beide ontwikkelingskoste en tye verminder. Die CubeSatstandaard het egter beperkings weens die klein formfaktor. Navorsing word op verskeie vlakke uitgevoer om die effektiwiteit van CubeSats te verbeter, met die kommunikasiesubstelsel wat ook aandag geniet. 'n Sleutelvraag is hoe die datadeurset van 'n satelliet tot grondstasie kommunikasie sessies verbeter kan word met spesiale metodes om datafoute te verhoed.

Vorige werk by die ESL het 'n hibriedeprotokol ontwerp vir AX.25 en FX.25 voorgestel, bekend as AFX.25 wat se simulasiereultate better gelyk het as die suiwer protokol toepassings. Die AX.25 protokol kan foute optel, maar kan nie hulle regstel nie, dus was FX.25 voorgestel as 'n aanpassing op AX.25 om data ontfouting uit te voer. Alhoewel AX.25 reeds populêr is vir Universiteitstoepassings, is 'n ondersoek ingestel om te bevestig of dit die beste keuse vir implementings is. Die CCSDS protokol is gekies vir hierdie vergelyking teenoor AFX.25 aangesien die protokol se werking soortgelyk aan FX.25 is. Die vergelyking het gebruik gemaak van simulaties en 'n analise van die hardwarekompleksiteit. SatSim was gebruik as die satellietnetwerk simulator. Die resultate het voorgestel dat AFX.25 better resultate lewer as CCSDS TC.

Die AFX.25 hardwareontwerp en implementering was dus gedoen vir 'n Field Programmable Gate Array (FPGA). Die vermoë van FPGA's om parallele verwerking uit te voer maak dit 'n goeie keuse vir fout enkodering en fout dekodering. AFX.25 was ontwerp om te skakel tussen 'n protokol sonder data ontfouting (AX.25) en een met data ontfouting (FX.25) waar die hoeveelheid herstelbare foute 8 is deur gebruik te maak van Reed Solomon kode (255, 239). Die oorskakel vanaf AX.25 tot FX.25 word bepaal deur die tempo waarteen pakkies verlore gaan terwyl die oorskakel vanaf FX.25 tot AX.25 word bepaal deur die tempo waarteen pakkies suksesvol gestuur word. Die sisteem was geïmplementeer op 'n Fusion M1AFS1500 ontwikkelingbord wat gewerk het met 'n half duplex RF-bord.

Toetse is suksesvol uitgevoer op 'n aardstoetsbank wat 'n tipiese satellite kommunikasie sessie gemodelleer het.

Acknowledgements

I would like to express my sincere gratitude to the following individuals and organisations:

Mnr A. Barnard - I would like to acknowledge my supervisor who assisted me throughout my study period. Thank you for the insightful, thoughtful ideas and timely recommendations that you gave me. The knowledge and experience that you shared give me the boldness to say that indeed I was standing on the shoulders of the great, from satellite communications to VHDL design, debugging and testing. I had challenging moments throughout my research. Thank you for the patience you showed and eye-opening discussions we had when I was thinking otherwise. The project would not have been realised without your intervention to acquire the FPGA boards.

Prof. Steyn - I would also like to acknowledge the first professor whom I contacted before being enrolled at Stellenbosch, Professor Steyn. Thank you, Prof, for giving me the opportunity to be part of the Maties community and experience what it is like to be among South Africa's best. And thank you for helping me with the bursary arrangements. Moreover, thank you for letting me use the RF boards for my project.

Electronic Systems Laboratory (ESL) - I spent my entire study time in the ESL lab, on my way to the ESL or in my house thinking about what I was going to do in the ESL. I would like to extend my appreciation to the entire ESL team; indeed you guys are great. Specifically, I cannot help but appreciate Willem (now Dr.) who always has a heart for welcoming new students and showing them around, and checking on them every now and then. He is the guy who is always willing to help irrespective of the time of day, so long as he is in the ESL. I would also acknowledge the lab engineers (who are now the CubeSpace engineers): Mark-Alec, those notes were helpful. I must not forget to mention the IT geeks in the labs who always ensured we accessed the software and equipment we needed: Cornelius, Andrew, Clint, Ryan. I also enjoyed the pub lunches, the camps and the end-of-year braais: I will miss the food.

EE Department - My degree would have never been realised if it weren't for the Electrical and Electronic Engineering department who offered me an opportunity to learn and further offered me a bursary. I am grateful. Mr Booysen: thank you, Sir. Let me also acknowledge the RF engineer in the 4th floor RF lab who gladly assisted me when I needed to debug the RF 'black magic': thank you Ma'am.

AJ Merts - Now this is the coolest guy, my office mate. Thank you AJ, I gained a lot from your

VHDL and programming experience. I enjoyed every moment in our office not because the research was a walk in the park but because the environment was conducive to work.

Bongani and Dino - Allow me to acknowledge the guys that I would hang out with during spare hours, had chats and with whom I share meals sometimes. But most importantly, thank you for allowing me to use your computers, bags, bikes and accompanying me to Coetzenberg mountain during the hardware test sessions.

Dr. Funlola Olojede - I would like to thank my pastor who taught me a lot in faith and well being and the prayers she made for me. Thank you ma.

My family - This one goes to my family who are always in my heart. They offered emotional, spiritual, and financial support and for that I am grateful.

And now to Him who is able to do more than we can think or imagine, to whom all the victory belongs, to my Father and Lord, all the glory be unto you.

Dedications

*I dedicate this work to:
My family, my mom, my sister and brothers and my nieces
My future family, my sweetie π and kids*

Contents

Declaration	i
Abstract	ii
Uittreksel	iii
Acknowledgements	iv
Dedications	vi
Contents	vii
List of Figures	ix
List of Tables	xi
Abbreviations	xii
1 Introduction	1
1.1 Problem Analysis	1
1.2 Related Work	3
1.3 Research Question	5
1.4 Research Statement	5
1.5 Research aim and objectives	5
1.6 Design requirements	6
1.7 Value of project	7
1.8 Overview of this work	7
2 Simulation Environment	8
2.1 Communication Simulation	8
2.2 Link Budget Simulation	10
3 Protocol Simulation and Evaluation	16
3.1 Introduction	16

<i>CONTENTS</i>	viii
3.2 AX.25	16
3.3 FX.25	20
3.4 AFX.25	22
3.5 CCSDS Telecommand(TC)	24
3.6 Simulation Scenario	29
4 Adaptive AFX.25 Hybrid Hardware	34
4.1 Protocol Choice	34
4.2 FEC Theoretical Background for FX.25	37
4.3 FPGA-based Hardware	48
4.4 Hardware Design and Implementation	49
4.5 Testing and Results	63
4.6 AFX.25 Implementation Real Estate Cost	70
5 Conclusion and Future Work	71
5.1 Conclusions	71
5.2 Future Work	72
Appendices	74
A Cubesat Communication Survey 2015	75
B AX.25 Layer 3 Protocol Definitions	77
C CCSDS Telecommand Additional Information	78
C.1 Sending Node Architecture	78
D GF(256) Elements	79
E Adeunis RF ARF6921D 869.525 MHz Board	82
E.1 Transmission Power Configuration	82
E.2 Busy	82
E.3 Specifications	83
References	84

List of Figures

1.1	Acquisition and Loss of Signal for a Satellite to Ground Communication link . . .	2
1.2	Theoretical Throughput for a FEC and non-FEC protocol over a satellite pass [1] .	3
2.1	Basic satellite-ground communication configuration	12
3.1	Amateur X.25 Protocol (AX.25) Protocol Stack. Adapted from AX.25 Specification Report [2], pg.2	16
3.2	AX.25 Frame types. Adapted from AX.25 Specification Report [2], pg.6	17
3.3	AX.25 UI frame. Adapted from Ref. [2]	19
3.4	Forward Error Correction Extension to AX.25 (FX.25) basic frame format. Adapted from FX.25 Specification Report [3], pg.3	20
3.5	Protocol Transition States	23
3.6	Throughput Curves for different BER for a 156 byte payload data	24
3.7	TC Sending Node. Adapted from [4], pg.2-12	25
3.8	TC Transfer Frame. Sourced from Telecommand Space Data Link Protocol Recommendation Report [4], pg.4-1, pg.4-2	26
3.9	TC Synchronisation and Channel Coding Sublayer and Physical Layer. Adapted from [5], pg.2-4	27
3.10	BCH Codeblock. Adapted from [5], pg.2-4	27
3.11	Communications Link Transmission Unit (CLTU) Unit Format. Adapted from [5] .	28
3.12	Throughput Curves for different BER for different payload sizes	31
3.13	Throughput for FX.25, AFX.25, CCSDS Telecommand of data size 156 bytes	33
4.1	BCH(63,56) Code Generator. Adapted from TC Synchronisation and Channel Coding Recommendation [5], pg.3-2	34
4.2	Reed Solomon (255,239) Encoder Architecture. Adapted from [6], pg.2	35
4.3	Galois Field Adder	39
4.4	Galois Field (256) Constant Multiplier for a constant α^3 (8_{10})	40
4.5	Reed Solomon Codeblock	41
4.6	Reed Solomon (RS)(n,k) Encoder for code generator polynomial $g(X) = 1 + g_1X + g_2X^2 + \dots + g_{n-k-1}X^{n-k-1} + g_{n-k}X^{n-k}$. Adapted from [7]	42
4.7	Reed Solomon Syndrome Computation Circuit for a single S_i . Adapted from [7] .	43

4.8	Error locator polynomial roots search unit. Adapted from [7]	47
4.9	FPGA Design Flow	49
4.10	System High-level Block Diagram	50
4.11	Application Layer Terminal	50
4.12	FPGA Board - Encoder	51
4.13	CRC-16-CCIT Generator, $g(X) = x^{16} + x^{12} + x^5 + 1$	52
4.14	Reed Solomon (255,239) Encoder Circuit	53
4.15	RF Transceiver Block Diagram. Adapted from [8]	54
4.16	RF Clock and Data Synchronisation. Adapted from [8]	55
4.17	System Decoder Block Diagram	55
4.18	Reed Solomon (255,239) Syndrome Circuit	58
4.19	Berlekamp Circuit	59
4.20	Forney's Error Magnitude Calculator Circuit	62
4.21	Semi closed loop Libero Smart Design for ModelSim simulation	64
4.22	Simulation showing channel noise injected in the data	65
4.23	Received Codeword Into and Out of RS Decoder	66
4.24	Throughput Curves for AX.25/FX.25/AFX.25 Indoor Tests	67
4.25	Terrestrial Test Environment	68
4.26	Throughput Curves for AX.25/FX.25/AFX.25 Terrestrial Tests	69
C.1	TC Sending Node. Adapted from [4]	78
E.1	Switching timing signals. Adapted from [8].	82

List of Tables

1.1	CubeSat Non-proprietary Communication Protocols Summary. Adapted from the CubeSat Communication System Table, Appendix A by Klofas [9]	5
2.1	RF Modulation Schemes and BER equations. Adapted from Ref [10]	11
3.1	AX.25 Control Field. Adapted from Ref. [2]	19
3.2	FX.25 Correlation Tag Values. Adapted from FX.25 Specification Report [3], pg.7 .	20
3.4	AX.25/AFX.25 Data Received over 400 seconds for different success rate thresholds	24
3.5	AX.25/FX.25/AFX.25/CCSDS TC Data Received over 800 seconds	33
4.1	Protocols Comparison Summary	36
4.3	GF(256) elements generated from the generator polynomial, $\phi(X) = X^8 + X^4 + X^3 + X^2 + 1$	38
4.5	Berlekamp Iterative Stages. Adapted from [7]	45
4.6	Adeunis ARF6921D RF Board Interface. Adapted from [8]	54
4.8	AX.25/FX.25/AFX.25 Data Received over 140 seconds indoor tests	68
4.9	Resource Utilisation for Fusion MFS1500	70
4.10	Resource Utilisation for IGLOO AGL600V5	70
B.1	AX.25 Layer 3 Protocol Definitions	77
E.1	Transmission power configuration. Adapted from [8]	82
E.2	ARF6921D Specifications	83

Abbreviations

- AOS** Acquisition of Signal. 2
- ARQ** Automatic Repeat reQuest. 2, 3, 19, 25
- ASK** Amplitude Shift Keying. 11
- ATM** Asynchronous Transfer Mode. 3
- AX.25** Amateur X.25 Protocol. ix, xi, 4, 16–24
- BCH** Bose-Chaudhuri-Hocquenghem. 27, 28, 37
- BER** Bit Error Rate. 10, 11, 15
- BPSK** Binary Phase Shift Keying. 11, 30
- BSC** Binary Symmetric Channel. 14
- CLTU** Communications Link Transmission Unit. ix, 28
- CRC** Cyclic Redundancy Check. 18, 51
- CSP** Cubesat Space Protocol. 4
- ESL** Electronic Systems Laboratory. 4, 6, 9, 49, 66
- FCS** Frame Check Sequence. 18
- FEC** Forward Error Correction. 2, 3, 5, 7, 20, 21, 37
- FPGA** Field Programmable Gate Array. 4, 5
- FSK** Frequency Shift Keying. 11
- FSM** Finite State Machine. 57
- FX.25** Forward Error Correction Extension to AX.25. ix, xi, 4, 20–22, 24, 32, 37
- GF** Galois Field. 37, 38, 61
- IP** Internet Protocol. 8
- IP over DVB-S2** Internet Protocol over Digital Video Broadcasting via Satellite Second Generation. 4

ITU International Telecommunications Union. 13

LDPC Low Density Parity Codes. 37

LEO Low Earth Orbit. 1, 3

LFSR Linear Feedback Shift Register. 34

LOS Loss of Signal. 2

M-ary PSK M-ary Phase Shift Keying. 11

MSK Minimum Shift Keying. 11

P/F Poll/Final. 18

PER Packet Error Rate. 4

QPSK Quadrature Phase Shift Keying. 11

RAM Random Access Memory. 52, 61

RS Reed Solomon. ix, 40, 42

RS Reed Solomon. 3, 4

SSID Secondary Station Identifier. 17

TF Transfer Frame. 25, 27

VC Virtual Channel. 25

Chapter 1

Introduction

1.1 Problem Analysis

Back in 1999, the CubeSat co-founders¹ envisioned the CubeSat standard as an educational tool to enable students to have hands-on experience with space. CubeSats can be designed and built in a relatively shorter period and with smaller budgets than traditional satellites. Seventeen years on from then, the CubeSat community has seen remarkable progress with the standard, with high schools, universities, non-governmental organizations and government agencies flying their CubeSat experiments into space. Furthermore, CubeSats are increasingly being adopted at industry level, with the likes of Planet Labs[12] on the front line.

However, much like every invention, the CubeSat suffers some limitations. The cube surface area does not allow the mounting of bigger solar panels, resulting in limited available power which imposes direct limitations on the CubeSat subsystems. Amongst the subsystems, the CubeSat has a communication subsystem which transmits to and receive information from other satellites or ground stations. The communication subsystem, which typically consumes a bigger percentage of power, requires a careful design to work within the tight power budget. This is even more necessary considering that CubeSats orbit the Low Earth Orbit (LEO) which results in narrow communication windows for space-to-ground communications.

Consider a CubeSat on a circular LEO² orbit as shown in Figure 1.1. Each satellite pass will last up to a maximum of 10-15 minutes for typical educational missions. Chu *et al.* [14] pointed out that approximately 78% of the communication window time yields useful communication data because at lower elevation angles the signal path loss is very high [14].

¹Professors, Jordi Puig-Suari and Bob Twiggs[11]

²LEO has altitudes between 160km and 2000km [13]

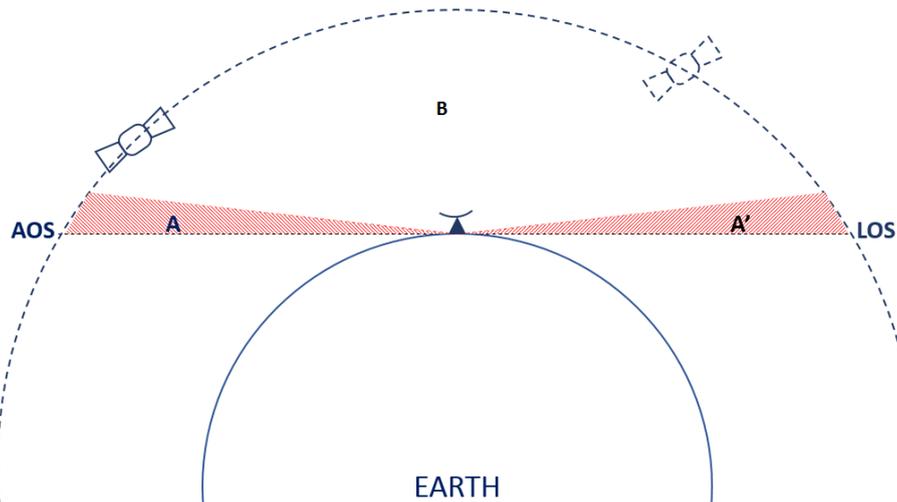


Figure 1.1: Acquisition and Loss of Signal for a Satellite to Ground Communication link

The satellite to ground station link is dynamically changing from the Acquisition of Signal (AOS) to the Loss of Signal (LOS) due to changes in the clearness of line of sight, antenna pointing errors, signal shadowing by obstacles, signal propagation distances between the 2 communication nodes [15]. This results in regions of severe signal degradations, better and finally good signal reception. In Figure 1.1, regions A and A' indicate regions where the signal to noise ratio is low and increases until the point of 'clear' line of sight in B. To guarantee a certain quality of service, regions A and A' requires a satellite's communication subsystem that implements an open-loop design (Forward Error Correction (FEC) protocol) to recover the lost packets for real time applications or a closed-loop (Automatic Repeat reQuest (ARQ)) design for time-insensitive applications or a combination of the two. In an optimal system, the open-loop module will execute first in an attempt to correct corrupt data packets of which upon failure will request retransmissions using the ARQ. Retransmissions come with round trip delays, thus this research work considers an open loop design.

A satellite communication session throughput analysis for an FEC enabled protocol and a non-FEC protocol for a 13 minutes satellite pass is shown in Figure 1.2. The protocol with error correction capability performs better than non-FEC in regions A and A', however; in region B the non-correction protocol outperforms the correction protocol. This is owing to the fact that at region B the packet loss rate is low, the FEC redundancy bits deteriorates the throughput since they are not necessary at this interval.

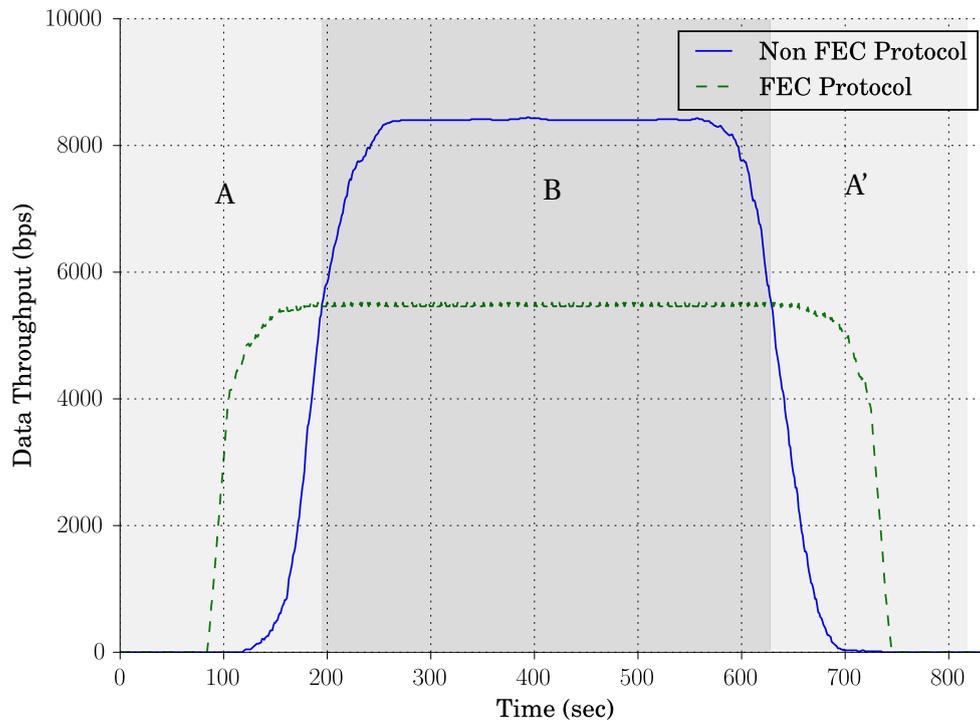


Figure 1.2: Theoretical Throughput for a FEC and non-FEC protocol over a satellite pass [1]

Several authors have adopted different approaches for optimal use of the bandwidth for maximum throughput in scenarios related to Figure 1.2 which is covered in the next section.

1.2 Related Work

Earlier works [16][17][18][19][20][21][22] have explored the subject for time-varying and noisy channels. Two submissions are prevalent, firstly using a rate adaptive code and secondly a hybrid with an ARQ protocol. Few literature sources could be found which are directly applicable to CubeSats, thus the following discussion covers mostly wireless channels as in many ways their principle of operation can be applicable to CubeSat LEO communication.

Emmelmann and Bischl [17] presented a hybrid protocol which switches between three protocols, a non-error correcting code (unnamed), a Reed Solomon (RS)(63,53) and a 1/2 rate turbo code concatenated with a RS(63,53) scheme for an Asynchronous Transfer Mode (ATM) based LEO network to provide a fixed data rate. Their protocol was simulated on FreeBSD[17]. However, the published reference does not give all the switching parameters and implementation details. It only mentions that the switching is based on the signal-to-noise ratio and the channel error rate at a given instance. On a video multi-cast application, Lamoriniere *et al.* [20] presented an error correction hybrid solution using four schemes, the adaptive XOR-based FEC, a RS based FEC, an Unequally Interleaved FEC, and a PRO-MPEG. The switching control is based on two performance metrics, the recovery ratio and efficiency

ratio which give indicators of recovered packets and total successfully received packets. A loss ratio is calculated each time, and a suitable scheme is selected.

Littman [16] proposed that a 3-state Markov model be used for the channel model with a code selection mechanism. The states represent the channel as bad, intermediate, or good, based on throughput and Packet Error Rate (PER) metrics. The code is concatenated, with a Rate Compatible Convolutional Code as an inner code, and a RS code as an outer code.

The Reed Solomon is seemingly a popular choice when implementing error correction for burst errors. Amongst others, Ahn *et al.* [19]'s design on wireless sensor networks incrementally infers an appropriate RS code rate based on the channel condition with a lower code rate when packet loss rate increases and a higher code rate for an error free zone. The design had four discrete levels, the RS(106,100), RS(112,100), RS(118,100), and RS(126, 100).

The above works, though, focused on noisy wireless networks but were not explicitly concentrating on CubeSat communications to cater for computational and bandwidth constraints. For example, implementing the algorithm proposed by Ahn *et al.* [19] would be heavy on real estate cost on dedicated hardware. Most of the designs are simulated at a higher level C implementation [19][17], which could be optimized when implemented on hardware. A Field Programmable Gate Array (FPGA) implementation of a hybrid Reed Solomon decoder was implemented by Shimiz *et al.* [22] which is reconfigurable with four RS codes, RS(255,253), RS(255,251), RS(255,249), RS(255,247) with error correcting capabilities of 1, 2, 3, and 4 respectively. The strategy was to use the Peterson algorithm in a shared matrix solver technique for the four codes to find the error locator polynomial. However, the Peterson algorithm is not the best in terms of efficiency in finding the error locator problem [23].

At the Electronic Systems Laboratory (ESL), Le Roux [1] in his work on a satellite network simulator, proposed a hybrid protocol of AX.25 and FX.25, which he called AFX.25. The principle of operation of AFX.25 is that, the FX.25 accounts for communication intervals of low signal to noise ratio and for high signal to noise ratios, the protocol state switches to AX.25. The choice of AX.25 is justified considering the 2015 survey on communication systems for CubeSats done by Klofas [9] which enlisted the AX.25 as the first amongst the non-proprietary protocols. The second larger share belongs to the CCSDS protocols at 6.06%. The numbers at a glance can be deceiving without a close investigation on the protocols. It was found that the Cubesat Space Protocol(CSP) uses the CCSDS standards at its layer 2[24]. Therefore, the CCSDS has a share of 6.06% when the 2.61% for the CSP is added. It was followed by the Internet Protocol over Digital Video Broadcasting via Satellite Second Generation (IP over DVB-S2), Mobitex, Cubesat Space Protocol (CSP) protocols and others as shown in Table 1.1.

Protocols	Percentage of CubeSats
AX.25	52.17 %
AX.25 plus CW beacon	25.22 %
IP over DVB-S2	5.22 %
CCSDS	3.48 %
Custom	3.48 %
CSP	2.61 %
Mobitex	2.61 %
CW	1.74 %
AX.25/SRLC plus beacon	0.87 %
AX25/Mobilex	0.87 %
FX.25	0.87 %
NSP	0.87 %
HDLC	0.87 %
CSP/CW	0.87 %
TI	0.87 %

Table 1.1: CubeSat Non-proprietary Communication Protocols Summary. Adapted from the CubeSat Communication System Table, Appendix A by Klofas [9]

1.3 Research Question

Le Roux [1] showed experimental results which indicate that FX.25 in a hybrid setup with AX.25 can have an improved overall throughput. Is there anything better than AX.25/FX.25 combination? If the experiment done by Le Roux [1] is to be considered, how feasible is that implementation on hardware?

1.4 Research Statement

As observed from Table 1.1, a couple of studies[25][9] suggest that AX.25 is trendy amongst CubeSat developers. It is worth noting though that the CCSDS standard protocols also have their share not only on small satellites but also on the traditional spacecrafts. This fact is derived from considering that the CCSDS agency provide their protocols as standards which might grow in numbers amongst CubeSats in the years to come. Due to its architecture [26] which is complex in nature, the CCSDS is not suited for smaller designs. In this work, an AX.25/FX.25 adaptive FEC protocol is designed and implemented on a dedicated Fusion MFS1500 FPGA board.

1.5 Research aim and objectives

The overall aim of the project was to design and implement a FPGA based FEC adaptive data link protocol for CubeSats.

Thus, the objectives of the research are:

- (i) Evaluating the CCSDS TC against the AX.25/FX.25/AFX.25 performance over a satellite pass.
- (ii) Implement an adaptive version of the better performer between the AFX.25 or the CCSDS TC protocol on a FPGA.
- (iii) Experimentally validate the hybrid protocol's performance and cost of implementation.

1.6 Design requirements

With the project aim and objectives outlined, there are design requirements which are based on the nature of this project. This work falls under the satellite research at ESL and as an educational based project, the following requirements are implied:

- CubeSat standard: The first requirement was that the communication module must be miniaturised to ensure that the whole CubeSat is within the standard volume and weight. Secondly, the power utilisation of the module must be within the CubeSat power supply allocation.
- Amateur radio focus: Much like South Africa's first satellite which was developed at Stellenbosch University's Electronic Systems Laboratory (ESL) laboratory and launched in 1999 [27] (which used amateur radio communications), this project aimed at making use of the amateur radio community intellectual resources as much as possible.
- Open standard: Although there are various protocols available, this work was confined to open standards to reduce financial barriers and increase accessibility.
- FPGA design: The design and implementation of the protocol was aimed at reaping the benefits of system-on-chip designs.

The project is constrained to improving the performance of communication for CubeSats from a layer 2 design perspective utilising error coding and control. At protocol level, the data link protocol has the potential of adding an improvement in performance through error correction.

As a limitation, the project testing will be performed by computer simulation and on a terrestrial equivalent environment which may compromise the exactness of the protocol performance from what it would be in outer space.

1.7 Value of project

The research work will contribute to the improvement of the CubeSat communication sub-system design by providing recommendations for an adaptive FEC protocol, which brings about an improvement in the overall throughput per session. As per the design requirements, the development recommendations will be more applicable to CubeSat projects.

1.8 Overview of this work

Chapter 1: Introduction

This chapter provides an introductory section of the work; the problems faced in CubeSat communications are highlighted, the project scope is narrowed to adaptation only with error correction schemes, the current research relating to adaptive FEC and CubeSat protocols is discussed. In response to the research question(s), the AFX.25 was found to be a better choice which leads to the AFX.25 adaptive design. The project aims were outlined, with design requirements and finally the project contribution.

Chapter 2: Simulation Environment

This introduces the simulation tools for a satellite network. A choice of SatSim is made as a simulation tool for this project after a comparison with other network simulators. The chapter ends with a discussion of the theory of a communication link budget, then how it was designed and implemented on SatSim.

Chapter 3: Protocol Simulation and Evaluation

The chapter on protocol simulation compares the protocol performances of the AFX.25 and the CCSDS TC which determines the protocol that is implemented on hardware in the next chapter.

Chapter 4: Adaptive AFX.25 Implementation

The chapter on protocol simulation and evaluation suggested that the AFX.25 was a better choice than the CCSDS TC. This chapter first introduces background literature for FX.25 correction codes, then the protocol design, and implementation on FPGAs. The chapter ends with discussion of results from the implementation.

Chapter 5: Conclusion and Future Work

Having covered the main aspects of the work, chapter 5 brings a summary of what has been achieved, and suggests what aspects of the project can be improved as part of future work.

Chapter 2

Simulation Environment

This chapter introduces the simulation tool that is used in the protocol simulation in the next chapter and key fundamentals for channel estimation and its modelling in the simulation environment.

2.1 Communication Simulation

2.1.1 Simulation Background

It was necessary that a satellite network simulation framework be selected for the analysis. The satellite network simulator in question must provide an environment of propagating satellite objects while allowing network communication between the dynamic node(s) and stationary ground stations. The network framework must allow for easy 'plug and play' addition of protocols and the customisation of existing ones. Moreover, event based frameworks/simulators are better estimators of realistic network performance because events are generated and executed based on the state of the preceding events instead of timed events. Several tools are available both on an open source and commercial basis ranging from frameworks to fully fledged simulators offering different features.

2.1.1.1 Network Simulator (NS3)

A series of network simulators under the codename *ns* have been released over the years from ns-1, ns-2, ns-3 under the public licence [28]. The ns3 which is the current active release is a discrete computer network simulator written in C++ which supports Internet Protocol (IP) and non-IP networks [29]. The ns3 does not officially support satellite communications, a third-party plug-in, the satellite network simulator, sns3[30] can be considered for satellite simulations. The challenge with the ns is that its scripting language is very complex which present a time-consuming task to master.

2.1.1.2 OMNET++

A similar public source discrete network simulator written in C++ is OMNET++[31]. It is a component-based simulation tool that allows network components to be assembled into larger network components with a network description language (NED) that separates the network interface design from the underlying functionality implementation on C++ class files. The challenge to using OMNET++ is that it is not a dedicated network simulator which presents a steep learning curve for new developers.

2.1.1.3 OPNET IT Modeller

OPNET IT Guru is another discrete event network simulator provided under a commercial licence[32]. OPNET provides a powerful graphical support for users to create network entities and topologies whilst providing a programming structure for advanced users. The company that develop the product provides an academic version of the simulator which has a certain maximum number of allowed events in a simulation.

2.1.1.4 SatSim

SatSim is a satellite network simulation tool developed by Le Roux [1] for his Master's work at the ESL. It is an event driven simulator written in Python based on the Simpy[33] event framework. This brings about ease in terms of simulation setup time since Python has an intuitive syntax and a broad developer community. The event driven model is superior to a time driven simulation because data packets are generated, queued and transmitted not at intervals but using the concept of queuing theory. An event only occurs when the previous one has been executed completely and pushed out of the queue stack. The spacecraft objects can be propagated using the PyEphem propagator or the Python-SGP4 propagator. The PyEphem[34] is a Python port of the XEphem astronomical package which was originally written in C. The Python-SGP4 package is a pure Python implementation of the SGP-4.

SatSim was selected as a choice for simulation because unlike the other simulators, it comes pre-packaged with the AX.25, FX.25 and AFX.25 protocols and allows for an easy addition of new protocols.

2.1.2 SatSim Architecture

SatSim has a modular structure which is presented below:

- **Graphical User Interface (GUI):** SatSim provides a GUI which can be used to configure simulation parameters or to visualise an animated simulation. The configuration allows for each node design at a physical layer (antenna parameters, modulation scheme, frequency, elevation, propagation models, etc), at data link layer (the coding protocols) and at application layer (data generation and scheduling).

- **Debug:** During a simulation, the state of a network (transmission and reception) can be viewed from the console output.
- **Data Logging:** The packets transferred, lost, and received with and without error are logged in a file which can be exported to a .csv file.
- **Graphing:** SatSim provides a graphing utility driven by the Matplotlib library for performing data analysis during or after the simulation.
- **RF Channel:** To properly model a wireless scenario, the simulator provides the RF channel module which propagates the packets. The packets/bits will be corrupted based on the random noise generator, the current signal to noise ratio computed from the link budget, and the error rate.
- **Node Objects:** Each spacecraft/ground station is modelled as a simulation object with propagation model and an elevation map. The propagation model propagates the satellite in simulated time and provides the satellite's positional information for the RF Channel module. Each node models the physical characteristics of the transmitter/receiver, the antenna properties, the modulation schemes, as well as the routing and coding schemes.

2.1.3 SatSim Programming Interface

The previous subsection 2.1.2 introduced a simulation option at a higher level, the GUI. For more detailed simulations, SatSim provides a scripting interface that allows for more customisation of simulations. The scripting allows for more control of the simulation parameters which can speed up the simulation, for example, allowing the simulation processes to be distributed among available processor cores in a multi-core computer. The scripts are written in Python. The scripting interface also allows for creating of new libraries. The simulator is shipped with a package of pre-assembled scripts that can be modified to suit certain simulation requirements. Specifically, for this work, SatSim provides libraries for encoding and decoding AX.25, FX.25 and AFX.25 protocols. The protocol classes are sub-classes of the base protocol super class which provides generic protocol definitions and functions.

2.2 Link Budget Simulation

2.2.1 Link Budgeting Theory

The basis for performance comparison for the protocols is the data throughput, which is a function of the quality of service, the Bit Error Rate (BER) which is dependent on the modulation scheme and the signal-to-noise ratio.

A selection of RF modulation schemes with corresponding BER equations are given in Table 2.1.

Table 2.1: RF Modulation Schemes and BER equations. Adapted from Ref [10]

Modulation Scheme	Description	BER	Bit Per Symbol	No Of Symbols
BPSK	In BPSK, two waveforms which are 180° out of phase are used to represent either a 0 or 1	$\left(\frac{1}{2}\right) \operatorname{erfc}\left(\sqrt{\frac{E_b}{N_o}}\right)$	1	2
QPSK	The QPSK has 4 symbols which are modulated using 4 waveforms which are $\pi/2$ rad out of phase	$\left(\frac{1}{2}\right) \operatorname{erfc}\left(\sqrt{\frac{E_b}{N_o}}\right)$	2	4
M-ary PSK	The number of symbols, M is greater than 4. Each symbol is modulated by a waveform that is shifted $2\pi/M$ out of phase	$\left(\frac{1}{k}\right) \operatorname{erfc}\left[\sqrt{k\frac{E_b}{N_o}} \sin\left(\frac{1}{M}\right)\right]$	k	M
FSK	M waveforms of different frequencies are used to represent digital data.	$\left(\frac{M-1}{2}\right) \operatorname{erfc}\left(\sqrt{\frac{k}{2}\frac{E_b}{N_o}}\right)$	k	M
MSK	The MSK waveform can be represented as an FSK with two signalling frequencies to represent data. It has a BER like that of QPSK	$\left(\frac{1}{2}\right) \operatorname{erfc}\left(\sqrt{\frac{E_b}{N_o}}\right)$	k	M
ASK	Equally spaced amplitude waveforms are used to transmit digital data. If the system is coherent (phase-locked to the received signal), it can operate as a unipolar, detecting only positive amplitudes or bipolar which detects positive and negative amplitudes.	Unipolar BER, $\frac{1}{k} \frac{M-1}{M} \operatorname{erfc}\left(\sqrt{\frac{3k}{2(M-1)(2M-1)}\frac{E_b}{N_o}}\right)$ Bipolar BER, $\left(\frac{1}{k}\right) \left(\frac{M-1}{M}\right) \operatorname{erfc}\left(\sqrt{\frac{3k}{(M^2-1)}\frac{E_b}{N_o}}\right)$	k	M

erfc is the complementary error function[35] defined by

$$\text{erfc}(x) = \frac{2}{\sqrt{\pi}} \int_x^{\infty} e^{-t^2} dt \quad (2.1)$$

The E_b/N_o is the energy per bit to noise power density ratio, which determines how strong the received signal is and is given by the equation[15]:

$$\frac{E_b}{N_o} = \frac{PL_lG_tL_sL_aG_r}{kT_sR} \quad (2.2)$$

where

P = transmitter power,

L_l = transmitter to antenna loss,

G_t = transmitter antenna gain,

L_s = free space loss,

L_a = transmission path loss,

G_r = receiver antenna gain,

k = Boltzmann's constant,

T_s = noise temperature,

R = data rate

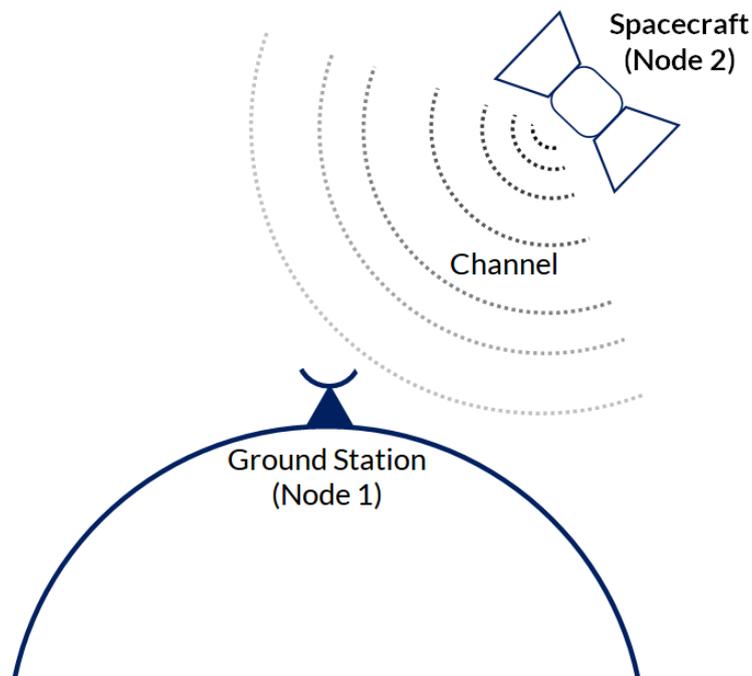


Figure 2.1: Basic satellite-ground communication configuration

The *free space loss*, L_s depict the losses due to spreading of the signal from node 2 to node 1 [36] as shown in Figure 2.1, thus is given by the relation:

$$L_s = \left(\frac{4\pi d}{\lambda} \right)^2 \quad (2.3)$$

where

d = range between the 2 nodes

λ = wavelength of the signal

and,

$$\lambda = \frac{c}{f} \quad (2.4)$$

where

c = speed of light

f = signal carrier frequency

The space-earth link can be degraded by atmospheric conditions which include absorption in atmospheric gases, scattering and depolarization by water, precipitation ice droplets, slow fading, and attenuation by local environment[37]. The International Telecommunications Union (ITU) provides an algorithm for calculating the the losses due these effect. However, for this project the *path loss* is assumed at 0 dB because these effects are critical for frequencies above 1 GHz. Rain, fog and clouds can also weaken the signal strength especially at high frequencies[37], but it is not easy to model weather conditions for the dynamic link. The project will assume clear weather conditions.

The antenna gain[15], G is defined by

$$G = \eta \left(\frac{4\pi}{\lambda^2} \right) A \quad (2.5)$$

where

η = efficiency of the antenna

λ = signal wavelength (m)

A = area of the antenna (m^2)

$$\lambda = \frac{c}{f} \quad (2.6)$$

where

c = speed of light

f = signal carrier frequency

The system noise temperature, T_s accounts for all the noise temperature from thermal radiation of objects within the range of view of the antenna, from losses in the antenna and within active and passive components of the receiver[15]. It is generally defined as

$$T_s = (T_A \times L) + ((1 - L) \times T_L) + ((F - 1) \times T_0)$$

where

T_A = antenna noise temperature

T_L = feed and other passive components noise temperature

L = antenna feed loss

F = receiver noise figure

T_0 = reference temperature = 290 K

2.2.2 Link Budget Design

The RF channel link budget was designed to be as follows[1]

From equation 2.2, the transmitted power (P) and the data rate (R) are inputs to the simulation. The antenna gains are dynamically computed within the simulation with the aid of antenna gain maps which evaluates the gain value from the map using the current position of the satellite relative to the ground station[1]. Each gain map graphic represents an antenna radiation pattern in form of a grey-scale pixel image. Each pixel color is mapped to a different gain value. The full antenna mapping algorithm can be found in Ref. [1].

Combining equations 2.3 and 2.4, and substituting $c = 3 \times 10^8$, L_s becomes

$$L_s = \left(\frac{4\pi df}{c} \right)^2 = 20 \log(d) + 20 \log(f) - 147.55 \text{ dB} \quad (2.7)$$

From ref. [15], $L_l = 0.5 \text{ dB}$. From ref. [1], $T_s = 412.037 \text{ K}$, $k = 1.38066 \times 10^{-23}$ and polarisation and implementation losses, $L_{add} = -5 \text{ dB}$.

The link budget is thus,

$$\begin{aligned} \frac{E_b}{N_0} &= P_{\text{dB}} + L_l \text{ dB} + G_t \text{ dB} + L_s \text{ dB} + L_a \text{ dB} + G_r \text{ dB} + 10 \log(1.38066 \times 10^{-23}) \\ &+ 10 \log(412.037) + 10 \log(R) + L_{\text{add}} \\ &= 10 \log(P) + 0.5 \text{ dB} + 10 \log(G_t) - 20 \log(d) - 20 \log(f) + 147.55 \text{ dB} - 0 \text{ dB} \\ &+ 10 \log(G_r) - 10 \log(R) - 10 \log(1.38066 \times 10^{-23}) - 10 \log(412.037) - 5 \end{aligned} \quad (2.8)$$

2.2.3 Error Modelling

The channel is modelled as a Binary Symmetric Channel (BSC) whereby the probability of a bit being flipped is not dependent on previous occurrences[7]. The reception of bits is thus a memoryless model which takes one bit at a time. A bit T_i^b is propagated through the channel with a probability of being flipped, p and is received correctly with probability $1 - p$.

The received bit, R_i^b is given by:

$$R_i^b = T_i^b + N_i^b \quad (2.9)$$

where $R_i^b, T_i^b, N_i^b \in \{0, 1\}$. N_i^b is the channel noise which can be inflicted on the original bit via a modulo 2 addition[38]. The likelihood function given by,

$$P(R_i^b | T_i^b) = \begin{cases} 1 - p & \text{if } R_i^b = T_i^b, \\ p, & \text{if } R_i^b \neq T_i^b, \end{cases} \quad (2.10)$$

gives a good error approximation at bit level. Given a communication scenario, one may be interested in the number of corrupted bits, k in a packet of n bits. Such a sequence $\{N_i, i \in \mathbb{Z} \geq 0\}$ is a Bernoulli random process where a bit is either in error or not[39].

Therefore, the probability that k bits are in error in a n -sized packet is given by:

$$P(k \text{ bits in error in } n \text{ bits}) = \frac{n!}{k!(n-k)!} p^k (1-p)^{n-k} \quad (2.11)$$

2.2.4 Implementation on SatSim

To generate channel noise, the value of the link budget equation 2.8 must be computed and combined with the bit error rate equation selected from Table 2.1. The other variables of equation 2.8 are based on the simulation configuration except the antenna gains and the range between the two nodes which are dynamic throughout the simulation. The propagation model computations play a key role in computing the two categories of variables. Antenna gain maps are represented as gain maps in the simulator which are attached to every transmitter and receiver. The elevation and azimuth computed from the spacecraft's position vector relative to the ground station is used to map the right pixel in the gain map. Depending on the colour of the pixel, the antenna gain is calculated. Each ground station has an elevation map to model obstacles which, when combined with the node's elevation azimuth, determine whether there is a line of sight. Details on the computation algorithms can be found in Ref. [1]. The range R is given by the difference in position between the communicating nodes.

Two possible noise generation techniques were considered, a packet level and a bit level invalidation. The packet level noise generation corrupts a packet based on the packet error rate. This approach is well suited for bigger simulations as it is time-efficient though it is not necessarily a good representation of channel noise. The second technique which was adapted for the simulations, was the bit level flipping which uses the BER to invalidate a bit. This gives a much finer approximation of a realistic channel although it is computationally heavy.

Chapter 3

Protocol Simulation and Evaluation

3.1 Introduction

As per the first design objective, this chapter evaluates the CCSDS TC, AX.25/FX.25 performance over a satellite pass. The CCSDS protocol stack comprises of three main data link layer protocols, the Telemetry(TM), the Telecommand(TC) and the Advanced Orbiting Spacecraft (AOS) protocols[26]. The TC was a choice for simulation over the TM and AOS due to its minimalistic implementation and its similar functionality to FX.25. This chapter's investigation should enable the identification of the protocol that yields a better throughput that will lead to modifying it to adapt dynamically throughout the changing channel conditions.

3.2 AX.25

3.2.1 AX.25 Definition

The AX.25 protocol is an amateur radio protocol that ensures link layer compatibility between stations irrespective of the upper layer employed. Of interest to this work, are the data link layer and the physical layer as depicted in Figure 3.1. The **Segmenter** breaks units of

Layer	Function(s)	
Data Link (2)	Segmenter ^(DLSAP)	Management
	Data Link	Data Link
	Link Multiplexer	
Physical (1)	Physical	
	Silicon/Radio	

Figure 3.1: AX.25 Protocol Stack. Adapted from AX.25 Specification Report [2], pg.2

data into smaller units if they exceed the maximum number of allowed information units for the frame for transmission. On the receiver, the de-segmenter reassembles the segments, and delivers them to upper layers[2].

The **Management Data Link** provides all the necessary parameter negotiation required in a communication session [2] between two nodes.

Multiple data links are multiplexed by the link **Link Multiplexer** [2] so they can share the same RF channel.

AX.25 offers three general types of frames, the information frame (I frame), supervisory frame (S frame) and the Unnumbered frame (U frame)[2]. Each frame is made up of the fields as shown in Figure 3.2.

Flag	Address	Control	Info	FCS	Flag
01111110	112/224 Bits	8/16 Bits	N*8 Bits	16 Bits	01111110

U and S frame construction.

Flag	Address	Control	PID	Info	FCS	Flag
01111110	112/224 Bits	8/16 Bits	8 Bits	N*8 Bits	16 Bits	01111110

Information frame construction.

Figure 3.2: AX.25 Frame types. Adapted from AX.25 Specification Report [2], pg.6

Flag Field: The flag field is an 8-bit long field that delimits the start and end of a packet. In a multi packet transmission, one delimiter flag can be used to mark the end of a frame and the beginning of another. The delimiter sequence is 01111110[2].

It is therefore not expected that a similar pattern occurs anywhere along the train of bits. If the data contains the pattern, then the encoder performs bit stuffing. Bit stuffing inserts a '0' bit after every contiguous five '1's. This implies that the receiver performs destuffing whereby any '0' detected after five contiguous '1's is discarded.

Address Field: As the packet traverses through the network, the address field aids the packet routing by providing the packet's source and destination addresses, and optionally Layer 2 repeater subfields. Each subfield contains the amateur callsign and the Secondary Station Identifier (SSID) [2].

In a non-repeater mode, the destination address is the callsign and SSID of the station which the frame is destined for. Similarly, the source address will contain the source callsign and the SSID of the sender.

The Layer 2 repeater allows more than one repeater to share the same RF channel. This is achieved by appending an additional address subfield at the end of the address field.

Control Field: The 1 or 2-byte Control field provides information about the type of the frame. If bit 0 is '0', the frame is an I frame. If bits [0-1] are '01' or '11', the frame is an S or U frame respectively. The other bits [2-7] or [2-15] contain the Poll/Final (P/F) bit and send and receive sequence numbers for the S and I frames, whereas, for a U frame, the bits [2-7] or [2-15] are the P/F bit and the modifier bits. The P/F bit is used a poll (request for a reply to a frame) or a final (reply to the poll).

The **Information** frame is a general frame for sending payload data.

The **Supervisory** frame provides acknowledgement, retransmission and window control information.

The **Unnumbered** frame contains link maintenance data. Some U frames may have information and PID fields.

Protocol Identifier (PID) Field: As the AX.25 is compatible with other high layer protocols, the PID field which is only on I and UI frames stores information about the layer 3 protocol implemented. A list of compatible layer 3 protocols is given in the Appendix, Table B.1.

Information(I) Field: The user payload data is stored in the Information field of the frame whose size is variable up to 256 bytes. The I field is found in I, UI, XID, TEST and FRMR frames.

Frame Check Sequence: Since data is transmitted between two different nodes, during which the data may incur errors along the channel path, the Frame Check Sequence (FCS) field contains a 16-bit Cyclic Redundancy Check (CRC) code for data validation. The FCS is transmitted with the most significant bit first.

All other fields are transmitted with the least significant bit first[2].

A frame is invalid if its length is less than 136 bits including the delimiter tags, or if it is not enclosed by the delimiters or if it does not have an integral number of octets.

The AX.25 can be operated in a connectionless mode where frames cannot be acknowledged in a frame format known as the Unnumbered Information frame (UI frame). This is achieved by setting the Control Field bits to correspond to the UI frame type. For full details on the AX.25, refer to [2].

Retransmission Mechanism: Both the sender and receiver maintain counters for sequence numbers from within the retransmission modules. Five state variables/numbers are used in the retransmission algorithm, the Send State Variable V(S), Send Sequence Number N(S), Receive State Variable V(R), Receive Sequence Number N(R) and the Acknowledge State Vari-

able $V(A)$. $V(S)$ contains the number to be assigned to the next I frame. $N(S)$ contains the sequence number of the I frame transmitted[2]. $V(R)$ is the next expected frame's sequence number. The $N(R)$ sequence number is updated before sending an I or S frame to be equal to the $V(R)$. If the $N(S)$ stored in the control field does not match the $V(R)$ of the receiver station, an error has occurred. A Selective Reject (SREJ) command is then transmitted in supervisory frame format to the transmitter to request a retransmission. This condition is cleared upon successful receipt of the requested I frame[2]. The $V(A)$ contain the sequence number of the last acknowledged frame.

3.2.2 AX.25 Implementation

For most CubeSat missions, the need for simplicity and reducing overheads is key, thus the connectionless version of AX.25 was implemented in SatSim[1]. The connectionless mode does not guarantee arrival of packets as it does not use of any error recovery methods, therefore the ARQ functionality is not used in this mode. In the connectionless mode, Unnumbered Information(UI) frames are used. The UI frame is shown in Figure 3.3.

Flag	Address	Control	PID	Info	FCS	Flag
01111110	112/224 Bits	8/16 Bits	8 Bits	N*8 Bits	16 Bits	01111110

Figure 3.3: AX.25 UI frame. Adapted from Ref. [2]

To use the UI frame, the Control field is set to 0x03 in the 8-bit mode according to the configuration given by Table 3.1. The 16 bit FCS is generated using the CRC-16-CCITT generator polynomial,

$$g(x) = x^{16} + x^{12} + x^5 + 1$$

Control Field Type		Type	Control-Field							
			7	6	5	4	3	2	1	0
Set Async Balanced Mode	SABME	Cmd	0	1	1	P	1	1	1	1
Set Async Balanced Mode	SABM	Cmd	0	0	1	P	1	1	1	1
Disconnect	DISC	Cmd	0	1	0	P	0	0	1	1
Disconnect Mode	DM	Res	0	0	0	F	1	1	1	1
Unnumbered Acknowledge	UA	Res	0	1	1	F	0	0	1	1
Frame Reject	FRMR	Res	1	0	0	F	0	1	1	1
Unnumbered Information	UI	Either	0	0	0	P/F	0	0	1	1
Exchange Identification	XID	Either	1	0	1	P/F	1	1	1	1
Test	TEST	Either	1	1	1	P/F	0	0	1	1

Table 3.1: AX.25 Control Field. Adapted from Ref. [2]

The full details of the implementation can be found in the work "Development of a satellite network simulator tool and simulation of AX.25, FX.25 and a hybrid protocol for nano-satellite communications" by Le Roux[1].

3.3 FX.25

3.3.1 FX.25 Definition

As a means of facilitating error correction with AX.25, the amateur community introduced a backward compatible extension protocol wrapper on the AX.25 frame, FX.25[3]. An FX.25 frame is constructed from an AX.25 frame by pre-adding a preamble and correlation tag fields and appending a pad field, FEC check symbols, and a postamble field[3]. A FX.25 frame is backward compatible with the AX.25 in that any station with AX.25 decoder but not FX.25 can decode the frames since the preamble and postamble will be discarded. A basic FX.25 frame structure is shown in Figure 3.4.

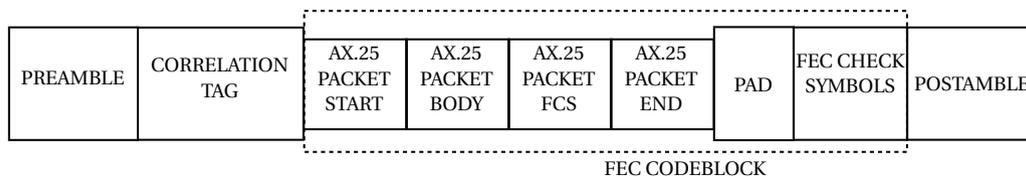


Figure 3.4: FX.25 basic frame format. Adapted from FX.25 Specification Report [3], pg.3

Preamble: The preamble block is a sequence of 01111110 bytes to enable receiver synchronisation[3]. A minimum of 4 bytes of the sequence is required.

Correlation Tag: A 64-bit correlation tag is used to indicate which FEC algorithm is used and marks the start of a frame. Gold codes are used to generate the correlation tags, ensuring a favourable auto- and cross-correlation. A list of correlation tags is shown in Table 3.2.

Table 3.2: FX.25 Correlation Tag Values. Adapted from FX.25 Specification Report [3], pg.7

Tag	Correlation Tag Value	FEC coding algorithm, number of information bytes available
Tag_00	0x566ED2717946107E	Reserved
Tag_01	0xB74DB7DF8A532F3E	RS(255, 239) 16-byte check value, 239 information bytes
Tag_02	0x26FF60A600CC8FDE	RS(144,128) - shortened RS(255, 239), 128 info bytes
Tag_03	0xC7DC0508F3D9B09E	RS(80,64) - shortened RS(255, 239), 64 info bytes
Tag_04	0x8F056EB4369660EE	RS(48,32) - shortened RS(255, 239), 32 info bytes
Tag_05	0x6E260B1AC5835FAE	RS(255, 223) 32-byte check value, 223 information bytes
Tag_06	0xFF94DC634F1CFF4E	RS(160,128) - shortened RS(255, 223), 128 info bytes
Tag_07	0x1EB7B9CDBC09C00E	RS(96,64) - shortened RS(255, 223), 64 info bytes
Tag_08	0xDBF869BD2DBB1776	RS(64,32) - shortened RS(255, 223), 32 info bytes
Tag_09	0x3ADB0C13DEAE2836	RS(255, 191) 64-byte check value, 191 information bytes

Tag_0A	0xAB69DB6A543188D6	RS(192, 128) - shortened RS(255, 191), 128 info bytes
Tag_0B	0x4A4ABEC4A724B796	RS(128, 64) - shortened RS(255, 191), 64 info bytes
Tag_0C	0x0293D578626B67E6	Undefined
Tag_0D	0xE3B0B0D6917E58A6	Undefined
Tag_0E	0x720267AF1BE1F846	Undefined
Tag_0F	0x93210201E8F4C706	Undefined
Tag_10		
to	Undefined	Undefined
Tag_3F		
Tag_40	0x41C246CB5DE62A7E	Reserved

Pad: If the AX.25 packet does fit within the codeblock's information length, the AX.25 is padded with a sequence of the pad byte 0x7E. If the AX.25 is not byte aligned, then the last bits of the non-byte aligned octet will be filled with the most significant bits of 0x7E.

FEC Check Symbols: The FEC field contains redundancy bits necessary for error recovery of corrupted bits on the receiver.

FEC Codeblock: The FEC codeblock comprises the AX.25 packet (unaltered), padding bytes (in the case where the length of the AX.25 packet and the number of FEC check symbols is less than the required frame size for the code used. Based on the AX.25 frame and the algorithm used, the FEC check symbols are generated[3]. The FX.25 error-correcting scheme is based on variants of the Reed-Solomon codes.

Postamble: The FX.25 frame is delimited by a postamble which is a sequence of 01111110 bytes[3].

The FX.25 bytes are transmitted in least significant bit first mode. The full specification reference for FX.25 can be found ref [3].

3.3.2 FX.25 Implementation

In creating the FX.25 packet, the FX.25 developers[3] suggested a step-wise approach whereby the packets are first bit stuffed, then passed to the Reed encoder to generate the parity bits and then pushed to the RF channel.

Le Roux's[1] implementation of the packet generator begins with 4 bytes of the sequence 0x7E forming the Preamble. A rolling window is used to detect the correlation tag. The RF bits are pushed into a 64-bit memory, if the first byte is not recognised as a valid correlation tag, it is chopped out and the bits are shifted to the left. The correlation algorithm requires a 90% matching of the received and expected correlation tag and after that the incoming train of bits can be saved into the packet buffer. The packet is delimited by a 2-byte sequence of

0x7E, the postamble.

Implementing the Reed Solomon algorithm as is on SatSim can increase in the overall simulation time due to its complexity. So, a modified approach for the error correction strategy was adopted[1]. The receiver node computes the Hamming distance of the sent and received bytes, and if the distance is greater than t for a t correction code, the packet is dropped, otherwise the packet is fixed. Fixing the packet assumes the original packet that was transmitted which is stored in the Tx copy buffer. This procedure is sufficient in a configuration where the priority is to determine the performance of the code but not considering the amount of time each code takes to correct the errors. This configuration is sufficient for this chapter as the hardware design implements the encoding and decoding algorithms as per the Reed Solomon specification.

3.4 AFX.25

3.4.1 AFX.25 Definition

AFX.25 is a hybrid of the AX.25 and the FX.25 protocols that switches to the FX.25 protocol for noisy RF channels else it defaults to AX.25. The basic frame and error correcting control mechanisms for AFX.25 uses the specifications for AX.25 and FX.25 as outlined in subsections 3.2.1 and 3.3.1 respectively.

Given that the protocol can operate as either AX.25 or FX.25, the protocol modes are represented as states, a and f for AX.25 and FX.25 respectively. The modes are particular to the transmitter to determine which packet format to encode. The state change is driven by the channel parameters from the receiver node. Both the transmitter and receiver manages timers at 20-second intervals. The 20 seconds is an experimentally chosen interval after several tests motivated by 2 reasons. Firstly, the interval must be wide enough not cause an unstable behaviour where the channel conditions are rapidly changing and the protocol mode is continually changing to and fro between the protocols. The second reason is that the window is not supposed to be too wide such that significant changes in the channel could occur and be cancelled out without being detected by the algorithm. On timer expiry, the receiver node computes the channel performance parameter and sends a switching command to the transmitter.

Assuming that the protocol mode is AX.25 at the interval t , the probability of the protocol switching to FX.25 is given by $1 - a$ and continues in AX.25 mode with probability a . Similarly, the FX.25 mode will be retained for the next interval with probability f but changes to AX.25 with probability $1 - f$. The transitions are summarised in Figure 3.5.

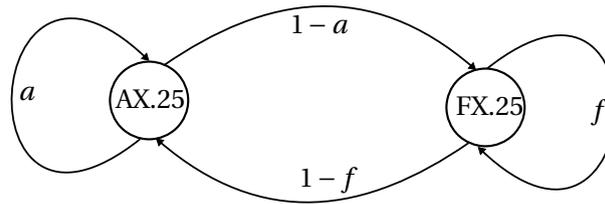


Figure 3.5: Protocol Transition States

If the mode is AX.25, the receiver calculates the packet loss rate given by:

$$\text{Packet loss rate, } l_a = \frac{\text{Packets received in error in } t}{\text{Total packets received in } t} \quad (3.1)$$

If the packet loss rate is above than a set threshold, a switch command is sent to the sender to switch to FX.25. The protocol packet loss threshold can be adjusted depending on the quality of service required for the mission.

For the FX.25 mode, the parameter of interest is the packet success rate

$$\text{Packet success rate, } r_s^{(a)} = \frac{\text{Error free packets received in } t}{\text{Total packets received in } t} \quad (3.2)$$

The parameter is evaluated within the FX.25 state machine whereby, if the success rate goes above the threshold, the channel is considered to have a sufficient link margin for AX.25.

3.4.2 AFX.25 Implementation

For a typical satellite pass, whose trajectory moves from acquisition to loss of signal, the transmitter will transmit in an error recoverable state, the FX.25 mode, until a point where the two nodes agree to switching to AX.25[1]. The receiver maintains a sliding window whereby the number of dropped, corrected and error free packets received are logged. If the drop rate falls below the threshold, the receiver sends a command frame to the transmitter negotiating a switch to AX.25. The threshold must be chosen carefully so that maximum throughput is achieved. A performance evaluation of different thresholds was necessary. Figures 3.6a and 3.6b show the switching points for randomly selected success rate thresholds of 0.6 and 0.95.

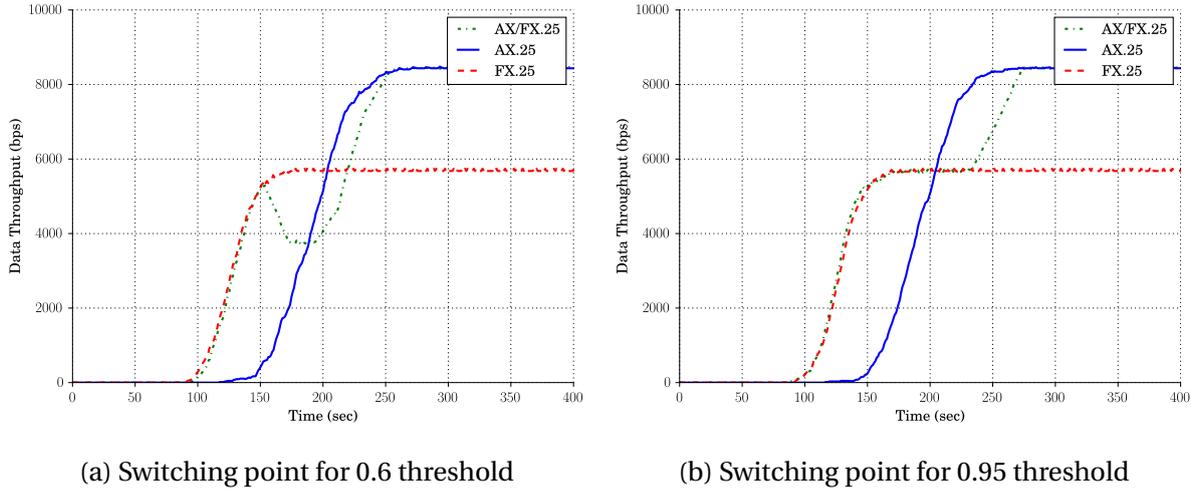


Figure 3.6: Throughput Curves for different BER for a 156 byte payload data

The total data received for each threshold for switching from FX.25 to AX.25 is given in Table 3.4.

Threshold	AX.25 Data Received (KB)	AFX.25 Data Received (KB)	AFX.25 Improvement over AX.25 (%)
0.6	319.96	338.74	5.87
0.95	319.39	344.53	7.87

Table 3.4: AX.25/AFX.25 Data Received over 400 seconds for different success rate thresholds

The 0.6 threshold causes the protocol to switch to FX.25 prematurely when the channel's probability of losing is still significant. This causes a drop in throughput as observed from Table 3.4. The 0.95 threshold shows a better performance even though the protocol does not switch immediately when the AX.25's throughput becomes more than the FX.25's.

Le Roux [1] implemented the decoding of packets in a layered-processing manner. If the packet contains errors, it is forwarded to the FEC recovery module for error correction. However, for an AX.25 packet, if the packet was invalidated by channel noise, it cannot be recovered since it has no FEC information bits. Such a packet will be dropped by the decoder. Error free packets before or after error correction are moved to higher layers.

3.5 CCSDS Telecommand(TC)

3.5.1 Telecommand Definition

The TC protocol is part of the CCSDS space communications protocols reference model[26]. The TC can be used for ground-to-space or space-to-space communication links. The TC protocol layer accepts data from upper layers, segment larger packets to reduce the proba-

bility of the packets being invalidated by channel noise. The small packets serve well in preserving the bandwidth in the case when the packet needs to be retransmitted, since only the smaller segment will be retransmitted instead of the original big packet[4]. To avoid smaller segments, data units that are smaller than the segment information limit are blocked to form the standard segment size. The TC employs two strategies of error recovery which are 'go-back-n' type retransmissions by the Communications Operation Procedure(COP) and the BCH error correction handled by the Synchronisation and Channel Coding sublayer.

The protocol performs its services in layered module architecture as shown in Figure 3.7.

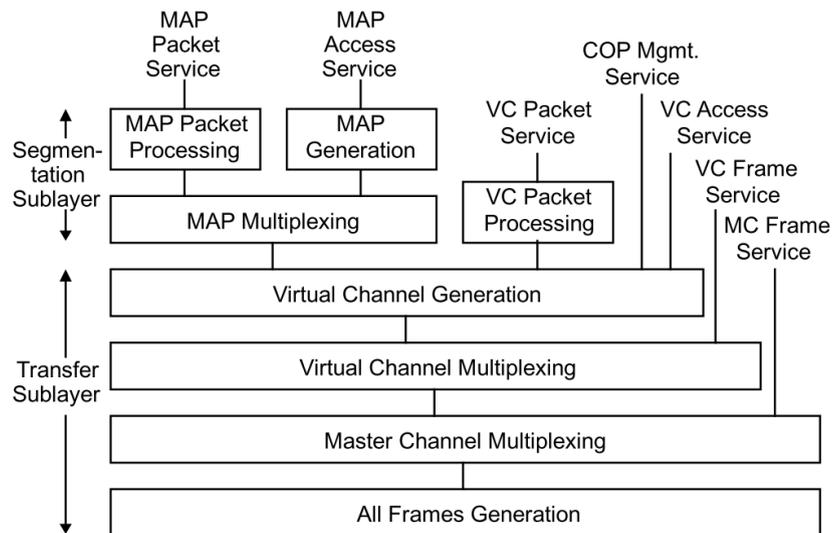


Figure 3.7: TC Sending Node. Adapted from [4], pg.2-12

Applications from a ground station can send data simultaneously in a shared channel through the Virtual Channel (VC) feature. Packets from different applications within the node are versioned with a packet version number which are multiplexed to share the Multiplexer Access Point (MAP) channel. Similarly packets of different versions can share a single virtual channel by being multiplexed to have a new identifier. Virtual channels are inputs to the Virtual Channel Generation which generates packet units called Transfer Frame (TF) and implements an ARQ window protocol for frame retransmissions for unacknowledged frames[40]. The structure of a TF is shown in the Figure 3.8.

Transfer Frame Primary Header: It is mandatory field with 8 subfields:

- TF Version Number(2 bits): Identifies the data unit as a TF frame, always '00'
- Bypass Flag (1 bit): Indicates whether a frame is a Type A (sequence-controlled with ARQ) or Type B(no retransmissions) frame.
- Control Command Flag (1 bit): Indicates whether the data field contains data or control commands.

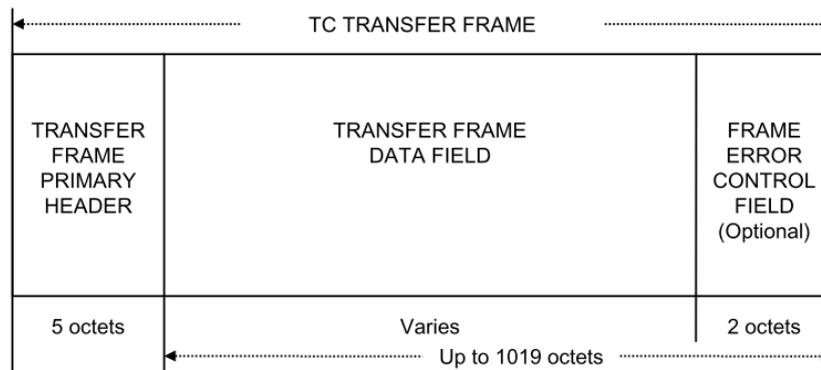


Figure 3.8: TC Transfer Frame. Sourced from Telecommand Space Data Link Protocol Recommendation Report [4], pg.4-1, pg.4-2

- Reserved Space (2 bits): Reserved for future use.
- Spacecraft Identifier (10 bits): The identifier for the spacecraft associated with the data.
- Virtual Channel Identifier (6 bits): Identifies the virtual channel
- Frame Length (10 bits): The number of bytes less than one in the packet.
- Frame Sequence Number (8 bits): The frame sequence number, N(S).

Data Field: The data field contains an integral number of bytes containing information up to 1019 bytes or 1017 bytes if Frame Error Control field is present[4].

Frame Error Control Field: The All Frames Generation module performs the error control procedure of adding an error control information if FEC is used[40]. The 16-bit error detection field is useful for double-checking undetected error from the Channel Synchronisation and Coding layer, and it is optional. The FEC[40] is a CRC-16 with a generator polynomial $x^{16} + x^{12} + x^5 + 1$.

The receiver has a similar service architecture as Figure 3.7, to perform acceptance checks and error correction, demultiplex virtual channels and extra packets. The receiving node's architecture is given in Appendix C.

Communications Operations Procedure (COP-1): The COP-1 Management specifies closed-loop mechanisms to provide guaranteed service delivery. The sending node Frame Operation Procedure (FOP) transmits a TF to the receiving node, which perform acceptance checks through the Frame Acceptance and Reporting Mechanism(FARM) and report back to the sender using Communications Link Control Words (CLCWs)[4].

Synchronisation and Channel Coding Sublayer

The layer below All Frames Generation is the Synchronisation and Channel Coding layer

which adds synchronisation bits and error control information. The internal structure of the Channel and Synchronisation and Channel Coding sublayer is shown in Figure 3.9. The

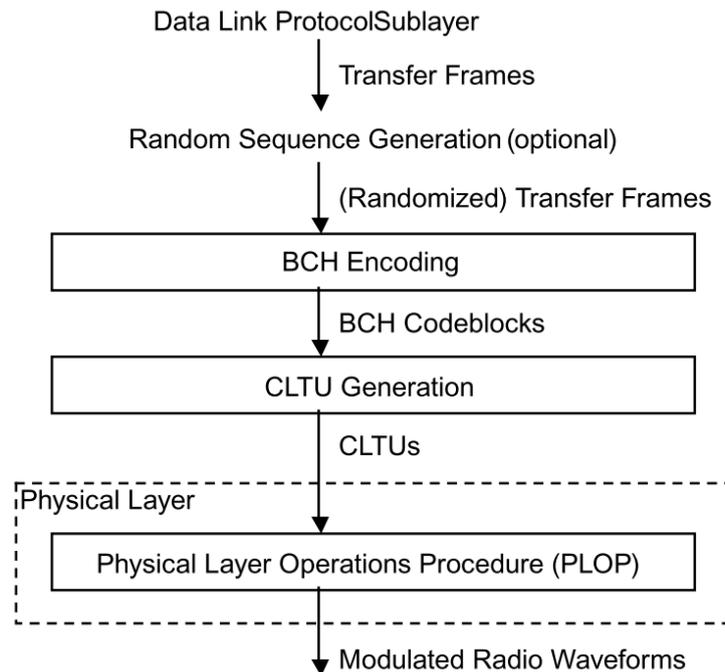


Figure 3.9: TC Synchronisation and Channel Coding Sublayer and Physical Layer. Adapted from [5], pg.2-4

Transfer Frame from the data link layer is optionally pseudo-randomised to improve bit synchronization.

BCH Encoding

The TFs are formatted into fixed length Bose-Chaudhuri-Hocquenghem (BCH) codeblocks using a systematic BCH code of length 64 bits where 56 of the bits are information bits[5], 7 parity bits, and a filler bit. The BCH codeblock format is provided in Figure 3.10.

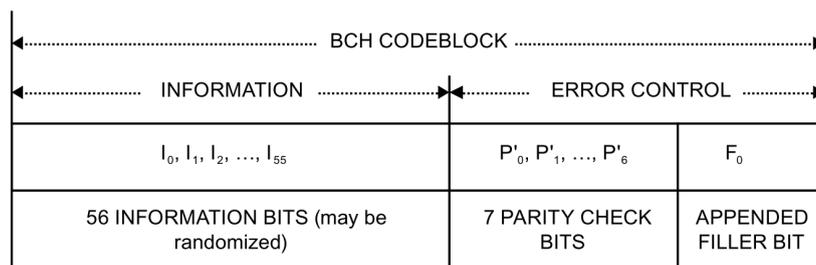


Figure 3.10: BCH Codeblock. Adapted from [5], pg.2-4

If the transfer frame is less in size than 56 bits, the frame is padded with a sequence of alternating '1's and '0's starting with a '0'. The receiver data link layer is then responsible for

stripping out the extra bits. The BCH(63, 56) code uses the generator polynomial,

$$g(x) = x^7 + x^6 + x^2 + 1 \quad (3.3)$$

to generate the 7-bit parity bits which are stored as complements in the codeblock.

The receiver can decode the frames either in an error-detecting mode (capable of detecting up to 3 bits in error) or an error correcting mode (capable of detecting 2 bits in error and one bit in error can be corrected)[5].

Communications Link Transmission Unit (CLTU)

The unit of transmission for the physical layer is the CLTU, which consists of the start sequence, BCH codeblocks, and a tail sequence[5]. The structural components of the are shown in Figure 3.11.

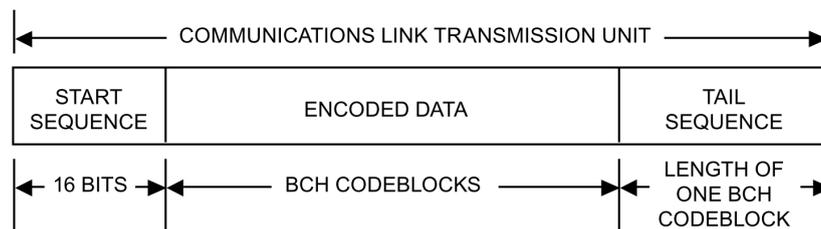


Figure 3.11: CLTU Unit Format. Adapted from [5]

Start Sequence: The 16-bit start sequence marks the start of the contiguous BCH blocks. It is made of the pattern

0000100111010111

which is transmitted as the least significant bit first.

Encoded Data: The encoded data consists of a set of BCH codeblocks either randomised or not depending on the mission design.

Tail Sequence: The tail sequence marks the end of the CLTU which is 64 bits long. The pattern is leading contiguous 7 bytes 11000101 and the eighth octet is 01111001.

110001011100010111000101110001011100010111000101110001011100010111000101111001

3.5.2 Telecommand Implementation

As with the two preceding protocols, the CCSDS TC was implemented in a minimalistic approach. In CCSDS standardisation, the protocol can be operated in an Expedited (Type B) service, which does not use the systematic retransmission mechanism[5].

The Transfer Frame Primary Header is set to

00 1 0 00 xxxxxxxxxxxx 000001 xxxxxxxxxxxx 00000000

with the 3-bit set to '1' as an indicator for a Type-B frame. The spacecraft identifier and frame length are represented with 'x's since they are simulation-dependent variables. The Control Command flag is cleared to '0' since for the purposes of the simulation only data(D) frames will be transferred. The result is a compounded Type-BD frame. The frame sequence number is not used, thus it will be held constant at 0x00. Due to the interest of the project in the underlying low-level data link layer, only one higher-layer application is assumed, such that only one virtual channel is utilised, hence the virtual channel number value of 1.

The FEC field is not used. Instead the BCH(63,56) error handling module is responsible for error checking and correction. The code generation is a pipelined version of a polynomial division of the message bits

$$m(x) = m_{70}x^{70} + m_{69}x^{69} + \dots + m_0x^0 \quad (3.4)$$

by the divisor,

$$g(x) = x^7 + x^6 + x^2 + 1 \quad (3.5)$$

to obtain the remainder, $b(x)$. The transmitted message which is represented in polynomial form $m(x)$, has coefficients for each term being either a 0 or 1. The receiver locks into incoming signal, and thereafter for each block, computes the syndromes $S(x)$ for a received vector, which is a polynomial division by $g(x)$. If $S(x)$ is zero, it is assumed that the block did not incur any error along the channel so it is delivered to the upper layers. If $S(x) \neq 0$, the block is handed to the error correction unit. The error correction implementation is similar to the one mentioned in subsection 3.3.2 which finds the Hamming distance between the received message vector, $r(x)$ and the original message vector, $t(x)$. Technically with the modified BCH(63,56) code, in triple error detection mode, up to 3 bits in error can be detected whereas in single error correction, 2 bits in error can be detected and 1 bit corrected. The latter is an appealing option. So, if the Hamming distance is less than or equal to 1 within a 64-bit block, the data is correctable otherwise the block is dropped.

The TC was not part of the initial implementation of the protocols(the AX.25, FX.25, AFX.25) in SatSim, but it has been added as part of the additions for this work.

3.6 Simulation Scenario

The simulation comparisons were based on the configurations:

- An omnidirectional ground station antenna is used without spacecraft tracking capability. The maximum antenna gain is 40dB.
- The transmit power is 1 W

- The spacecraft is in circular 600km sun synchronous orbit, propagated by the PyEphem
- The modulation scheme used is BPSK
- The baud rate is 9600 bps
- The randomness of the experiment requires that the results are averaged over a number of repeats. Le Roux [1] noted that the results gets to stabilise over 8 to 16 repeats. However, for this project the results were averaged over 4 repeats because the simulation is time consuming, it took around 4.5 hours a simulation for 4 repeats. Furthermore, from the data averaging analysis performed by Le Roux [1] and graphically shown on [1, pp. 53] Figure 3.37, the data obtained over 4 repeats gives enough information having compared the 4 and 8 repeat subplots.
- Packet sizes of 100, 156, 177, 205, 303, 506, 702, 905, 1017 information bytes were simulated. The packets are generated on demand (dynamically when needed).

The metric used for comparison is the throughput, which is the number of data bits received per unit of time,

$$\text{Throughput}(\text{bps}) = \frac{\text{Amounts of data received (bits)}}{\text{Amount of time (s)}} \quad (3.6)$$

FX.25 can use different code rates of the Reed Solomon for error correction from 239/255 to 191/255. The comparisons will use the RS(255, 239) for the FX.25 and the BCH(64,56) for the CCSDS TC. The RS(255,239) is the basic among the FX.25 FEC codes with error correction of 8 bytes. The CCSDS TC uses only one FEC scheme, the BCH(64,56).

3.6.1 FX.25 vs CCSDS TC

The simulation comparisons for FX.25 against CCSDS TC were based on the configurations:

- Two ground stations are used, one for sending FX.25 frames on 437.31 MHz and the other for sending CCSDS TC packets on 437.41 MHz.
- Two satellites, one receiving FX.25 frames and the other receiving CCSDS TC frames.
- The satellite pass was 800 seconds

3.6.2 AFX.25 and CCSDS TC

The simulation comparisons for AFX.25 against CCSDS TC were based on the following configurations:

- AFX.25 : The ground station uses a transmitter at 437.11 MHz and a receiver at 437.21 MHz. The transmitter sends data packets and the receiver listens for switching commands from the satellite. The satellite has a transmitter sending commands at 437.21 MHz and a receiver listening to packets at 437.11 MHz.
- CCSDS TC: The CCSDS ground station has a transmitter sending data packets on 437.41 MHz. The corresponding spacecraft receives data packets at the ground station's frequency.
- The satellite pass was 800 seconds.

3.6.3 Discussion of Results

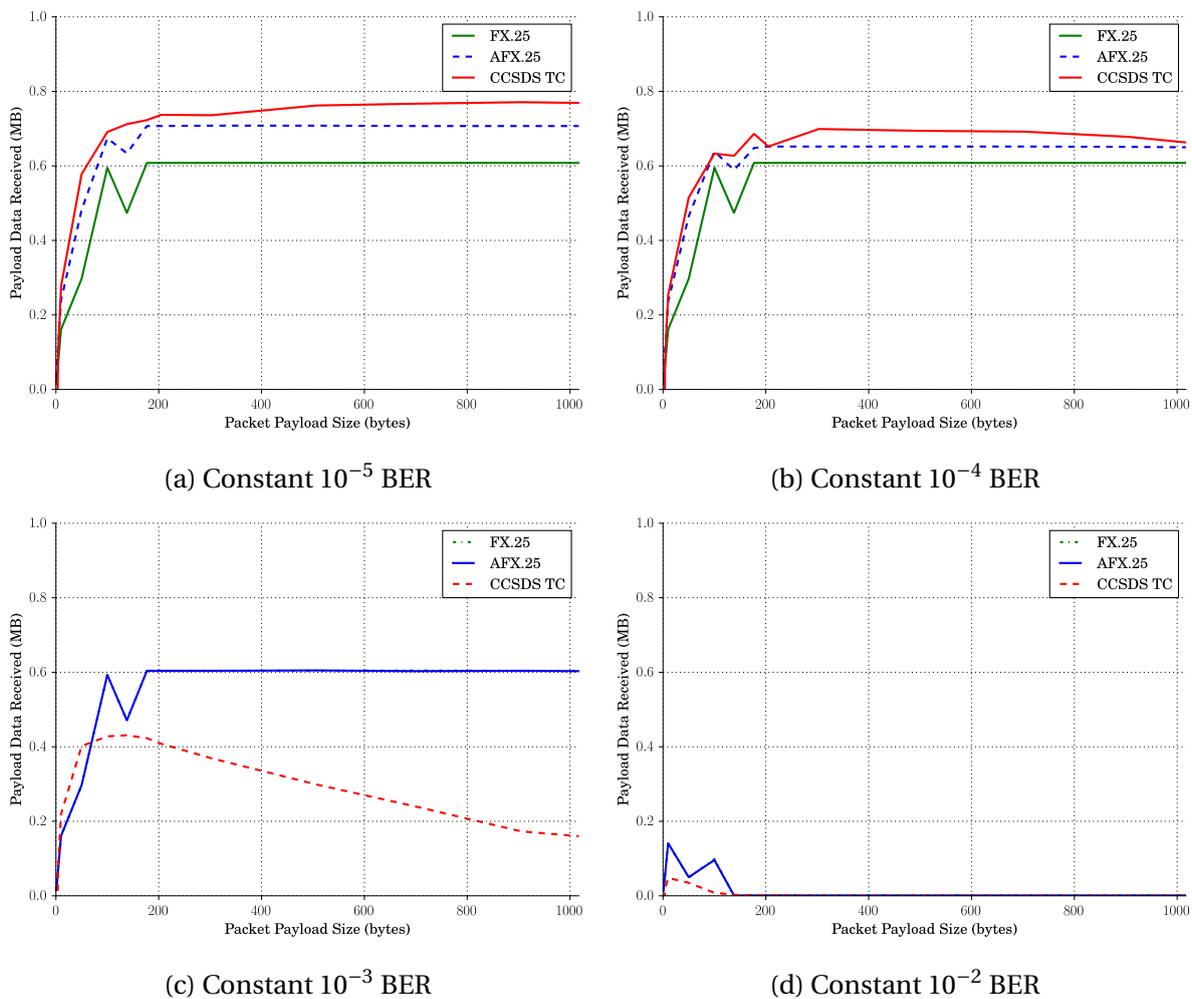


Figure 3.12: Throughput Curves for different BER for different payload sizes

The amount of payload data received per session for each protocol for different payload sizes is shown in Figures 3.12a, 3.12b, 3.12c, 3.12d for different constant BER. Different payload data sizes were used for the tests from 100 bytes to 1017 bytes. The objective is determining the protocol's response to various data sizes and different bit error rates. Amongst the

compared protocols, the CCSDS TC has the maximum allowed payload size of 1017 bytes. For Figures 3.12a and 3.12b, the total usable data is steadily increasing as the packet size increases until a peak point is reached where total payload data is constant. If the payload size is small, the total amount of usable data will be smaller compared with larger packet sizes because smaller payload sizes require more padding bits. As the payload size increases, the amount of information bytes received also increases until all the information bytes are filled with actual data. For the FX.25, the maximum payload goes as high as around 210 bytes, according to [1] to allow for the bits introduced by the AX.25 bit stuffing. The CCSDS TC has better performance over the FX.25 as it can be observed from Figure 3.12a for less noisy channels. The CCSDS TC has a better performance than the FX.25 in the region due to the padding methods which gives smaller padding for the CCSDS TC in the bit range while the FX.25 has a larger padding in the byte range.

However, the BCH(64,56) code does not perform well with low signal-to-noise ratios. The CCSDS TC performance degrades for more noisy channels as observed from Figures 3.12b to 3.12c. The Reed Solomon has strong error correction capabilities which is the reason that the FX.25 performs better than the CCSDS TC for higher bit error rates where bursty errors are prevalent. In these regions, the AFX.25 operates as a pure FX.25 protocol. From Figure 3.12c, it is observed that the CCSDS error correction capability degrades with increasing packet size because the maximum CCSDS TC transfer frame has a maximum of 1017 bytes of which the probability of data being invalidated increases with bigger packets. The packet size and data corruption can be observed also from Figure 3.12d which has a very low signal-to-noise ratio. The first packet sizes 100, 156, 177 show some received packets after which with increased payload size, all the packets get corrupted.

Having observed the payload bytes received for constant BERs, Figure 3.13 shows results logged for an experiment for a dynamic BER for satellite pass from acquisition of signal to loss of signal.

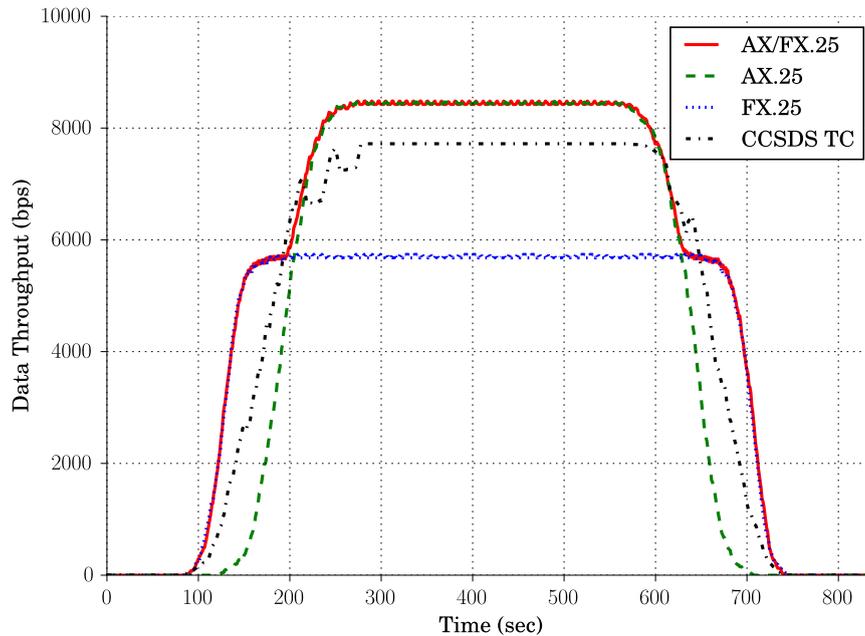


Figure 3.13: Throughput for FX.25, AFX.25, CCSDS Telecommand of data size 156 bytes

The simulation in Figure 3.13 uses an idealistic switching from FX.25 to AX.25 and back which was hard-coded to make the switchovers on the 200 and 620 seconds intervals. The data throughput shows that the AFX.25 throughput is the highest followed by the CCSDS TC. The AFX.25 operates in FX.25 from 0 to 200 seconds, a protocol change to AX.25 occurs at 200 seconds and a change to the FX.25 at 625 seconds.

The total data received for the simulation shown in Figure 3.13 is shown in Table 3.5. The relationship compares the protocols where the AX.25 is used as a base for the comparisons.

Protocol	Total Data Received (KB)	Relative Performance (%)
AX.25	459.7	100.0
FX.25	400.5	87.1
AFX.25	531.3	115.6
CCSDS TC	463.5	100.8

Table 3.5: AX.25/FX.25/AFX.25/CCSDS TC Data Received over 800 seconds

From the experiment, it is clear that AFX.25 is the best candidate based on the throughput. However, the real estate cost of implementing the protocol must be considered before a final choice of candidate protocol for implementation on FPGA is considered.

Chapter 4

Adaptive AFX.25 Hybrid Hardware

The previous chapter evaluated the protocol performance at software simulation. This chapter first investigates the projected hardware implementation complexity to evaluate the costs given the improvement in throughput.

4.1 Protocol Choice

4.1.1 Projected Cost of Hardware Implementation

1. **Encoding** The BCH(63,56) code generator makes use of the generator polynomial $g(x) = x^7 + x^6 + x^2 + 1$ [5]. The hardware design with linear shift registers is shown in Figure 4.1.

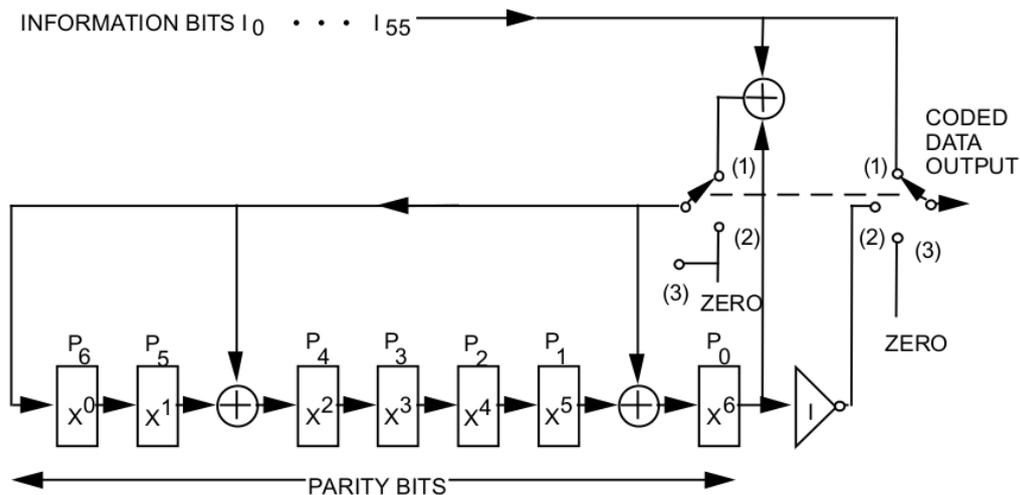


Figure 4.1: BCH(63,56) Code Generator. Adapted from TC Synchronisation and Channel Coding Recommendation [5], pg.3-2

The R-S (255,239) encoder architecture based on the Linear Feedback Shift Register (LFSR) design is shown in Figure 4.14. The encoder uses the coefficients generated

from the generator polynomial [6]:

$$g(x) = x^{16} + 59x^{15} + 13x^{14} + 104x^{13} + 189x^{12} + 68x^{11} + 209x^{10} + 30x^9 + 8x^8 + 163x^7 + 65x^6 + 41x^5 + 229x^4 + 98x^3 + 50x^2 + 36x + 39 \quad (4.1)$$

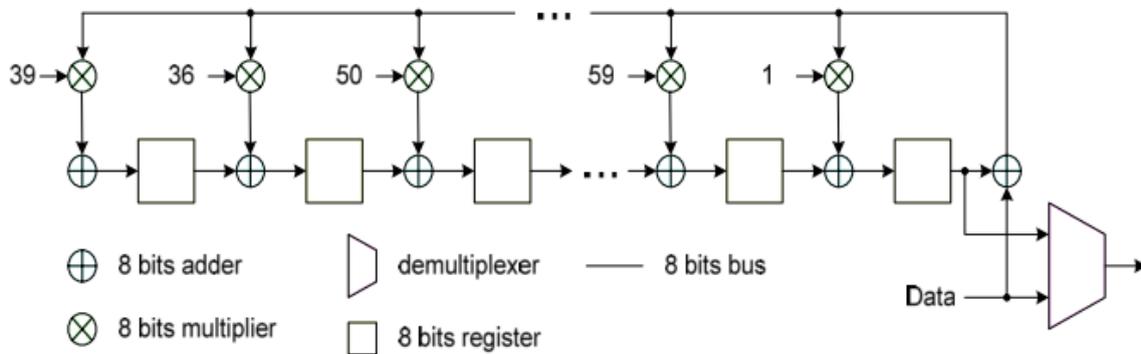


Figure 4.2: Reed Solomon (255,239) Encoder Architecture. Adapted from [6], pg.2

2. Decoding

Decoding received data bits is computationally more expensive than encoding. For the pre-design analysis, a qualitative comparison of the BCH and RS is presented. The encoding and decoding algorithms will be discussed in full in the sections to follow. The algebraic decoding of BCH or RS codes has the following general steps[38]:

- Computing the syndrome
- Determining the error locator polynomial to find the error locations. The Peterson's, Berlekamp-Massey and Euclidean algorithms are used.
- Solving the error locator polynomial using the Chien search
- Finally, the Forney algorithm is used to determine the error values in the case of RS and non-binary BCH.

The Reed Solomon is essentially a non-binary BCH code which makes its hardware circuit to have more gates than the binary version. For a binary BCH the polynomial coefficient is either a 0 or 1, rendering the multipliers in Figure 4.2 either a wire or an open connection as can be viewed in Figure 4.1.

Secondly, the RS(255, 239) is capable of correcting more bytes (8 bytes) in error than the BCH(63,56), which comes with a proportional cost of hardware implementation.

4.1.2 Protocol Choice of Implementation

From the simulations performed in subsections 3.6.1, 3.6.2, 4.1.1, it can be deduced that the hybrid AFX.25 has an improved throughput over the ordinary AX.25, FX.25 or CCSDS TC. Even though the AFX.25 protocol shows a theoretical improvement, the complexity of the hardware implementation indicates that it requires more hardware logic units than the BCH (64,56).

A summary of comparisons of the protocols studied in the previous chapter and subsections and from the work of Le Roux [1] is given in Table 4.1.

Table 4.1: Protocols Comparison Summary

Protocol	Simulation Performance	Hardware Cost	Popularity
AX.25	Good on error free channels[1]	Simple Implementation	52.17% among CubeSats [9]
FX.25	Good on noisy channels[1]	High in Hardware Complexity (due to non-binary encoding and decoding)	0.87% usage on CubeSats [9]
AFX.25	Good performance both on noisy and noiseless channels	Highest in Hardware Complexity (FX.25 and AX.25 decoding plus switching)	New protocol, not yet available amongst CubeSat developers
CCSDS TC	Good performance for smaller packet sizes	Lighter in hardware complexity than FX.25	3.48% among CubeSat launches [9]

After the simulation and hardware analysis, it is a good option to choose the AFX.25 as the protocol of implementation for this work for the following reasons:

- It has a performance significantly higher than all the other investigated protocols as observed from Figure 3.13
- It has a backward compatibility with the existing AX.25 systems which are more widely used in the CubeSat community

Having chosen the AFX.25, the fact that it comes with high costs of hardware implementation cannot be ignored. There was a need to implement the protocol on hardware to get a better understanding and estimate of the true implementation cost.

It is worth mentioning that the CCSDS Telecommand is a relatively old protocol which was introduced specifically for its simplicity in traditional satellites. Even though the CCSDS

standards are reviewed every five years, the error-coding protocol has not changed for many years. Kazz *et al.* [41] made their submission in 2012, to replace the CCSDS TC with the Next Generation Uplink (NGU) which adopts six variants of error correction schemes amongst other improvements. These error codes include the BCH(63,56), Low Density Parity Codes (LDPC) $_{\frac{1}{2}}$ (128,64), LDPC $_{\frac{1}{2}}$ (256,128), LDPC $_{\frac{1}{2}}$ (512,256), Non-binary LDPC $_{\frac{1}{2}}$ (512,256), and the LDPC $_{\frac{1}{2}}$ (2048,1024). These error corrections will undoubtedly improve the overall throughput of the TC under certain conditions[41]. However, the suggestions from their work have not been implemented in the simulation of the TC since it is not yet included in the official recommended standards which were updated in 2015.

4.2 FEC Theoretical Background for FX.25

FX.25 employs the Reed Solomon FEC scheme. In the following section, the theory of operation of BCH which will be required to implement the FX.25 FEC code in hardware is discussed.

4.2.1 Introduction

Reed Solomon codes are heavily dependent on abstract algebra mathematics in the Galois Field, named after the French mathematician Évariste Galois.

4.2.2 Galois Field

A Galois Field (GF) consists of a finite set of elements whereby the addition, subtraction, multiplication and division operations obey certain rules. The elements are based on a primitive element[7], α which is a root of the field generator polynomial, $\phi(X)$. The elements take the form:

$$0, \alpha^0, \alpha^1, \alpha^2, \dots, \alpha^{(N-1)} \quad (4.2)$$

where $N = 2^m - 1$. Such a field will be a $GF(2^m)$. The Reed Solomon code (255,239) is in the $GF(256)$ field where $m = 8$.

To construct such a field, the primitive element is used as root of the field generator polynomial. For $\phi(X) = X^8 + X^4 + X^3 + X^2 + 1$ [42],

$$\alpha^8 + \alpha^4 + \alpha^3 + \alpha^2 + 1 = 0 \quad (4.3)$$

therefore

$$\alpha^8 = \alpha^4 + \alpha^3 + \alpha^2 + 1 \quad (4.4)$$

The two equations are equivalent, as addition and subtraction yield similar results within a field, as it will be shown in the following subsection.

The first element is 0, followed by α^0 and multiplied by α at each stage and substituting α^8 for $\alpha^4 + \alpha^3 + \alpha^2 + 1$. Table 4.3 shows the first few elements of the GF(256). The complete set can be found in Appendix D.

Table 4.3: GF(256) elements generated from the generator polynomial, $\phi(X) = X^8 + X^4 + X^3 + X^2 + 1$

Index Form	Polynomial Form	Binary Form	Decimal Form
0	0	00000000	0
α^0	1	00000001	1
α^1	α^1	00000010	2
α^2	α^2	00000100	4
α^3	α^3	00001000	8
α^4	α^4	00010000	16
α^5	α^5	00100000	32
α^6	α^6	01000000	64
α^7	α^7	10000000	128
α^8	$\alpha^4 + \alpha^3 + \alpha^2 + 1$	00011101	29
α^9	$\alpha^5 + \alpha^4 + \alpha^3 + \alpha^1$	00111010	58
α^{10}	$\alpha^6 + \alpha^5 + \alpha^4 + \alpha^2$	01110100	116
α^{11}	$\alpha^7 + \alpha^6 + \alpha^5 + \alpha^3$	11101000	232
\vdots	\vdots	\vdots	\vdots
α^{253}	$\alpha^6 + \alpha^2 + \alpha^1 + 1$	01000111	71
α^{254}	$\alpha^7 + \alpha^3 + \alpha^2 + \alpha^1$	10001110	142
α^{255}	1	00000001	1

The GF basically has two operations addition and multiplication, subtraction and division are additive and multiplicative inverses respectively.

4.2.2.1 Addition

Given two polynomials,

$$(a_{m-1}X^{m-1} + \dots + a_1X^1 + a_0X^0) + (b_{m-1}X^{m-1} + \dots + b_1X^1 + b_0X^0) = (c_{m-1}X^{m-1} + \dots + c_1X^1 + c_0X^0)$$

the sum c is a modulo-two addition of the coefficients, which in binary form takes an exclusive-OR operation[42] of bits in corresponding bit positions. In a nutshell,

$$(c_{m-1}X^{m-1} + \dots + c_1X^1 + c_0X^0) = ((a_{m-1} + b_{m-1})X^{m-1} + \dots + (a_1 + b_1)X^1 + (a_0 + b_0)X^0)$$

A basic GF adder circuit can be viewed in Figure 4.3.

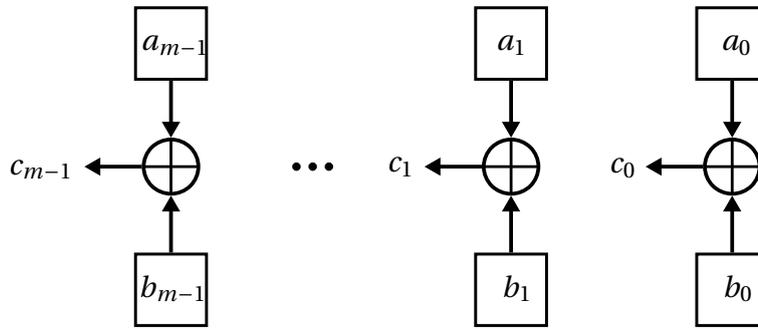


Figure 4.3: Galois Field Adder

The additive inverse of an element a is $-a$ such that $a + -a = 0$, thus in the GF, addition is similar to subtraction.

4.2.2.2 Multiplication

Galois Field multiplication is similar to general polynomial multiplication with the exception that product, $p(X)$ must have a degree less than the generator polynomial $\phi(X)$'s degree. If it happens that the degree is higher, then the final $p(X) = p(X) \bmod \phi(X)$. Alternatively, the multiplication can be achieved through lookup tables of the nature of table 4.3. Traditionally, multiplying 2 and 4,

$$2_{10} = 010_b = X_{(\text{polynomial form})}$$

$$4_{10} = 100_b = X^2_{(\text{polynomial form})}$$

$$X \cdot X^2 = X^3 = 1000_b = 8_{10}$$

Alternatively, using lookup table 4.3, 2 is α^1 and 4 is α^2 . Adding the powers of the 2 elements, we have α^3 whose decimal form is 8.

An example of a circuit that multiplies an element β of $GF(2^8)$ by the element α^3 which is the element, 8_{10} .

$$\beta = b_0 + b_1\alpha + b_2\alpha^2 + b_3\alpha^3 + b_4\alpha^4 + b_5\alpha^5 + b_6\alpha^6 + b_7\alpha^7$$

So

$$\alpha^3\beta = b_0\alpha^3 + b_1\alpha^4 + b_2\alpha^5 + b_3\alpha^6 + b_4\alpha^7 + b_5\alpha^8 + b_6\alpha^9 + b_7\alpha^{10}$$

And the primitive polynomial is $\phi(X) = X^8 + X^4 + X^3 + X^2 + 1$, therefore

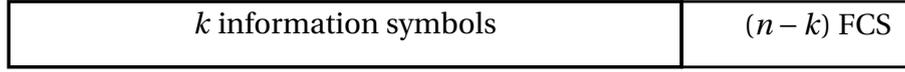
$$\begin{aligned} \alpha^3\beta &= b_0\alpha^3 + b_1\alpha^4 + b_2\alpha^5 + b_3\alpha^6 + b_4\alpha^7 + b_5(\alpha^4 + \alpha^3 + \alpha^2 + 1) \\ &\quad + b_6(\alpha^5 + \alpha^4 + \alpha^3 + \alpha) + b_7(\alpha^6 + \alpha^5 + \alpha^4 + \alpha^2) \end{aligned} \quad (4.5)$$

And grouping similar terms, the result is

$$\begin{aligned} \alpha^3\beta &= b_5 + b_6\alpha + (b_5 + b_7)\alpha^2 + (b_0 + b_5 + b_6)\alpha^3 + (b_1 + b_5 + b_6 + b_7)\alpha^4 \\ &\quad + (b_2 + b_6 + b_7)\alpha^5 + (b_3 + b_7)\alpha^6 + b_4\alpha^7 \end{aligned} \quad (4.6)$$

4.2.3.1 Encoding

A Reed Solomon codeblock represented as an (n, k) block is a formulated n -sized data format that is recognised both by the encoder and decoder. The format of the block is shown in Figure 4.5.



FCS is the Forward Error Correction Symbols

Figure 4.5: Reed Solomon Codeblock

The codeblock is generated using a code generator polynomial, $g(X)$ which has $2t$ factors[42],

$$g(X) = \prod_{i=0}^{2t-1} (X + \alpha^{b+i}) = (X + \alpha^b)(X + \alpha^{(b+1)}) \dots (X + \alpha^{(b+2t-1)}) \quad (4.7)$$

where $b \geq 0$, but typically 0 or 1 depending on the code design.

Suppose the hexadecimal message information $7E, 10, \dots, BC$ is to be transmitted from satellite to ground node. The message can be represented as a polynomial of the form

$$M(X) = M_{k-1}X^{k-1} + \dots + M_1X + M_0$$

whose coefficients M_{k-1}, \dots, M_1, M_0 corresponds to $7E, 10, \dots, BC$. The encoder creates the codeblock which necessitates data recovery at the receiving node in the case of data corruption in the channel. The encoding process is a typical 3 step process outlined[7]:

1. Shift the message polynomial $M(X)$ by $n-k$ bytes to the left. Mathematically, $M(X) \cdot X^{n-k}$.
2. Then divide $M(X) \cdot X^{n-k}$ by the generator polynomial $g(X)$ to get the remainder, $b(X)$.
3. Lastly the codeword is a concatenation of $M(X) \cdot X^{(n-k)}$ and $b(X)$

The transmitted codeword, $C(X)$ then becomes

$$C(X) = \sum_{i=0}^{n-1} C_i X^i = M_{k-1}(X)X^{n-1} + \dots + M_0X^{n-k} + b_{n-k-1}(X)X^{n-k-1} + \dots + b_0 \quad (4.8)$$

An implementation of the encoder can be achieved through polynomial division.

To simplify the hardware circuit, the value of $b = 0$ is chosen

$$g(X) = 1 + g_1X + g_2X^2 + \dots + g_{n-k-1}X^{n-k-1} + g_{n-k}X^{n-k}$$

The polynomial has the roots, $1, \alpha, \alpha^2, \alpha^3, \dots, \alpha^{2t-1}$ and the coefficients from within the field, $GF(2^m)$. A representative hardware encoder circuit is given in Figure 4.6.

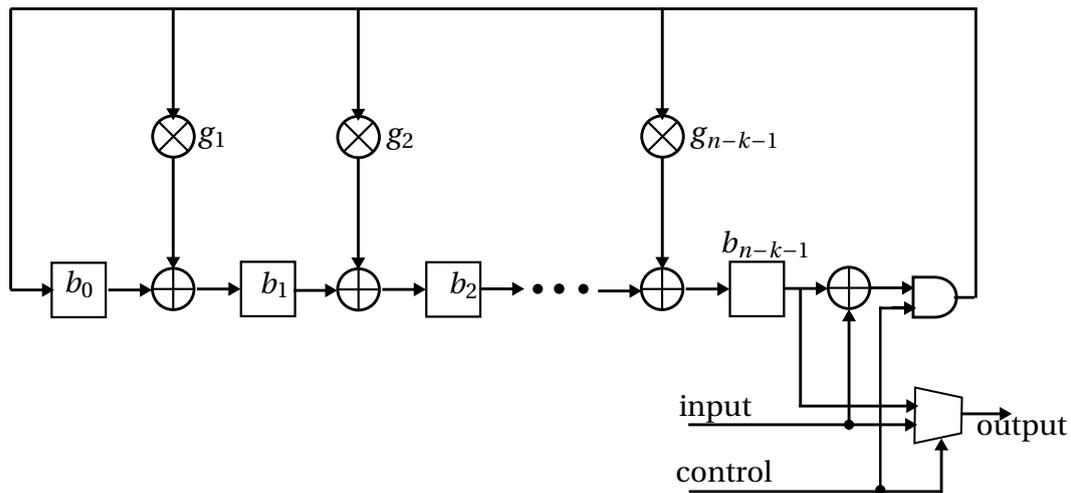


Figure 4.6: RS(n,k) Encoder for code generator polynomial $g(X) = 1 + g_1X + g_2X^2 + \dots + g_{n-k-1}X^{n-k-1} + g_{n-k}X^{n-k}$. Adapted from [7]

The message $M(X)$ is serially fed into the encoder, and simultaneously to the channel via the output multiplexer. After the last byte of the message is received in the circuit, the contents of the registers $b_{n-k-1}, b_{n-k-2}, \dots, b_2, b_1, b_0$ contains the remainder values. Each of the registers b_i can store a field element from the field $GF(2^m)$. Thus, the output buffer then disables the raw message input but selects the remainders as the multiplexer input.

4.2.3.2 Decoding

During transmission, it is possible that the codeword may incur channel noise resulting in the received message, $R(X) = R_0 + R_1X + R_2X^2 + \dots + R_{n-1}X^{n-1}$ to be a combination of the original codeword and noise. $R(X)$ can be represented as

$$R(X) = C(X) + E(X) \quad (4.9)$$

where $E(X)$ is the error pattern

$$E(X) = \sum_{i=0}^{n-1} E_i X^i = E_0 + E_1X + E_2X^2 + \dots + E_{n-1}X^{n-1}$$

S.1 Syndrome Computation

The first step in the decoding begins with calculating the syndromes, S_1, S_2, \dots, S_{2t} which gives an indicator as to whether the original message was invalidated or not. The syndromes are also useful in the error location and magnitude algorithms (to be discussed shortly in **S.2** and **S.4**).

The syndromes can be computed by substituting the field elements $\alpha, \alpha^2, \dots, \alpha^{2t}$ into the receiver polynomial.

$$\begin{aligned} S_i &= R(\alpha^i) \\ &= R_0 + R_1(\alpha^i) + R_2(\alpha^i)^2 + \dots + R_{n-1}(\alpha^i)^{n-1} \\ &= S_0 + S_1X + S_2X^2 + \dots + S_{2t-1}X^{2t-1} \end{aligned} \quad (4.10)$$

From encoding, it is known that $X + \alpha^i$ is a factor of $T(X)$ implying that $T(\alpha^i) = 0$, therefore from 4.9,

$$R(\alpha^i) = E(\alpha^i) = S_i \quad (4.11)$$

If $S_i(X) = 0$, the received message does not contain any errors and no further processing is required, the packet is delivered to upper layers. If $S_i(X) \neq 0$, the message contains errors.

The error pattern contains ν errors at locations j_1, j_2, \dots, j_ν where $\nu \leq t$.

The error pattern equation can be represented as

$$E(X) = e_1X^{j_1} + e_2X^{j_2} + \dots + e_\nu X^{j_\nu} \quad (4.12)$$

where e_1, e_2, \dots, e_ν are the error values at the locations. Both the error locations, j_ν s and the error values, e_ν s are unknowns at this particular time as well as the number of errors, ν . Finding the solutions to equation 4.12 means a step closer to correcting the errors, but the equation has many possible solutions which yield different error patterns. The solution that has the lowest order is the most probable solution. To arrive to such a solution, the Berlekamp iterative algorithm which is considered to be amongst the most efficient so far [7] [42] will be used to find the solution in **S.2**.

Combining equations 4.11 and 4.12,

$$\begin{aligned} S_i &= E(\alpha^i) \\ &= e_1\alpha^{j_1} + e_2\alpha^{j_2} + \dots + e_\nu\alpha^{j_\nu} \\ &= e_1\beta_1^i + e_2\beta_2^i + \dots + e_\nu\beta_\nu^i \end{aligned} \quad (4.13)$$

where

$\beta_1^i, \beta_2^i, \dots, \beta_\nu^i$ are error locators

A pipelined division circuit for finding S_i would be to have $R(X)$ divided by $X + \alpha^i$ to obtain S_i as shown in Figure 4.7. The circuit shows a single syndrome S_i which is representative of the entire circuit with different constant multipliers, $\alpha, \alpha^2, \dots, \alpha^{2t}$. The respective syndrome values will be read from the output of the buffer after a clock pulse.

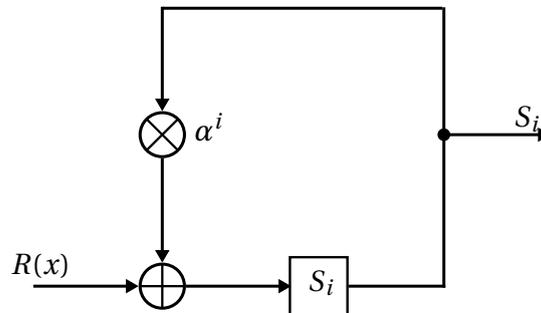


Figure 4.7: Reed Solomon Syndrome Computation Circuit for a single S_i . Adapted from [7]

The syndromes can be represented in a matrix form as:

$$\begin{pmatrix} S_0 \\ S_1 \\ \vdots \\ S_{2t-1} \end{pmatrix} = \begin{pmatrix} \beta_1^0 & \beta_2^0 & \cdots & \beta_v^0 \\ \beta_1^1 & \beta_2^1 & \cdots & \beta_v^1 \\ \vdots & \vdots & \ddots & \vdots \\ \beta_1^{2t-1} & \beta_2^{2t-1} & \cdots & \beta_v^{2t-1} \end{pmatrix} \begin{pmatrix} e_1 \\ e_2 \\ \vdots \\ e_v \end{pmatrix} \quad (4.14)$$

Solving the system of non-linear equations of 4.14 directly is a challenging mathematical task, so an intermediate polynomial is introduced.

S.2 Error locator polynomial

Suppose that

$$\sigma(X) = 1 + \sigma_1 X + \cdots + \sigma_v X^v \quad (4.15)$$

is the intermediate polynomial, the error polynomial $\sigma(X)$ which has zeros at the locations β_1, \dots, β_v for $l = 1, \dots, v$ [23]. The equation expressed as factored solutions is

$$\sigma(X) = (1 - \beta_1 X)(1 - \beta_2 X) \cdots (1 - \beta_v X) \quad (4.16)$$

Multiplying equations 4.15 and 4.16 above by $e_l \beta_l^{j+v}$ and setting $X = \beta^{-1}$ gives

$$\begin{aligned} 0 &= e_1 \beta_1^{j+v} (1 + \sigma_1 \beta^{-1} + \sigma_2 \beta^{-2} + \cdots + \sigma_v \beta^{-v}) \\ &= e_1 (\beta_1^{j+v} + \sigma_1 \beta_1^{j+v-1} + \sigma_2 \beta_1^{j+v-2} + \cdots + \sigma_v \beta_1^j) \\ &= \sum_{l=1}^v e_l (\beta_l^{j+v} + \sigma_1 \beta_l^{j+v-1} + \sigma_2 \beta_l^{j+v-2} + \cdots + \sigma_v \beta_l^j) \end{aligned} \quad (4.17)$$

This equation is the same as

$$\begin{aligned} S_{j+v} + \sigma_1 S_{j+v-1} + \sigma_2 S_{j+v-2} + \cdots + \sigma_v S_j &= 0 \\ \sigma_1 S_{j+v-1} + \sigma_2 S_{j+v-2} + \cdots + \sigma_v S_j &= -S_{j+v} \end{aligned} \quad (4.18)$$

For correctable errors, the number j is $1 \leq j \leq 2t - v$. The systems of equations are simplified to a linear set of equations 4.19 which can be solved directly.

$$\begin{pmatrix} -S_{v+1} \\ -S_{v+2} \\ \vdots \\ -S_{2v} \end{pmatrix} = \begin{pmatrix} S_1^1 & S_2 & \cdots & S_v \\ S_2 & S_3 & \cdots & S_{v+1} \\ \vdots & \vdots & \ddots & \vdots \\ S_v & S_{v+1} & \cdots & S_{2v-1} \end{pmatrix} \begin{pmatrix} \sigma_1 \\ \sigma_2 \\ \vdots \\ \sigma_v \end{pmatrix} \quad (4.19)$$

The Peterson algorithm [23] computes the solution by inverting the matrix for the different values of v . However, such a computation is heavy especially when the number of errors is high since the number of multiplications required will be proportional to

ν^3 . Berlekamp proposed a faster algorithm for finding the coefficients σ s for the smallest $\nu \leq t$ in an iterative manner[23].

Berlekamp algorithm

Let

$$\sigma^{(\mu)}(X) = 1 + \sigma_1^{(\mu)}X + \sigma_2^{(\mu)}X^2 + \dots + \sigma_{l_\mu}^{(\mu)}X^{l_\mu} \quad (4.20)$$

be the polynomial determined at the μ th step of iteration. The algorithm iterative steps fill in the values for Table 4.5 where each column value is computed at each μ th stage from 1 to $2t$.

μ	$\sigma^{(\mu)}(X)$	d_μ	l_μ	$\mu - l_\mu$
-1	1	1	0	-1
0	1	S_1	0	0
1				
2				
\vdots				
$2t$				

Table 4.5: Berlekamp Iterative Stages. Adapted from [7]

The equation 4.21 calculates the *discrepancy*, d_μ at the μ th stage

$$d_\mu = S_{\mu+1} + \sigma_1^{(\mu)}S_\mu + \sigma_2^{(\mu)}S_{\mu-1} + \dots + \sigma_{l_\mu}^{(\mu)}S_{\mu+1-l_\mu} \quad (4.21)$$

If $d_\mu = 0$ at stage μ , the next stage error locator polynomial will be equivalent to the current stage error polynomial.

$$\sigma^{(\mu+1)}(X) = \sigma^{(\mu)}(X) \quad (4.22)$$

and

$$l_{\mu+1} = l_\mu$$

If there is a discrepancy, $d \neq 0$, the next stage error location polynomial will be a corrected version of the current stage error location polynomial. By correction in this context, it does not mean *error correction* but a way of adjusting the stage σ so that it finally converges to a polynomial of minimum degree at stage $2t$ [7]. To find the correction terms, the previous stages are looked up to find a discrepancy at some stage, ρ whose value $d_\rho \neq 0$ and $\rho - l_\rho$ has the largest value, where l_ρ is the degree of $\sigma^{(\rho)}(X)$. The next stage $\mu^{(\mu+1)}(X)$ is found to be

$$\sigma^{(\mu+1)}(X) = \sigma^{(\mu)}(X) + d_\mu d_\rho^{-1} X^{(\mu-\rho)} \sigma^{(\rho)} \quad (4.23)$$

and

$$l_{\mu+1} = \max(l_\mu, l_\rho + \mu - \rho) \quad (4.24)$$

The last stage $2t$ error locator polynomial $\sigma^{(2t)}(X)$ is the required error locator polynomial.

S.3 Finding error locations

Having found $\sigma(X)$, the next step is finding the solutions of $\sigma(X)$ from the received message polynomial, $r(X)$.

Peterson algorithm

Peterson's algorithm for finding solutions to $\sigma(s)$ is substituting the values $1, \alpha, \alpha^2, \dots, \alpha^{n-1}$ into $\sigma(X)$, in each of the substitution and solve[7]. If the answer is 0, that is an indication that the element is a root whereby its reciprocal gives the error location.

Chien search

Chien formalised the algorithm of substituting and finding the roots to be thus. The decoder tests $r(X)$ byte by byte starting from the higher order bytes. Given the received message as

$$r(X) = r_0 + r_1X + r_2X^2 + \dots + r_{n-1}X^{n-1}$$

the $r_{n-1}X^{n-1}$ will be tested first. To decode r_{n-l} , the decoder tests the sum

$$1 + \sigma_1\alpha^l + \sigma_2\alpha^{2l} + \dots + \sigma_t\alpha^{tl}$$

to see if it equals 0. If so, then α^l is a root of $\sigma(X)$, therefore the byte in position $n-l$ is in error[7]. The successive test of each non-zero element can be illustrated in the form,

$$\begin{aligned} \sigma(\alpha) &= 1 + \sigma_1(\alpha) + \sigma_2(\alpha)^2 + \dots + \sigma_t(\alpha)^t \\ \sigma(\alpha^2) &= 1 + \sigma_1(\alpha^2) + \sigma_2(\alpha^2)^2 + \dots + \sigma_t(\alpha^2)^t \\ &\vdots \\ \sigma(\alpha^{n-1}) &= 1 + \sigma_1(\alpha^{n-1}) + \sigma_2(\alpha^{n-1})^2 + \dots + \sigma_t(\alpha^{n-1})^t \end{aligned} \tag{4.25}$$

A typical search unit is shown in Figure 4.8. The coefficients of $\sigma(X)$ are stored in the registers σ_1 to σ_t and pushed into the multiplier \otimes to be multiplied by α^{il} . The products of the multiplications are summed \oplus to determine the output[7].

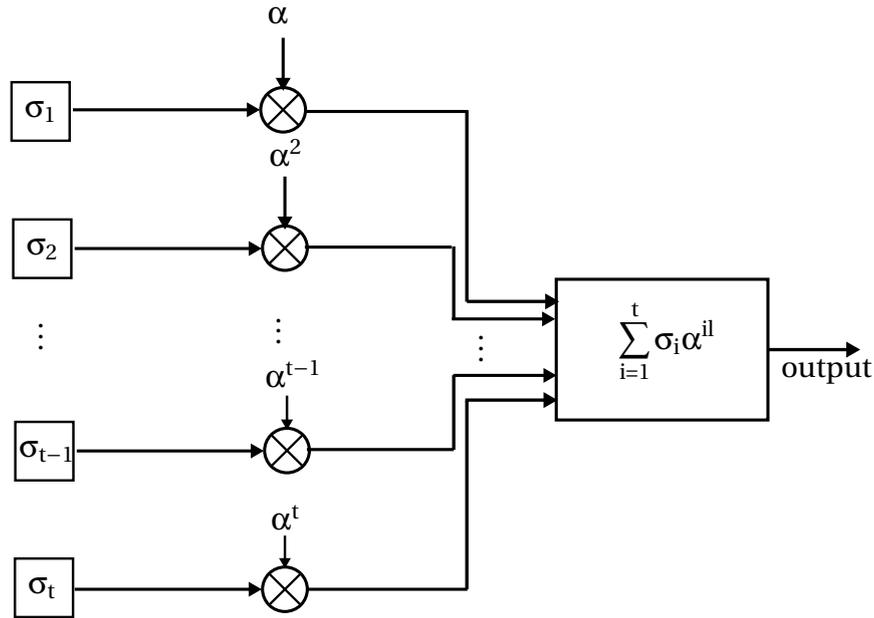


Figure 4.8: Error locator polynomial roots search unit. Adapted from [7]

At this step of decoding, the values for $\beta_1^i, \beta_2^i, \dots, \beta_v^i$ from equation 4.12 are known. The next step is finding the error magnitudes e_1, e_2, \dots, e_v using Forney's algorithm.

S.4 Finding error magnitudes

Forney's algorithm

Let

$$\begin{aligned}
 \Gamma(X) &= [1 + S(X)] \sigma(X) \\
 &= (S_0 + S_1X + S_2X^2 + \dots + S_{2t-1}X^{2t-1})(1 + \sigma_1X + \sigma_2X^2 + \dots + \sigma_vX^v) \\
 &= 1 + (S_1 + \sigma_1)X + (S_2 + \sigma_1S_1 + \sigma_2)X^2 + \\
 &\quad \dots + (S_v + \sigma_1S_{v-1} + \sigma_2S_{v-2} + \dots + \sigma_v)X^v
 \end{aligned} \tag{4.26}$$

be the error magnitude polynomial, then the error value at the found location β_l can be found by substituting the found values to the following equation[7]

$$e_1 = \frac{\Gamma(\beta_1^{-1})}{\prod_{i=0, i \neq 1}^{v-1} (1 + \beta_i \beta_1^{-1})} \tag{4.27}$$

The result of this step gives the values for e_1, e_2, \dots, e_v . Now all the unknowns have been solved, the data can be corrected.

The Gorenstein-Zierler[7] can also be used to determine the error magnitudes but it is not discussed in this work because it is computationally inefficient as it uses the direct matrix determinant method to determine the error magnitudes.

S.5 Error Correction

The data is corrected by XORing the error pattern

$$E(X) = e_1X^{j_1} + e_2X^{j_2} + \dots + e_vX^{j_v}$$

and the received information polynomial

$$R(X) = R_0 + R_1X + R_2X^2 + \dots + R_{n-1}X^{n-1}$$

such that the corrected data $R'(X)$ at l is

$$R'_l(X) = e_l \oplus R_l$$

4.3 FPGA-based Hardware

The discussion in the previous section clearly indicates that error correction requires significant hardware resources and computational time. Field Programmable Gate Arrays (FPGAs) were considered the suitable implementation of choice. An FPGA contains a matrix of gate array logic circuitry that can be configured using a Hardware Description Language (HDL) to give a certain output. Two HDL languages are popular with designing FPGAs, which are Verilog and Very High-Speed Integrated Circuit Hardware Description Language (VHDL). FPGAs were preferred over microcontrollers for the following reasons:

- *Processing Efficiency:* Complex signal processing system designs require a lot of computations to be achieved in a very short interval of time. On multi-core computers, this is normally achieved through distributing the processes among the processor cores. Most microcontrollers do not have such a capability. However, since FPGAs have a configurable structure of gates, processing can be done in parallel on different sections of the array.
- Secondly, FPGA design offer engineers the advantage of predetermining the structure of the circuit. Depending on the requirement, certain blocks can be configured to compute results within a few clock cycles.
- *Flexibility:* FPGAs give the freedom to design any kind of circuit no matter its complexity by configuring the gate interconnections of the array.
- *Microcontroller Embedded:* If the project specifically requires a microcontroller, FPGAs allows for a 'soft-core' processor design and more advanced FPGAs are ported with a hard-core processor on them.
- *Project Requirement:* The project is a proof of concept from the work done by Le Roux [1], at hardware level. Given that the objective was having the system implemented on hardware, FPGAs were the way to go instead of a microcontroller which is still in many ways software.

Having discussed the pros of FPGAs, there are penalties to pay too. The first of these is that FPGAs are relatively expensive compared to microcontrollers. Secondly, for inexperienced developers, FPGA designing and debugging can present a steep learning curve.

The choice was to use the Microsemi M1AFS1500-FGG484 FPGA on a Fusion Embedded Development Kit because the board provides all the required functionalities in one package. Amongst other features, it is pre-packaged with a USB to UART converter circuit which is necessary for PC-to-FPGA communication. The M1AFS1500 development board provides easy integration with other boards aided by the interface connector pins. This was necessary for this project to allow for connecting the RF board. Moreover, the Fusion boards were already available for use in the ESL.

The FPGA design flow is a back-and-forward flow with stages; design, simulation, synthesis, place and route, and flashing as shown in Figure 4.9. It is not a straight-down flow because each of the first three stages' simulation results may require adjustments to the design parameters of the first stage. The names asterisked (*) are the tools used in each stage on the Microsemi design tool chain.

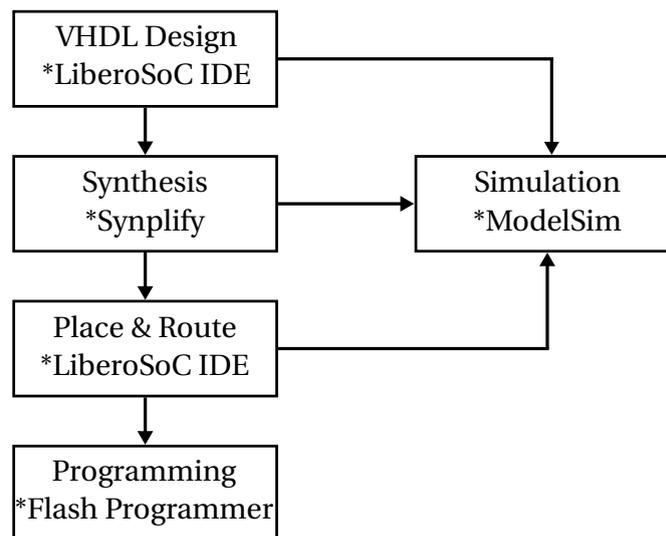


Figure 4.9: FPGA Design Flow

4.4 Hardware Design and Implementation

This subsection covers the design and implementation of the protocol on hardware. The system design of the protocol consists of transmitter and receiver node as illustrated in Figure 4.10. The design presented here does not cover all the OSI layers but it instead outlines the layers pertinent to this work which are the Application, Data Link and the Physical Layer.

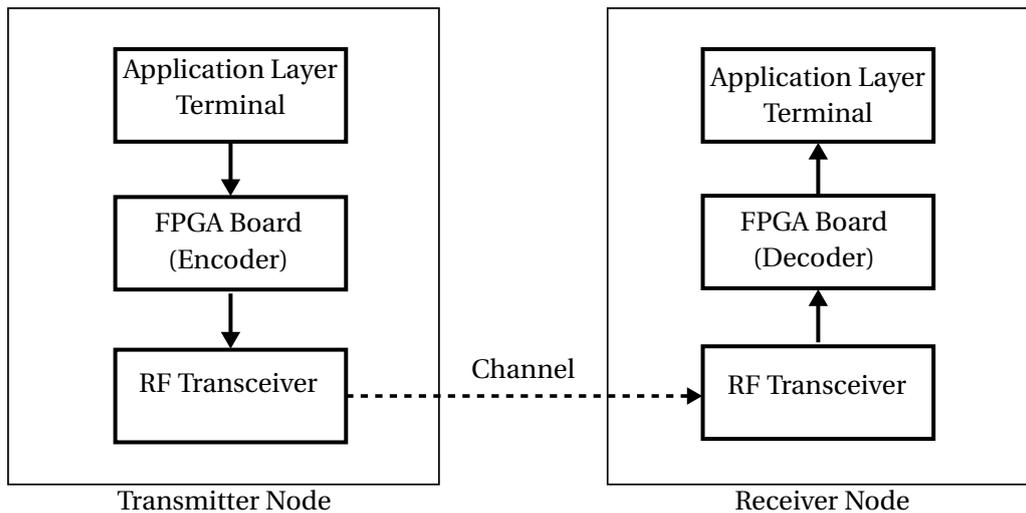


Figure 4.10: System High-level Block Diagram

4.4.1 Application Layer Terminal

The Application Layer Terminal is a computer terminal that generates and schedules data on the transmitter node and listens for incoming data on the receiving end. The terminal program is a script written in Python which generates data. The data generated is transmitted to the FPGA encoder through the serial port upon the reception of the *CTS* command. This negotiation necessitates the need for an event-based transmission rather than a timed sequence. The *CTS* is returned by the encoder as an indication for the next data batch. The data that is queued into the FPGA is time stamped and logged by the data logger as illustrated in Figure 4.11.

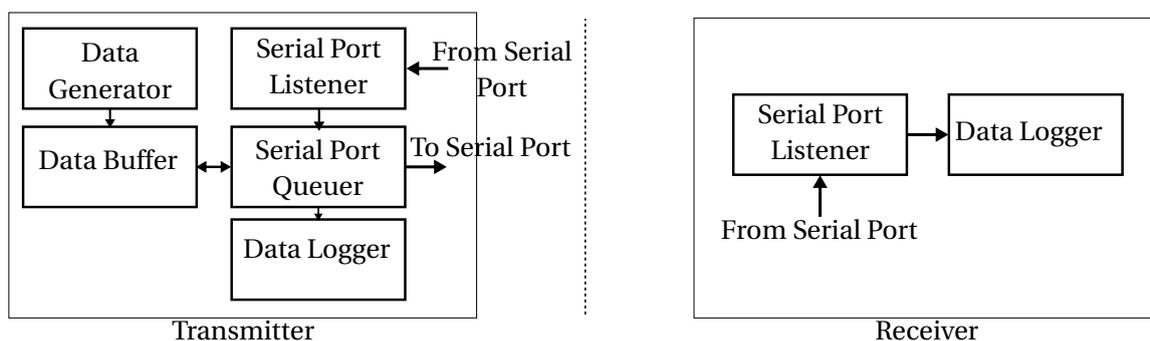


Figure 4.11: Application Layer Terminal

The receiver end terminal listens to incoming data from the serial port. If new bytes are detected, the port listener accepts data from the serial port, passes it to the data logger to add timestamps to it and logs it into a file.

4.4.2 FPGA-based Transmitter

The FPGA encoder accepts data from the upper layer and encodes it into the desired protocol format depending on the protocol mode. Figure 4.12 shows the FPGA encoder. The data from the PC terminal is transferred to the FPGA using the RS232 protocol and read serially by the UART module. The UART data listener reads the received data as bytes into a 255 byte data buffer. Even though the data buffer is 255 bytes long, it can load up to 239 in FX.25 mode as the other bytes will consist of the FCS bytes. The CRC Generator updates the CRC bits with the reception of each byte. The calculation is completed before the reception of the next byte.

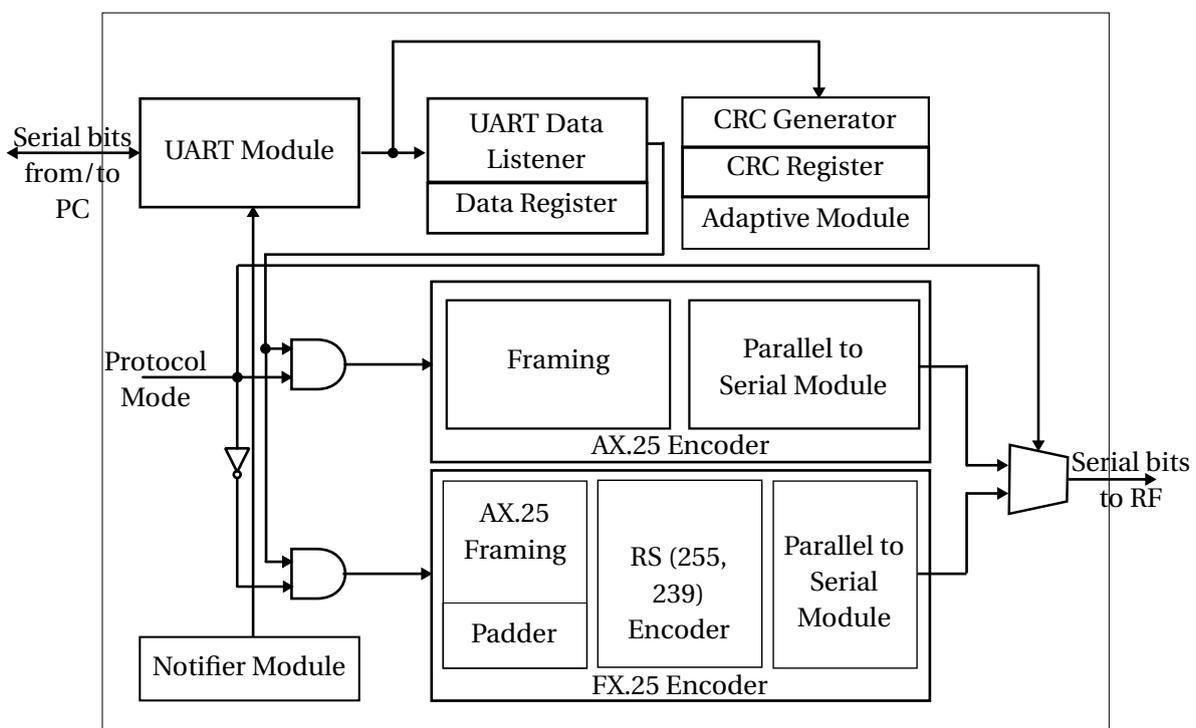


Figure 4.12: FPGA Board - Encoder

The CRC-16 checksum is required by the AX.25 error control to verify data integrity on the receiver. A CRC generator is a circuit of linear shift registers and gates designed to perform polynomial division such that at the end of the message bits, the registers b_0, b_1, \dots, b_{15} contain the checksum. The checksum is basically the remainder after the polynomial division.

The CRC-16-CCIT with a generator polynomial $g(X) = x^{16} + x^{12} + x^5 + 1$ calculates the error check bits in a bit-wise implementation with the circuit in Figure 4.13.

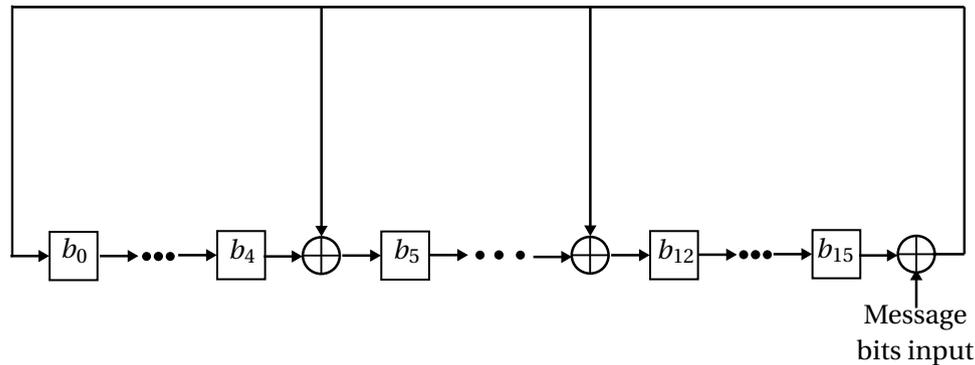


Figure 4.13: CRC-16-CCIT Generator, $g(X) = x^{16} + x^{12} + x^5 + 1$

The CRC bits are stored in the 16-bit CRC register which is read by the Framing modules when creating packet checksums. The encoder selects the protocol encoder to use based on the value of the Protocol Mode flag bit where the value '0' selects the AX.25 and '1' selects the FX.25. To assimilate as much as possible the results obtained from the simulation, the hardware configuration will in many ways resemble the protocol implemented in SatSim.

4.4.2.1 AX.25 Encoder

The AX.25 connectionless mode is implemented with a packet framing that is the same as the one designed for the simulation which can be found in subsection 3.2. The Framing module reads the Data and CRC Random Access Memory (RAM)s to create the AX.25 frame as per the AX.25 specification. This module also performs bit stuffing of the payload bits. The frame bytes are read by the Parallel to Serial module to output the packet bit by bit to the RF transmitter. Upon sending the last bit, it notifies the PC terminal by sending a *CTS* command via the serial port.

4.4.2.2 FX.25 Encoder

The general framing and transmission is the same for the FX.25 hardware except that the simplified error correction algorithm used earlier cannot work in a realistic environment. The Reed Solomon (255, 239) code is used for error correction. The primitive polynomial for the $GF(2^8)$ is chosen to be $\phi(X) = X^8 + X^4 + X^3 + X^2 + 1$ whose elements can be referred to from

Appendix 4.3. The (255, 239) code has a generator polynomial

$$\begin{aligned}
 g(X) &= \prod_{i=0}^{2^{(8)}-1} (X + \alpha^i) = (X + \alpha^0)(X + \alpha^1) \dots (X + \alpha^{(2^{(8)}-1)}) \\
 &= (X + \alpha^0)(X + \alpha^1) \dots (X + \alpha^{15}) \\
 &= X^{16} + 59X^{15} + 13X^{14} + 104X^{13} + 189X^{12} + 68X^{11} + 209X^{10} + 30X^9 \\
 &\quad + 8X^8 + 163X^7 + 65X^6 + 41X^5 + 229X^4 + 98X^3 + 50X^2 + 36X^1 + 59
 \end{aligned} \tag{4.28}$$

resulting in an encoder circuit illustrated in Figure 4.14. The multipliers were implemented using the constant multiplier approach discussed in subsection 4.2.2.2. The bytes from the message to be transmitted are fed serially, and after 239 clock cycles the output of the encoder will be reading out the shifted values of the parity bytes. If the AX.25 packet from the Framing module is less than 239, the Padding module adds redundant filler 0111110 bits until the data is byte aligned at 239 bytes. The Framing module performs the bit stuffing on the data bits.

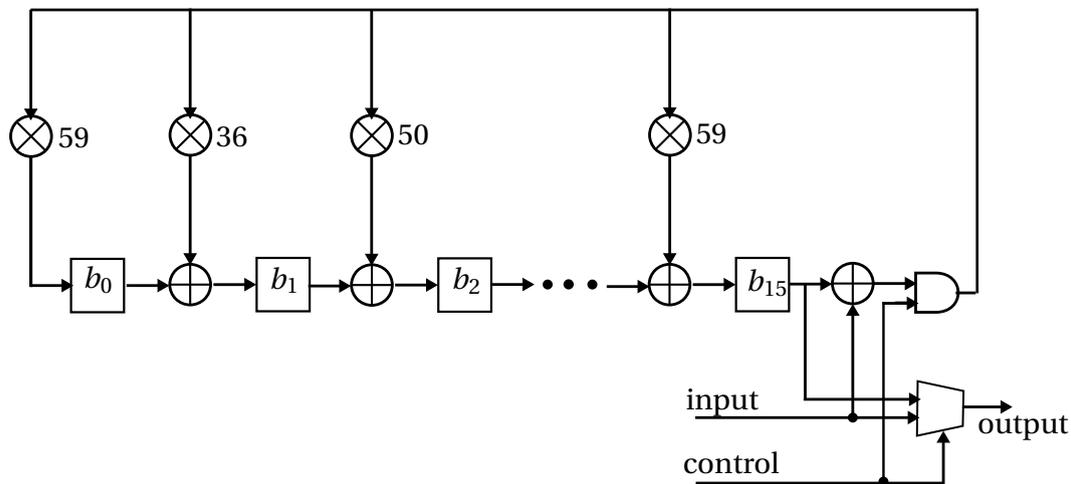


Figure 4.14: Reed Solomon (255,239) Encoder Circuit

For the first 239 clock cycles, the output multiplexer control signal is '1' allowing data into the encoder yet pushing the same copy of bytes into the output port. After 239 cycles, the control is cleared to '0' thus turning off the input into the encoder yet allowing the parity bytes out of the encoder. The bytes are shifted byte-by-byte from the 8 byte registers until the value of b_0 is read. The Parallel to Serial Queue module translates the received bytes into a bit by bit stream into the RF transmitter. This is necessary for RF transmission since the data is serially transmitted on the RF side. This module also prepends the Correlation tag, Preamble and appends the Postamble bytes to the data.

4.4.3 RF Transceiver

The RF Transceiver board is a 869 MHz Adeunis ARF6921D half-duplex board with a transmission power up to 500 mW. The transceiver can be operated as either a transmitter or

receiver depending on the value of the RF Rx/Tx mode selection pin. The board is unavailable when changing mode for approximately 2.2ms from reception to transmission, 3.1ms for transmission to reception. The switching timing diagram is found Appendix E.1. The RF block diagram is shown in Figure 4.15.

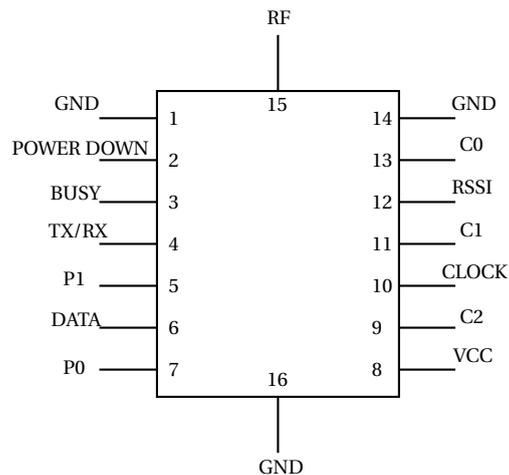


Figure 4.15: RF Transceiver Block Diagram. Adapted from [8]

Pin	Name	Direction	Function
1,14,16	GND	Power Supply	
2	Power Down	Input	Module in standby. 0V = normal mode, VCC = standby mode which limits consumption to 400 μ A
3	Busy	Output	Indicates when module is not available. This occurs when the module changes mode
4	Tx/Rx	Input	Transmitter/Receiver mode selection. 0V = reception mode, VCC = transmission mode
5, 7	P1, P0	Input	Pin P0 and P1 enable transmission power level selection.
6	Data	Input/Output	Data to be transmitted in Tx mode/ Data received in Rx mode
8	VCC	Power supply	Power supply input
9	C2	Input	Channel selection
10	Clock	Output	Clock recovery for data synchronisation
11	C1	Input	Unused
12	RSSI	Output	Indicates RF level received
13	C0	Input	Unused
15	RF	Input/Output	RF output/Module antenna

Table 4.6: Adeunis ARF6921D RF Board Interface. Adapted from [8]

The RF board is powered by a 3V regulated voltage with available current of 600 mA [8]. The RF board can transmit at four different power levels; 14 dBm, 20 dBm, 23 dBm, 27 dBm which

is configured by changing the values of P0 and P1 according to Appendix E.1. The serial data from the FPGA encoder is fed into the data pin in synchronisation with the RF Clock during transmission. The transmitter clocks data on the rising edge as illustrated in Figure 4.16.

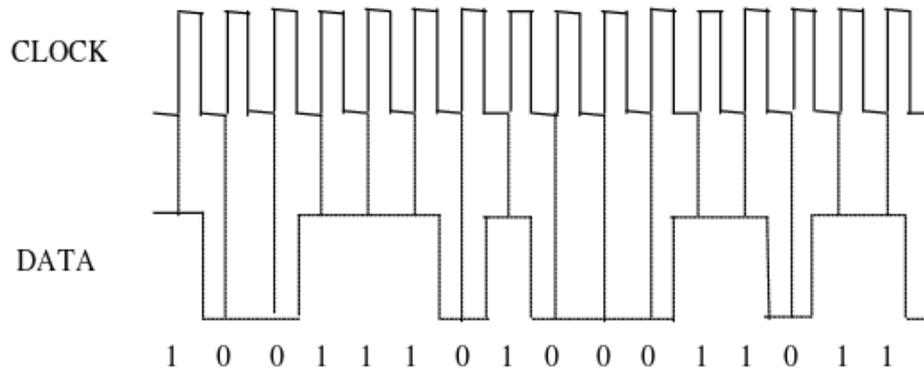


Figure 4.16: RF Clock and Data Synchronisation. Adapted from [8]

On the receiver, data is read from the same data pin into the FPGA decoder. The RF signal is Frequency Shift Keying modulated. The full specifications for RF transceiver are given by Appendix E.2.

4.4.4 FPGA-based Receiver

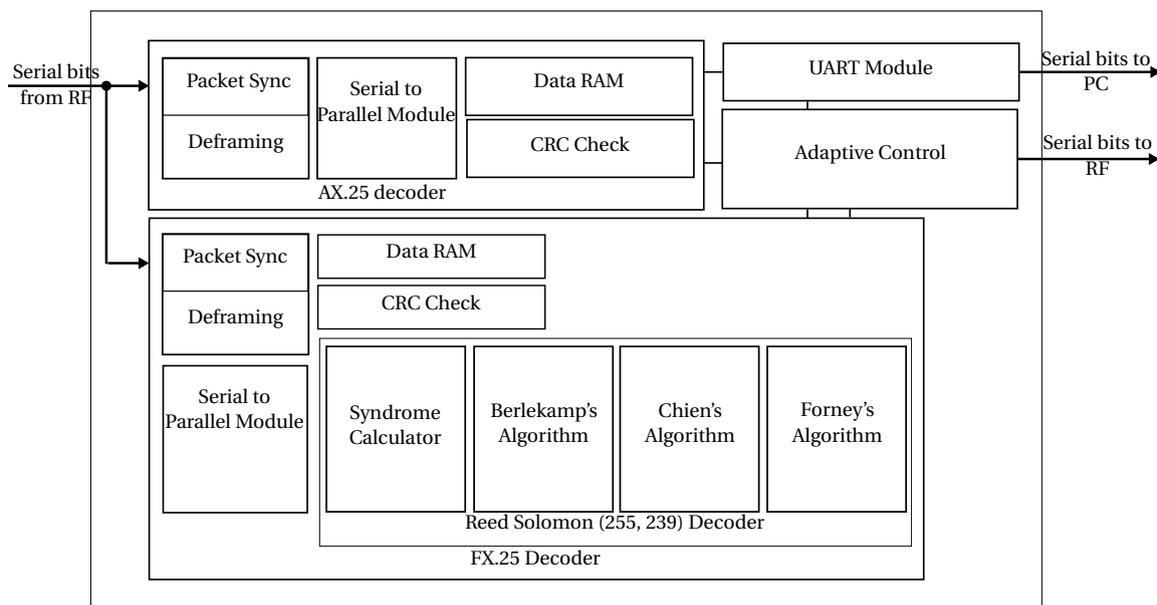


Figure 4.17: System Decoder Block Diagram

The receiver block diagram is shown in Figure 4.17. The decoder parallel decoding enables it to decode either AX.25 or FX.25 packets seamlessly.

4.4.4.1 AX.25 Decoder

The AX.25 Packet Sync continually listens to the RF and buffers the serial input into an 8-bit shift register. If the shift register value is the AX.25 start sequence 0x7e, the decoder state machine transitions to De-framing and Data Capture. In this state, the decoder reads the incoming data, discards the address bits (since only two nodes are used in this communication), discards control bits (the protocol only uses UI frames) and store information bits into data RAM. The data capturing process runs so long as the end tag has not been detected or until the maximum number of allowed information bytes (253 bytes) is reached. The 253 maximum is considered the absolute maximum with the assumption that it can go up as much if a pure AX.25 packet was transmitted otherwise for a packet wrapped in a FX.25 frame, it will be far less as the FX.25 codeblock spans 239 bytes. The De-framer also destuffs the incoming information bits.

The destuffed byte buffer passes through a CRC calculator before being written to the RAM. The CRC circuit is similar to the CRC generator in Figure 4.13. The principle of operation of the CRC checker is that since the same circuit was used to generate the checksum which is essentially the remainder, dividing the polynomial with its remainder polynomial appended to it should give a remainder of 0x0000. That is if the data has not been altered, otherwise the remainder will not be 0. Upon detecting the end tag, the CRC buffer value is checked. Every valid frame is forwarded to the PC terminal via serial port. Every corrupted frame is dropped, and a packet drop indicator message is sent to the PC terminal. A simplified algorithm for the decoder is shown in Algorithm 1.

Algorithm 1 AX.25 Decoder

```

1: Packet Sync
2: while RF_BUF  $\neq$  0x7E do
3:   Shift RF bit in
4: end while
5: read in, and drop address, control bits
6: while END_TAG not detected or max read bytes read do
7:   shift RF bits in
8:   bit destuffing
9:   if RF_BUF has a new byte then
10:    save byte to RAM
11:    calculate CRC
12:   end if
13: end while
14: if CRC_BUF = 0x0000 then
15:   send information bytes from RAM to PC
16: else
17:   drop packet, send packet drop notifier to PC
18: end if
19: Go to Packet Sync

```

The AX.25 decoder can also decode FX.25 packets since the inner wrapped AX.25 packet is not altered. If a FX.25 packet is received, the preamble, correlation tags, and postamble will not be detected and will thus be discarded. After the shift register has read in the AX.25 start tag, the decoder state machine thereafter processes the rest of the AX.25 packet in a predefined manner.

4.4.4.2 FX.25 Decoder

The FX.25 Packet Sync is different from the equivalent in AX.25 due to the fact that the FX.25 signal is locked based on the received correlation tags rather than the preamble. The decoder is synced once an 80% correlation is achieved. The decoder's Finite State Machine (FSM) next state goes to reading the preamble after which a AX.25 decoding process similar to algorithm 1 occurs. If the data is found to be corrupt, instead of dropping the packet, the RF data is read until 255 bytes have been received. The 255 byte data block is the Reed Solomon codeblock that is handed to the Reed Solomon decoder.

The RS(255,239) syndrome polynomial given by

$$S_i(x) = S_0 + S_1X + S_2X^2 + \dots + S_{15}X^{15} \quad (4.29)$$

The full **Syndrome** circuit is illustrated in Figure 4.18. The syndrome is computed in 255 clock cycles with the R(x) byte in position per cycle. It is already known from the CRC calculation in the previous stage that the syndrome computation will give a non-zero value. The encoder issues a *START* pulse signal to the syndrome calculator to initiate start of syndrome calculation.

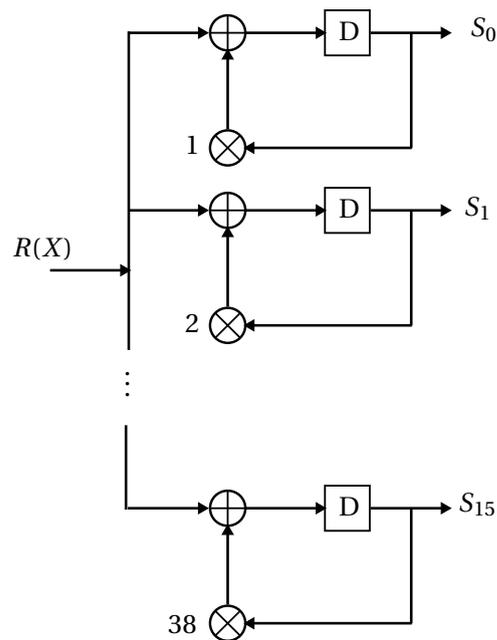


Figure 4.18: Reed Solomon (255,239) Syndrome Circuit

At each clock cycle, the syndrome calculator multiplies the received byte by the roots of the generator polynomial $g(x)$ from α^0 to α^{15} . The value of each syndrome is accumulated on the register D which after 255 clock cycles contains the coefficients, $R(\alpha^0), R(\alpha^1), \dots, R(\alpha^{15})$. The syndrome calculator triggers the Berlekamp circuit to start the locator decoding.

The **Berlekamp** circuit is shown in Figure 4.19. The syndrome registers are not a standalone memory but from the previous step. The $\sigma^{(\mu+1)}$ and $\sigma^{(\rho)}$ are each 9-byte memory arrays that store the coefficients of the current and ρ^{th} error locator polynomial respectively. The $d_{\mu, \rho, \mu}$ 8-bit registers stores the discrepancy, the last stage with non-zero discrepancy, and the inverted value of the discrepancy at that stage respectively. The $\mu - \rho$ in equation 4.23 is accounted for by the shifting circuitry control. Initially the registers for $\sigma^{(\mu+1)}$ and $\sigma^{(\rho)}$ contains the values of 1 to match the values of rows 1 and 0 of Table 4.5 respectively.

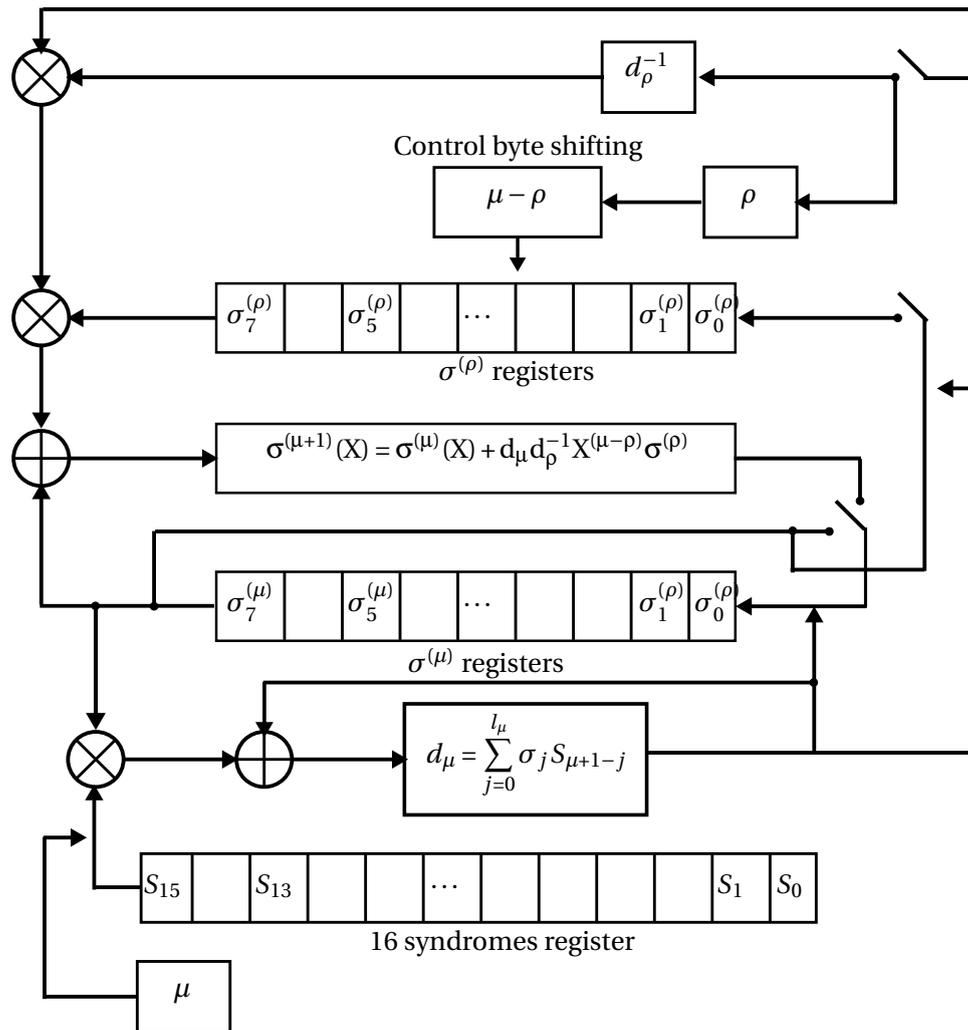


Figure 4.19: Berlekamp Circuit

Upon reception of the *START* pulse, the circuit checks the d_μ register. If it has a value of 0, then the current value of σ is retained implying that the σ feedback switch is closed. If not, a new value of σ is calculated which takes the value of the σ registers and adds them to the output of the multiplier from $\sigma^{(\rho)}$. This process involves shifting the $\sigma^{(\rho)}$ registers $\mu - \rho$ bytes to the left, multiplying each coefficient of $\sigma^{(\rho)}$ by $d_\mu d_\rho^{-1}$ and adding it to the corresponding terms of the σ . The Berlekamp multiplication is different from the multiplications from the previous subsections in that neither of the values to be multiplied is known in advance. The multipliers for the Berlekamp uses the principle for deriving constant multipliers, but has more complexity and resource requirements. The inverse calculation is based on an inverse ROM lookup table for the inverse of the field elements.

The circuit flow can be illustrated by Algorithm 2.

Algorithm 2 Berlekamp's Algorithm

```

1: INIT: Set registers  $\sigma^{(\mu+1)} \leftarrow 0x01, \sigma^{(\rho)} \leftarrow 0x01,$ 
2: Set registers for  $\mu \leftarrow 0x00, l_\mu \leftarrow 0x00, (\mu - l_\mu) \leftarrow 0x00$ 
3: NEXT_ITER:
4: if  $d_\mu$  register value =  $x00$  then
5:   Go to COMPUTE_ $d_\mu$ 
6: else
7:   Go to COMPUTE_ $\sigma$ 
8: end if
9: COMPUTE_ $d_\mu$  :
10: while  $j \leq l_\mu$  do
11:    $d_\mu += \sigma_j S_{\mu+1-j}$ 
12: end while
13: COMPUTE_ $\sigma$ :
14: if (register value  $d_\mu \neq 0x00$  and  $(\mu - l_\mu) \geq (\rho - l_\rho)$ ) then
15:   Assign to registers  $\sigma^{(\rho)} \leftarrow \sigma^{(\mu)}, \rho \leftarrow \mu, (\rho - l_\rho) \leftarrow 0x00$ 
16: else
17:   Add a  $0x01$  to the value of  $(\rho - l_\rho)$  register
18: end if
19: if (the register containing  $(\mu - \rho) > 0x08$ ) then
20:   % The degree of  $(\sigma^\rho) > 0x08$ 
21:   Go to HALT
22: end if
23: Shift the bytes  $\sigma^\rho \ll (\mu - \rho)$ 
24: MULTIPLY_DMU_DRHO: Multiply  $d_{mu}$  and  $d_\rho^{-1}$ 
25: Multiply the product MULTIPLY_DMU_DRHO by each coefficient of  $\sigma^{(\rho)}$ 
26: Add the coefficients of the operation from the previous step to the coefficients of  $\sigma^{(\mu)}$ 
27: CHECK_ITER:
28: if  $\mu = t$  then
29:   Go to SUCCESS
30: else
31:   Go to NEXT_ITER
32: end if
33: SUCCESS: Error locator polynomial found
34: HALT: Errors cannot be corrected

```

This process is succeeded by calculating and updating the discrepancy which is achieved by summing the terms of multiplication operation of the coefficients of the stage error locator and the corresponding term of the syndrome $\sigma_j S_{\mu+1-j}$. An address counter for j is used to move the address pointer at each iteration. After getting the discrepancy, and updating the registers for $d_\mu, l_\mu,$ and $(\mu - l_\mu)$, the value of μ is incremented and the process starts all over again. The iteration is halted when $\mu = 2t$ and the error locator is found or when the degree of the stage polynomial $\sigma(\mu + 1)$ is greater than t . This occurs when the shifting circuitry is attempting to shift bytes to the left more than the 9-byte size. In the latter case, the number of errors has exceeded the correction capability. If the locator decoding succeeds, the Chien search algorithm is triggered.

The **Chien** search is implemented based on Figure 4.8 with the values of the error locator

registers as 8 bits, $\sigma_1, \sigma_2, \dots, \sigma_8$. The second logic unit in the circuitry 4.8 is the multiplier for the GF element raised to the powers 1, 2, \dots , 8. The power function is implemented in hardware based on the principle of multiplication introduced in subsection 4.2.2.2. For the second multiplicand for instance, α^2 , the power circuitry will be designed as follows from the element $\beta = b_0 + b_1\alpha + \dots + b_7\alpha^7$

$$\begin{aligned}
 (\beta)^2 &= b_0 + b_1\alpha^2 + b_2\alpha^4 + b_3\alpha^6 + b_4\alpha^8 + b_5\alpha^{10} + b_6\alpha^{12} + b_7\alpha^{14} \\
 &= b_0 + b_1\alpha^2 + b_2\alpha^4 + b_3\alpha^6 + b_4(\alpha^4 + \alpha^3 + \alpha^2 + 1) + b_5(\alpha^6 + \alpha^5 + \alpha^4 + \alpha^2) + \\
 & b_6(\alpha^7 + \alpha^6 + \alpha^3 + \alpha^2 + 1) + b_7(\alpha^7 + \alpha^4 + \alpha^3) \\
 &= (b_0 + b_4 + b_6)\alpha^0 + (b_1 + b_4 + b_5 + b_6)\alpha^2 + (b_4 + b_6 + b_7)\alpha^3 + (b_4 + b_5 + b_7)\alpha^4 + \\
 & b_5\alpha^5 + (b_3 + b_6)\alpha^6 + (b_6 + b_7)\alpha^7
 \end{aligned} \tag{4.30}$$

The full multiplier combinational logic circuitry is derived from each of the terms from 4.30. The search for the roots takes 255 clock cycles to exhaust all the elements, and the solution of a zero at location i indicates that the inverse of that element is the location of the error. The inverse logic is achieved through an inverse ROM lookup table.

The last but one step is the finding the error values at the locations using **Forney's algorithm**. The algorithm uses values from the previous computations namely the syndromes and the error-locator coefficients which will be accessed from the respective RAMs. The address counters move the address pointers for the syndromes and $\sigma(X)$ to find the numerator of equation 4.27. This is achieved by circuit 4.20 whereby the products for $\beta \times S$ are summed and stored on the 8-bit register $\Gamma(\beta^{-1})$ at each clock cycle. On completion of the summation loop, the value is XORed with 0x01. Much like in the Chien search, the inverse of any element is achieved through a lookup table. The denominator, on the other hand is obtained by calculating the $\beta_i\beta_l^{-1}$ through the multiplier and the product added to 0x01. The process loops for all the found error locations and finally after 2ν clock cycles, the values from numerator and denominator registers will be added. This is the same as division due to the multiplicative inverse principle.

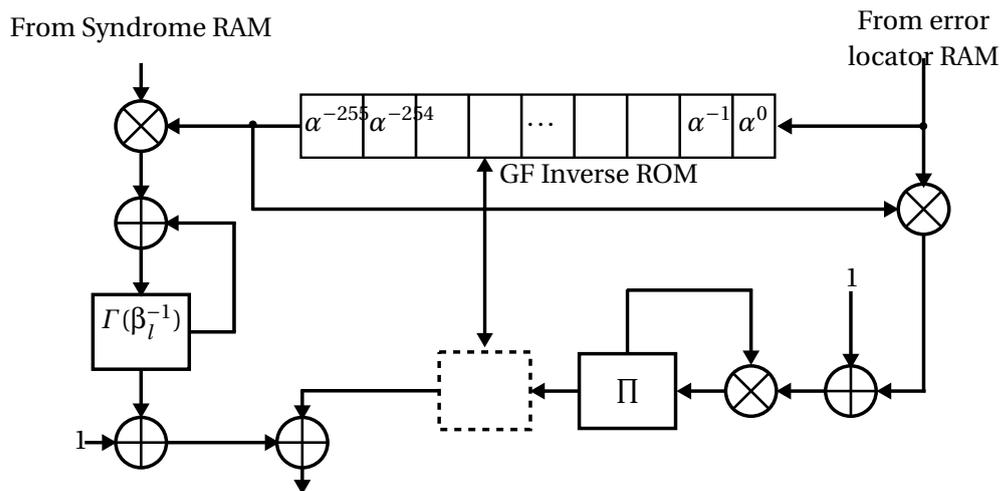


Figure 4.20: Forney's Error Magnitude Calculator Circuit

The design then requires that the denominator get its inverse from the inverse lookup table and is added to the numerator since the denominator is a numerator with a negative power.

Finally, the original $R(x)$ at the v locations and the found error values are fed into an XOR gate to correct the error.

The data from the decoder is read back by the AX.25/FX.25 decoder to validate whether the data was successfully corrected by re-computing the CRC checksum for the corrected data. If the checksum is still not equal zero, then the number of errors incurred were more than 8. In a scenario this, the packet is dropped and a packet notification message is sent to the PC.

4.4.5 Adaptive Module

The AFX.25 switching algorithm is implemented in a hardware based on its principle of operation outlined in 3.4.2. The transmitter, will use a TRANSMIT-AND-WAIT state machine to change the RF transceiver from transmitter to reception mode to listen to switch commands since the RF boards are half-duplex.

The comparisons of the packet success or drop rate was implemented on hardware using the cross multiplication comparison method for fractions. This solution is economical because it eliminates the gates required by floating point division. As an example, for a 0.95 threshold, the values 95 and 100 are stored on registers. If at some window, the number of non-erroneous packets is 23 and the total number of received packets is 50, the comparison circuit will be evaluating if (23×100) is greater than (95×50) .

4.5 Testing and Results

4.5.1 ModelSim Simulation Verification

The protocol VHDL design was first simulated on ModelSim to evaluate if it gives the expected response. Due to the complexity of ModelSim, the design could not be simulated as a complete system. To simulate the whole project would have required that the Python Data Scheduler and Serial Port module queues the data packets into ModelSim and reads the data back from the output in some "closed loop" format. This would take a significant amount of time to implement. The ModelSim simulation is necessary to verify the correct functionality of the VHDL code. A semi "closed loop" simulation was achieved through a Smart design in Libero and was simulated on ModelSim as illustrated by Figure 4.21.

The SmartDesign schematic specifies the simulation components as the hardware testbench, transmitter, channel, and receiver. The transmitter and receiver components are the actual sub-units of the implemented design and the other components are testbench components. The hardware testbench emulates the physical hardware components of the system, namely the RS232 interfaces which provide the PC-to-FPGA connectivity. The RS232 Tx module accepts one byte at a time from the upper layer and outputs one bit serially to the UART module. The bytes from the upper layer are an output from the testbench driver script. The UART core receives the serial data and forwards it to the Transmitter module. The GenericControlTx and RS encoder modules perform framing and encoding on the received payload data. After data encoding, the data is propagated through the channel that consists of a noise generator module, the BitFlipper. The BitFlipper will at certain interval(s) flip the channel bit. If the transmission packets fall within that interval, then the data received on the decoder will be invalid.

The receiver decodes the serial RF bits, check for errors, correct corrupt packets and forward the data/response to the RS232Rx via the UART core. The output is observed on the Model-Sim console.

Consider the following testbench for a transmitting node that requires to transfer the message "Hello":

- The testbench creates a byte RAM containing 0x48656c6c6f (hexadecimal equivalent)
- The testbench initiates a transmission session by asserting the LD pin
- One byte is read into the testbench output every baud clock cycle
- The last data byte is succeeded by a delimiter byte

With the protocol on FX.25 mode, after the last byte of the testbench is received, the Reed Solomon encoder creates the codeblock and the transmitter propagates the bits into the channel. Figure 4.22 shows the RS codeblock transmitted with leading AX.25 start tag 0x7E. The second signal inside the red circle shows the bit that was flipped resulting in the first letter "l" in "Hello" to be corrupted into 0x4C.

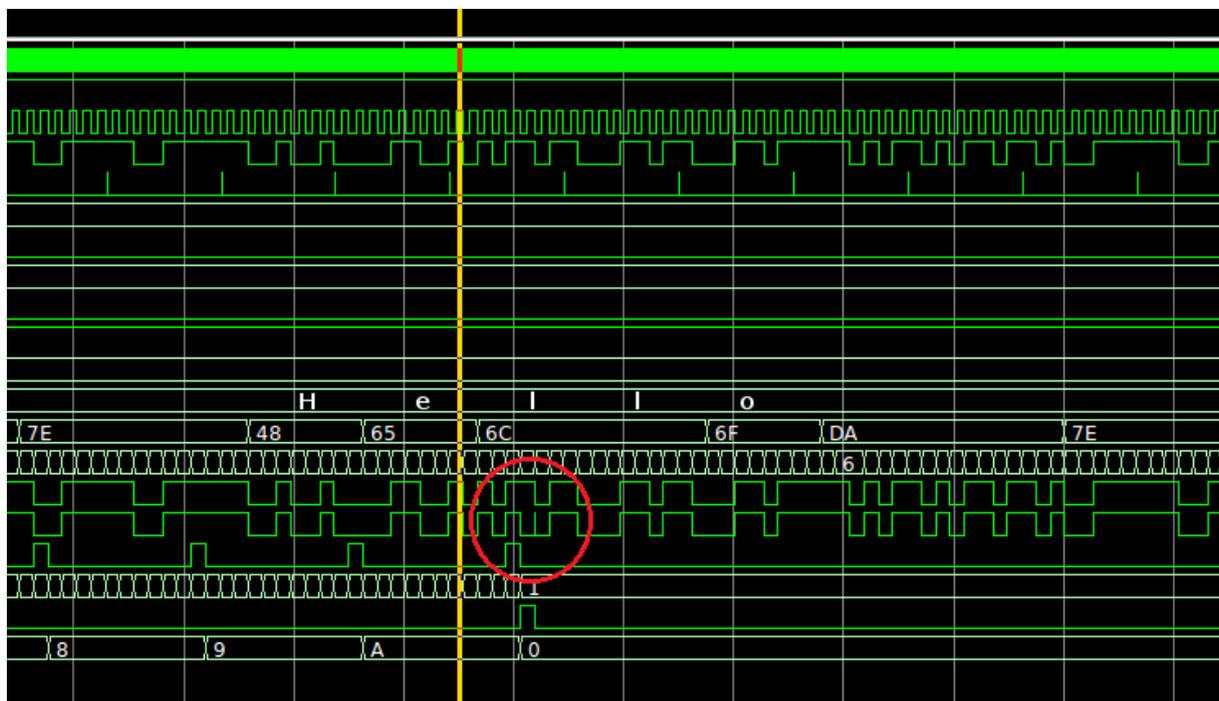
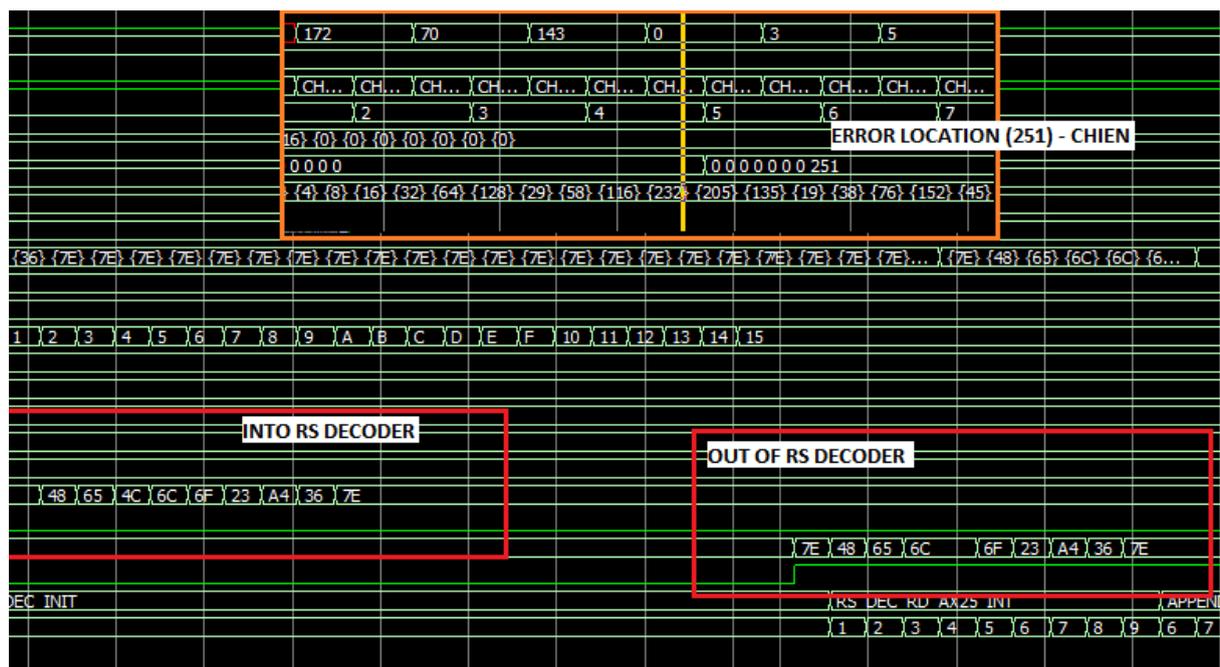


Figure 4.22: Simulation showing channel noise injected in the data

The receiver reads in the codeblock and detects that the checksum is not zero. The receiver then initiates error recovery with the RS decoder module as shown in Figure 4.23. The decoder goes through the decoding process, calculating syndromes, finding error locator and solving it, finding and the error magnitudes and correcting the invalid bits. The Chien search located the error at location 251 along the yellow line in Figure 4.23. The 251 position is where the "1" was in 255 FX.25 frame since the first 254th byte is the AX.25 delimiter. The result of the decoder can be observed from the output console. The decoder succeeded in correcting the 0x4C into 0x6C which was the original byte. After several testbenches with different test inputs, the decoder was verified to function as expected.



incrementally step forward at 1-second intervals with the help of a stop watch. The human movements were carefully observed so that the steps are equal for all the experiments so that the mobile node reaches the second floor after 80 seconds and back at the laboratory after 140 seconds. The signal deteriorates as the wall thickness changes, which alters the diffraction rate of the signal. The signal-to-noise ratio at the third floor is lower compared to the ground floor close to the fixed node.

The tests were conducted for a fixed size 150-byte packet size with a baud rate of 9600 bps where the packets are generated on demand. The results were logged and the combined results are displayed graphically on Figure 4.24.

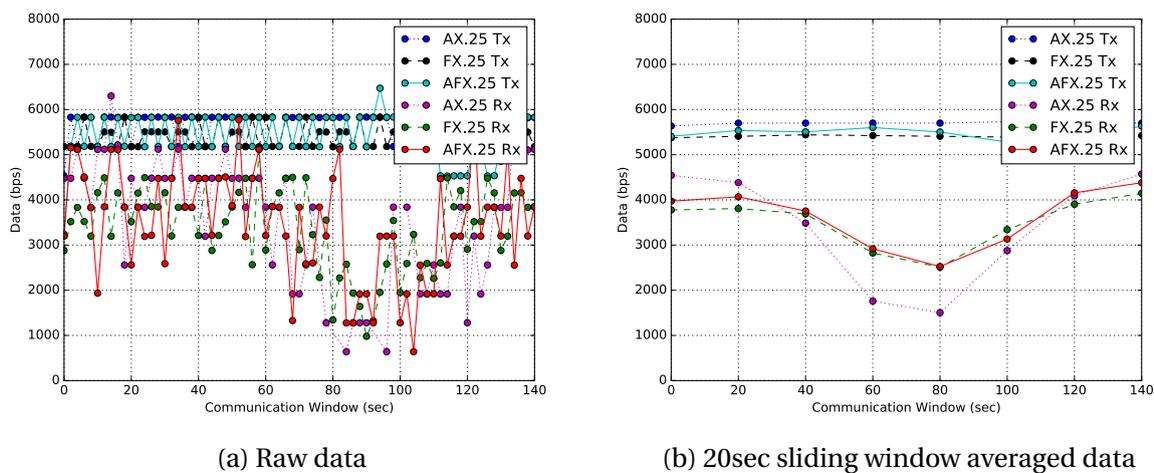


Figure 4.24: Throughput Curves for AX.25/FX.25/AFX.25 Indoor Tests

Figure 4.24a shows the throughput evaluated per second. The protocol performance cannot be visualised correctly due to the scatter of the individual points. It is for that reason the data points were averaged over a 20 seconds sliding window resulting in Figure 4.24b. The 0-second point is the start point of the mobile movement, the 80-second is the farthest point (second floor) and at 140 seconds the mobile node is back at the starting point. It can be observed that at the start point, the AX.25 performance is the highest but drops as the mobile node moves away from fixed node. At the farthest distance at 80 seconds, the AX.25 throughput is the worst. The FX.25 throughput does not have the greatest performance near the transmitter node but at the farthest point (80 seconds). This displays its strength in correcting corrupted packets. The AFX.25 throughput follows the AX.25 curve from 0 to 40 seconds, the FX.25 from 40 to 110 seconds, and the AX.25 from 110 to 140 seconds. This indicates its adaptability functionality. The AFX.25 does have an initial throughput less than the AX.25 because on system start-up, the protocol operates in FX.25 mode. The highest throughput from the experiment is 6000 bps which is a deviation from 9600 bps, the theoretical throughput. This is so because of the time losses during the transmission; the time it takes for bytes to be serially transferred from the PC to the FPGA and from the FPGA to the PC. Secondly, the transmitter always wait for a response from the receiver every 20 seconds. The waiting time

also accounts for the 3 to 5 ms required by the RF transceiver to change from transmission to reception or reception to transmission mode. The total data received for the 140 seconds communication is given in Table 4.8.

Protocol	Total Data Received (KB)	Relative Performance (%)
AX.25	92.5	100.0
FX.25	94.1	101.7
AFX.25	96.5	104.3

Table 4.8: AX.25/FX.25/AFX.25 Data Received over 140 seconds indoor tests

A terrestrial environment was thereafter assumed. It does not have exactly the same characteristics for a space-to-ground communication but was adopted because of its ability to mimic the rise and fall satellite trajectory on the horizon from an observer ground station. The task was to identify a road with a semi-circular bend where the peak of the bend would be the ground station location.



Figure 4.25: Terrestrial Test Environment

Figure 4.25 shows the test environment where points A and A' are zones of high signal loss and B is clear line of sight. The ground station node is fixed at point B. The mobile node moves from B to A', then from A' it passes via B to A and back to B for each session. As the mobile unit moves from B to A or A', the signal-to-loss ratio lowers gradually until point A or A', where the ground node is out of sight. This is so because of the elevated topography that

provides an obstruction as you go along the road. Since the road bend length is small, it was required that the RF modules were operated on minimum transmitting power.

The experiment was conducted over a test period of 8 minutes with fixed sized 156 byte packets generated on demand. The node movements were human aided as with the indoor test. However, the outdoor setting allowed for footstep calibration. The ground along the path was marked at equal intervals so that each marked distance is covered per second. Three experiments were performed with each protocol (AX.25, FX.25, AFX.25) per session. The results were logged and are represented by Figure 4.26.

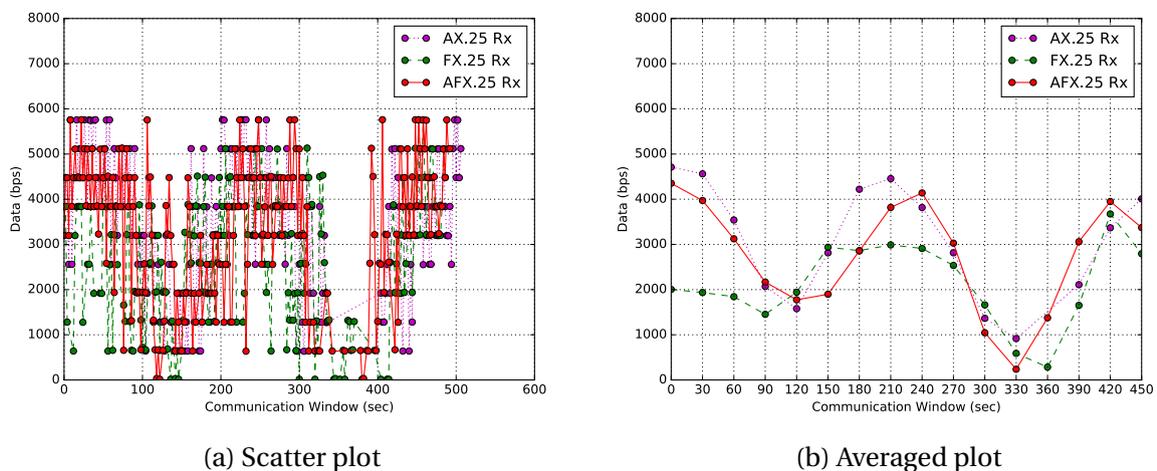


Figure 4.26: Throughput Curves for AX.25/FX.25/AFX.25 Terrestrial Tests

The 0-second point is at the starting point, B. Point A' is at 120 seconds, point B is at 210/240 seconds, point A at 330 seconds and point B at 420 seconds. Having a closer look at the 120 to 330 seconds region, it can be observed the FX.25 has a better throughput than the AX.25 but drops at around 210 seconds when the AX.25 is at its maximum. This has been established by the simulation chapter that at the clear line of sight, the FX.25 FEC symbols waste the bandwidth. During this window, the AFX.25 switching capability is observed as the highest overall data received for AFX.25 is high.

However, some unexpected overlapping in performance is observed, for example at 330 seconds, the AFX.25 has a lesser throughput than the FX.25 and AX.25. This is not expected as the AFX.25 is a hybrid of the FX.25 and AX.25, so its throughput is expected to be higher. Ideally this would be the case for parallel tests run at the same time with identical equipment but it was not the case. The results were concluded from three standalone experiments with tried timings and the results vary for each experiment. Firstly, the data generation algorithm is random for each experiment. Secondly, in as much as the same path is followed for each time step, but the errors imposed on the data cannot be exactly be similar for all the experiments. Thirdly, the time steps are within 1 meter accuracy.

4.6 AFX.25 Implementation Real Estate Cost

This section summarises the design resource utilisation summary after synthesis on a Fusion MFS1500 device running on a 24 MHz clock.

Table 4.9: Resource Utilisation for Fusion MFS1500

Module	No. of Core Cells	No. of IO Cells	No. of Block RAM (255 × 8 bits)	Max. Freq. (MHz)	Min Period (ns)
RS Encoder	355	21	0	70.8	14.109
Transmitter (incl. RS enc)	1131	14	2	70.8	14.109
RS Decoder	5857	52	1	45.6	21.907
Receiver (incl. RS dec)	7925	13	4	45.6	21.907
Total (Tx and Rx)	9056	27	6	45.6	21.907

From the analysis in Table 4.9, the number of IO cells for both the RS encoder and decoder is higher than the overall because their calculations were done on standalone modules. A bigger number of IO cells are not used in the combined transmitter/receiver because the IO cells now become port connections to the GenericController module. The total estimated power consumption of the receiver is 55.225 mW and that of the transmitter 34.167 mW amounting to 89.392mW for the full design excluding RF front end electronics consumption.

The smallest Microsemi FPGA that can be used for the design is the IGLOO AGL600V5 which has 13,824 core cells, 285 I/O cells and 108x1024kbits of RAM. The resource utilisation for the IGLOO AGL600V5 is shown in Figure 4.10

Table 4.10: Resource Utilisation for IGLOO AGL600V5

Module	No. of Core Cells	No. of IO Cells	No. of Block RAM (255 × 8 bits)	Max. Freq. (MHz)	Min Period (ns)
RS Encoder	430	21	0	27.0	37.1
Transmitter (incl. RS enc)	1440	7	2	27.0	37.1
RS Decoder	4906	22	1	15.7	63.7
Receiver (incl. RS dec)	6868	6	3	15.7	63.7
Total (Tx and Rx)	8308	13	5	27.0	37.1

Chapter 5

Conclusion and Future Work

5.1 Conclusions

The project objectives were met; an adaptive AFX.25 protocol has been designed and implemented on a FPGA.

5.1.1 Protocol Simulations

The choice of AX.25/FX.25 was heavily influenced by the popularity of the AX.25 amongst amateur developers[9]. Le Roux [1] initiated the idea of hybridisation of the two protocols and conducted a performance simulation. Whilst the current trend has a bigger share on the AX.25, it was imperative that a further investigation is done for other popular data link protocols. The CCSDS standards were found to be trendy too, more especially with big satellites. That motivated the protocol simulation for the AX.25, FX.25, CCSDS TC in chapter - 2. The results of the simulation proved that the AFX.25 performs better.

It was noted that even though each protocol may have a larger payload size like the CCSDS TC, the protocol's performance with large packets is not as good as with packets with an average size of around 200 bytes. This is because larger packets have a higher probability of being corrupted than smaller ones. Upon corruption, the amount of data lost is significantly larger than if a smaller packet is lost. It was also observed that with the FX.25/AFX.25, the protocol is at its best around that margin. Theoretically, the FX.25 would have high performance if the codeblock information contains a full AX.25 data structure without padding as the padding does not provide useful content. However, the number of information bytes cannot be determined exactly beforehand due to the AX.25's bit stuffing algorithm.

Even though the project was geared towards a hardware design, the importance of a network simulator like SatSim cannot be underestimated as it was handy in verifying the protocol's functionality. Another advantage of using a simulation tool is the short development and

testing time.

5.1.2 Hardware Implementation

Most of the project time was spent in designing and implementing the error correction codes hardware. The field of error correction has been studied for years and many applications using the codes have been developed over the years. However, there are gaps in the literature between the concepts explained in books and their practical implementations in hardware. There are some literature sources that discuss the concepts and applications but in a more abstract mathematical manner. The simplest approach for inexperienced developers will be to adapt existing codes written in high level languages like C or Python to VHDL. Some of the codes can be used, but in most cases it does not translate well for hardware designs, resulting in unnecessarily long hours of debugging and frustration. Irrespective of the time spent in understanding the theory behind them, there is a reward in designing from concept directly to the hardware architecture in a non software approach.

The terrestrial tests showed that Le Roux's [1] idea of the AFX.25 yields good performance metrics. However, the improvements could not be quantified down to an exact value. This limitation was because of the way the tests were conducted. The terrestrial test environment didn't represent the exact rise and fall time of a typical LEO satellite trajectory. Secondly, the test data generation was probabilistic which varied the results each test session. At simulation level, this can be overcome by running several simulations with same input parameters and averaging the results over the number of repeats. But the tests could not be performed up to that level due to time constraints and the available workstation power when out in the test field.

5.2 Future Work

Having designed and tested the adaptive protocol on hardware, the following ideas are considered for future work. The current implementation's adaptability algorithm switches between code rates 239/255 and 1. A full adaptive stack could be implemented to cover the best performance code rates compatible with the FX.25 outlined by Table 3.2. Not all the codes will be implemented as an increase in correction capability adds a non-linear response to the circuit complexity and decoding time. An example would be an interleaved code which can enhance the probability of a packet being corrected due to the randomisation of the packet bits. An improperly designed interleaver circuit can, however, use a big chunk of resources and add an increase to the overall system latency. The decoder can correct almost all the errors if they are less than the code rate in use otherwise it fails. When the latter occurs, the

system needs a reporting and resend mechanism for the corrupted packets. That will require that the error correction will trigger the ARQ module on decode error.

This loss rate using equation 3.1, does not take into consideration the number of unknown packets that do not reach the receiver. It could be considered as part of improving the protocol to incorporate the number of packets that are dropped in the channel. This can be achieved by using sequence numbers. The resultant system can manage sequence counters on the transmitter and receiver. If the packet received has a sequence number that is greater than the expected sequence number, the total number of packets can therefore be updated by adding the difference. Thus, the new $r_l^{(a)}$ can be

$$\text{Packet loss rate, } r_l^{(a)} = \frac{\text{Packets received in error in } t}{\text{Total packets received in } t + \text{Packets lost in } t} \quad (5.1)$$

The current switching method is based on thresholds configured into the switching module. However, this can be improved by adapting machine learning algorithms.

Wai and Yang [43] pointed out that the Berlekamp algorithm uses fewer resources than the Euclidean algorithm for finding the error locator polynomial. However, they also noted that the Euclidean algorithm used less resources on the Xilinx Virtex4 [44] architecture. This was attributed to the fact that FPGA vendors use different algorithms for synthesis. Since real estate is one of the key watch areas on FPGAs, it would be worth a while to implement the Euclidean algorithm on the FPGA and compare the resource utilisation and performance.

Appendices

Appendix A: Cubesat Communication Survey 2015

Version 9; Page 1 of 2. This chart shows the 256 total CubeSats deployed in orbit so far, for a total of 499 Units. March 10, 2015.

Bryan Klofas. bklofas@gmail.com. Green are University CubeSats; Red are Commercial or Private; Blue are US Government.

Launch	Satellite	Object	Green	Size	Radio	Downlink	Satellite Service	Power	TNC	Protocol	Data Rate/Modulation	Downloaded	Lifetime	Antenna	Status	Updated
NASA/JPL/Bochs 28 June 2003	AAU CubeSat	2786	1U	Wood & Douglas SX450	437.475 MHz	amateur	500 mW	MX909	AX.25, Mollotex	9600 baud GMSK	1 kB	3 months	dipole	Dead	April 2013	
	DTUat-1	2784	1U	RFMID RF2905	437.475 MHz	amateur	400 mW	AX.25	AX.25	2400 baud FSK	0 ¹	0 days	cauted turnstile	DOA	April 2013	
	Conk1	2787	1U	Melex	437.880 MHz	amateur	500 mW	Custom	AX.25	1200 baud FSK	0 ¹	0 days	crossed dipoles	DOA	April 2013	
	Orte-1	2784	1U	Alinco DJ-C4 (data)	437.470 MHz	amateur	350 mW	MX614	AX.25	1200 baud AFSK	>10 MB ²	118+ months	N/A	Alive	April 2013	
	(CO-55)	2784	1U	Maki Denki (beacon)	436.8375 MHz	amateur	100 mW	PIC16LC73A	CW	50 WPM	N/A	0	0 days	monopole	DOA	April 2013
SSETI Express 27 Oct 2005	XIV	28896	1U	Nishi RF Lab (data)	437.345 MHz	amateur	1 W	PIC16G622	AX.25	1200 baud AFSK	N/A	90+ months	dipole	Alive	April 2013	
	(CO-58)	28896	1U	Nishi RF Lab (data)	437.465 MHz	amateur	80 mW	PIC16G716	CW	50 WPM	N/A	0	0 days	dipole	DOA	April 2013
	NCube-2	28897 ³	1U	Custom (beacon)	437.505 MHz	amateur	1 W	HSS/2974R ⁴	AX.25	1200 baud AFSK	0 ¹	0 days	monopole	DOA	April 2013	
	UWET	28892	1U	PR480	437.505 MHz	amateur	1 W	HSS/2974R ⁴	AX.25	1200 baud AFSK	0 ¹	0 days	emf-led dipole	DOA	April 2013	
	Cine-1.7-APD (CO-56)	2894	2U	Alinco DJ-C5	437.505 MHz	amateur	300 mW	CMX309A	AX.25, S8LL	1200 AFSK, 9600 GMSK	<1 MB ²	2.5 months	N/A	DOA	April 2013	
Minotaur-1 11 Dec 2006	GemSat-1	2965	3U+ ⁵	Microhard MHX-2400	2.4 GHz	experimental	1 W	Integrated ⁶	Proprietary	AX.25	500 KB	3 months	patch	DOA	April 2013	
	Stensat (beacon) ⁷	2965	3U+ ⁵	Microhard MHX-2400	437.067 MHz	amateur	500 mW	PIC12C617	AX.25	1200 baud AFSK	500 KB	1 month	monopole	DOA	April 2013	
Dropsy 17 Apr 2007	OSTBI	3112	1U	Yesu VX-2R	400.0375 MHz	experimental	300 mW	PIC	Proprietary	1200 baud AFSK	6.77 MB	18 months	dipole	Dead	April 2013	
	AeroCube-2	3113	1U	FreeWave FGRM	915 MHz	experimental	2 W	Integrated ⁶	Proprietary	38.4 kbaud	500 KB	1 week	patch	DOA	April 2013	
	CP1	3112	1U	TI CC1000/RF2117	437.325 MHz	amateur	1 W	PIC18LF6720	AX.25	1200 baud FSK	487 KB	2 months	dipole	DOA	April 2013	
	Libertad-1	31128	1U	Stensat	437.405 MHz	amateur	400 mW	AX.25	AX.25	1200 baud FSK	0 ¹	1 month	monopole	DOA	April 2013	
	CAPE1	31130	1U	TI CC1020	435.245 MHz	amateur	1 W	PIC16LF452	AX.25	9600 baud FSK	0 ¹	4 months	dipole	DOA	April 2013	
NASA/PSNEX/CS 26 Apr 2007	CP3	31129	1U	TI CC1000/RF2117	436.845 MHz	experimental	1 W	PIC18LF6720	AX.25	1200 baud FSK	2.0 MB ²	19+ months	dipole	DOA	April 2013	
	CP4	31128	1U	Microhard MHX-2400	2.4 GHz	experimental	1 W	Integrated ⁶	Proprietary	45 kbps	>1 MB ²	0.75 months	patch	DOA	April 2013	
	Duke-C3 (DO-64)	32789	3U	Custom Transponder	145.9-145.55 MHz	amateur	200 mW	N/A	AX.25	40 kHz CW	N/A	60 MB ¹	60+ months	turnstile	Alive	April 2013
	Seeds-2 (CO-66)	32791	1U	Musashino Electric (data)	437.485 MHz	amateur	450 mW	AX.25	AX.25	1200 baud AFSK	500 KB ²	60+ months	monopole	DOA	April 2013	
	CP6	32790	3U	Custom (beacon)	437.485 MHz	amateur	90 mW	AX.25	AX.25	1200 baud AFSK	N/A	60+ months	monopole	DOA	April 2013	
Minotaur-1 10 Sep 2009	AeroCube-3	3506	1U	FreeWave FGRM	915 MHz	experimental	2 W	Integrated	Proprietary	77 kbaud GFSK	52 MB	7 months	patch	DOA	April 2013	
	CP6	35043	1U	CC1000/RF2117	437.365 MHz	amateur	1 W	PIC18LF6720	AX.25	1200 baud FSK	0.4 KB	4 months	dipole	DOA	April 2013	
	Harvest-1	35004	1U	Microhard MHX-425	437.215 MHz	amateur	1 W	Integrated	Proprietary	AX.25	0.4 KB	0 days	dipole	DOA	April 2013	
	PharmaSat	35002	3U+ ⁵	Microhard MHX-2400	2.4 GHz	experimental	1 W	Integrated	Proprietary	AX.25	10 kbps	650 KB	10 days	patch	DOA	April 2013
	Stensat (beacon) ⁷	35002	3U+ ⁵	Stensat (beacon) ⁷	437.465 MHz	amateur	500 mW	AX.25	AX.25	1200 baud AFSK	N/A	1 month	monopole	DOA	April 2013	
ESL/Amal/PSV/CH1 23 Sep 2009	BEESAT-1	3593	1U	STE BK-77B	436.000 MHz	amateur ¹¹	500 mW	CMX309B	Mollotex	4800/9600 baud GMSK	43+ months	monopole	Alive	April 2013		
	UWET-2	3593	1U	Custom (beacon)	437.385 MHz	amateur	1 W	Integrated	Proprietary	AX.25	1200 baud AFSK	487 KB	2 weeks	dipole	DOA	April 2013
	ITU/PSAT-1	3593	1U	Microhard MHX-425	437.325 MHz	amateur	1 W	Integrated	Proprietary	AX.25	19200 baud	0.4 KB ²	43+ months	dipole	DOA	April 2013
	Beeline/CC1050	3593	1U	Beeline/CC1050	437.325 MHz	amateur	350 mW	AX.25	AX.25	1200 baud FSK	N/A	0	0 days	dipole	DOA	April 2013
	SwissCube	3593	1U	Butler oscillator/RF5110G	437.505 MHz	amateur	1 W	MSP430F1611	AX.25	1200 baud FSK	6 MB	N/A	43+ months	monopole	DOA	April 2013
EULA/F17 30 May 2010	Hayato	36573	1U	Custom (beacon)	13.275 GHz	Earth exploration	100 mW	AX.25	AX.25	10 kbps/1 Mbps BPSK	0.4 KB ²	18 days	patch	DOA	April 2013	
	Wesend/SAP2	36574	1U	TXE430-301A	437.485 MHz	amateur	150 mW	HS/3052F ⁸	AX.25	9600 baud FSK	0.4 KB	0 days	dipole	DOA	April 2013	
	Nagai-Star	36575	1U	Data	437.305 MHz	amateur	150 mW	AX.25	AX.25	1200 baud FSK	N/A	1 month	dipole	DOA	April 2013	
	Beacon Radio	36575	1U	Beacon Radio	437.305 MHz	amateur	100 mW	AX.25	AX.25	50 WPM	N/A	0	0 days	dipole	DOA	April 2013
	Tsai-1	36790	1U	Alinco IJ-C6	437.365 MHz	amateur	500 mW	MSP430F160	AX.25	1200 baud AFSK	N/A	33+ months	monopole	DOA	April 2013	
NLS-6/ PSV-C15 12 July 2010	StuSat	36796	1U	CC1020	437.505 MHz	amateur	500 mW	MSP430F160	AX.25	15.4 kbaud	N/A	0	0 days	monopole	DOA	April 2013
	MAXI142 (beacon)	36796	1U	MAXI142 (beacon)	437.860 MHz	amateur	10 mW	UC3A0612 ⁹	AX.25	4800 baud FSK	0.4 KB ²	5 days	monopole	DOA	April 2013	
SUTS/SB 10 Nov 2010	RAX-1	37223	3U	Lithium-1	437.505 MHz	amateur	750 mW	Integrated	AX.25	9600 baud GMSK	4.8 MB	2 months	turnstile	DOA	April 2013	
	O/OREOS	37224	3U+ ⁵	Microhard MHX-2400	2.4 GHz	experimental	1 W	Integrated	Proprietary	AX.25	Variable	8 MB	29+ months	patch	DOA	April 2013
	Nanosat-D2	37361	3U+ ⁵	Microhard MHX-2400	2.4 GHz	experimental	500 mW	Integrated	Proprietary	AX.25	Variable	N/A	5+ days ¹⁰	patch	DOA	April 2013
ELASCO/02 8 Dec 2010	Poseidon (4)	37251+ ¹¹	1.5U	TI CC	450 MHz	government	1 W	Integrated	Proprietary	9600 baud GMSK	1 month	dipole	DOA	April 2013		
	MQX (2)	37249+ ¹¹	3U	Pericle	UHF	government	1 W	Integrated	Proprietary	9600 baud GMSK	1 month	quadralair helix	DOA	April 2013		
	SMD-CONE	37249	3U	Pericle	UHF	government	1 W	Integrated	Proprietary	9600 baud GMSK	1 month	quadralair helix	DOA	April 2013		
PSV/C18 12 Oct 2011	Juggat	37839	3U	CC1070/RF5110G	437.505 MHz	amateur	1 W	Integrated	Proprietary	AX.25	2400 baud FSK	N/A	18+ months	monopole	DOA	April 2013
	MAXI142 (beacon)	37839	3U	MAXI142 (beacon)	437.505 MHz	amateur	10 mW	AX.25	AX.25	20 WPM	N/A	0	0 days	monopole	DOA	April 2013
ELASCO-1 NPP 26 Oct 2011	AuroSat-1	37854	1U	Melex (FH201)	437.475 MHz	amateur	800 mW	ATmega281 ¹²	CW	20 WPM	0.4 KB	22+ months	dipole	DOA	April 2013	
	DIC (2)	37851	1.5U	LI	465.3 MHz	monopole	1 W	Integrated	Proprietary	2.6 Mbps QPSK	8.4 GB	30 months	dipole	DOA	April 2013	
	HRBE	37855	1U	CC1000	437.505 MHz	amateur	850 mW	AX.25	AX.25	1200 baud FSK	N/A	22+ months	monopole	DOA	April 2013	
	M-Cubed	37855	1U	Lithium-1	437.485 MHz	amateur	1 W	Integrated	AX.25	1200 baud FSK	0.4 KB ²	22+ months	monopole	DOA	April 2013	
	RAX-2	37853	3U	Lithium-1	437.345 MHz	amateur	1 W	Integrated	AX.25	9600 baud GMSK	242 MB	18+ months	turnstile	DOA	April 2013	
New VVOI 13 Feb 2012	Xatoboo	38082	1U	GomSpace U89C	437.365 MHz	amateur	500 mW	Integrated	AX.25/CW	1200 baud MSK/20 WPM	0.4 KB ²	18+ months	turnstile	DOA	April 2013	
	ROBUSTA	38081	1U	MC121 (Max2005)	437.325 MHz	amateur	800 mW	PIC18F1550 ¹³	AX.25	1200 baud AFSK	0.4 KB ²	2.4 days	dipole	DOA	April 2013	
	e-Stri	38079	1U	BHX2-437-5	437.445 MHz	amateur	500 mW	PIC16	AX.25	1200 baud AFSK	0.4 KB ²	3 days	dipole	DOA	April 2013	
	Gollat	38085	1U	Alinco DJ-C7	437.485 MHz	amateur	500 mW	FX614/MSK140	AX.25/CW	1200 baud AFSK/20 WPM	0.4 KB ²	1 week	monopole	DOA	April 2013	
	PW-Sat	38083	1U	Microhard MHX-2400	2.4 GHz	experimental	1 W	Integrated	Proprietary	AX.25	Variable	0.4 KB ²	10+ months	patch	DOA	April 2013
ELASCO-2 NPP 25 Sep 2012	Misat-1	38764	3U	S1432	145.900 MHz	amateur	100/400 mW	dsPIC33F ¹⁴	Custom/CW	GFSK/120 CFM	305 MB	18+ months	monopole	DOA	April 2013	
	UnCubeSat-CG	38764	1U	AstroDev Custom	437.305 MHz	amateur	500 mW	Integrated	AX.25/CW	9600 baud GFSK	0.4 KB ²	2 days	dipole	DOA	April 2013	
	SMD-CONE (2)	38766+ ¹¹	3U	Pericle	UHF	government	1 W	Integrated	Proprietary	9600 baud GMSK	1 month	quadralair helix	DOA	April 2013		
	AeroCube-4 (3)	38767+ ¹¹	3U	FreeWave MM2	915 MHz	experimental	2 W	Integrated	Proprietary	38.4 kbaud	500 kbps	8+ months	patch	DOA	April 2013	
	Aeneas	38760	3U	MHX-425	437.000 MHz	amateur	1 W	Integrated	Proprietary	AX.25	Variable	N/A	8+ months	monopole	DOA	April 2013
ITV-6/BSB 4 Oct 2012	CSSWE	38761	3U	Lithium-1	437.345 MHz	experimental	1 W	Integrated	AX.25	9600 baud FSK	60 MB	12+ months	monopole	DOA	April 2013	
	CC1000/RF2117	38763	1U	CC1000/RF2117	437.405 MHz	amateur	500 mW	PIC18LF6720	AX.25	1200 baud FSK	500 KB	4 months	dipole	DOA	April 2013	
	CXBN	38762	2U	Lithium-1	437.525 MHz	amateur	1 W	Integrated	AX.25	9600 baud GFSK	N/A	8+ months	turnstile	DOA	April 2013	
	CINEMA	38764	3U	Emisler	2900 MHz	space research	1 W	FPGA	Proprietary	1 Mbps FSK	N/A	8+ months	patch	DOA	April 2013	
	Re	38765	3U	Helium-100	915 MHz	government	1 W	Integrated	AX.25	57.6 kbps FSK	0.4 KB	0 days	dipole	DOA	April 2013	
PSV/C20 25 Feb 2013	FTTSat-1	38853	1U	High-speed Custom	5.84 GHz	amateur	2 W	PIC16F886	AX.25	115.2 kbps FSK	N/A	9 months	patch	DOA	April 2013	
	CC1000/RF2117	38854	1U	CC1000/RF2117	437.445 MHz	amateur	1 W	PIC16F1519	AX.25	1200 baud AFSK	N/A	0	0 days	monopole	DOA	April 2013
	TechSat10 ¹⁵	38854	1U	Stensat (beacon) ⁷	437.465 MHz	amateur	1 W	PIC16F1519	AX.25	1200 baud AFSK	N/A	7 months	dipole	DOA	April 2013	
	F-1	38855	1U	VX-3R	145.980 MHz	amateur	1 W	PIC18F	AX.25	1200 baud AFSK	0.4 KB</					

Version 9; Page 2 of 2. This chart shows the 256 total CubeSats deployed in orbit so far, for a total of 499 Units. March 10, 2015.
 Bryan Klofas. bklofas@gmail.com. Green are University CubeSats; Red are Commercial or Private; Blue are US Government.

Launch	Satellite	Object	Size	Radio	Downlink	Satellite Service	Power	TNC	Protocol	Data Rate/Modulation	Downloaded	Lifetime	Antenna	Status	Updated	
HTV-1/ISS 19 Nov 2013	ArduSat-1	30413	1U	NanoCom U482C	437,000 MHz	Experimental	1 W	NanoMind AT12D ¹	CSP	9600 baud FM-MSK/20 WPM CW		2+ months	turnstile	Alive/Deorbited	Mar 2015	
	ArduSat-X	30414	1U	NanoCom U482C	437,000 MHz	Experimental	1 W	NanoMind AT12D ¹	OSP	9600 baud FM-MSK/20 WPM CW		2+ months	turnstile	Deorbited	Mar 2015	
	Pico Dragon	30412	1U	(data) (beacon)	437,365 MHz 437,250 MHz	Amateur	800 mW 100 mW		AX 25 CW	1200 baud AFSK		1 month	monopole	Alive/Deorbited	Mar 2015	
ORS-3/ElanNa-2 20 Nov 2013	TechESat-3P	30415	3U	Stensa (beacon) Iridium QP602	437,465 MHz 1616 MHz	Experimental	1 W 1 W		AX 25 Proprietary	1200 baud AFSK	N/A	1 month	monopole patch	Deorbited	Mar 2015	
	COPPER	30395	1U	AstroDev He-100	437,290 MHz	Experimental	1 W	Integrated	AX 25	9600 baud FSK			dipole	DOA	Jan 2014	
	Vermont Lunar Cube	30407	1U	Bellium-100	437,305 MHz	Experimental	1.5 W	Integrated	AX 25	9600 baud FSK		1+ month	Crossed Dipoles	Alive	Jan 2014	
	SwampSat	30402	1U		437,385 MHz	Experimental	1 W	Integrated	AX 25	9600 baud FSK					Jan 2014	
	CAPE-2	30382	1U	CC1101	437,325 MHz	Experimental	1 W	Integrated	TI	9600 baud FSK		1+ month	turnstile	Alive	July 2014	
	DragonSat-1	30388	1U	Microhard MHX-2120	437,220 MHz	Experimental	1 W	Integrated	Proprietary	FSK	57.5 kbps	N/A		patch monopole	DOA	Jan 2014
	PhoSat-v2.4	30381	1U	StenSat	437,425 MHz	Experimental	1 W	Integrated	AX 25	1200 baud AFSK				patch monopole	Alive	Jan 2014
	Traillblazer	30400	1U	AstroDev He-100	437,425 MHz	Experimental	1 W	Integrated	AX 25	9600 baud FSK				turnstile	DOA	Jan 2014
	KySat-2	30384	1U	AstroDev Li-1	437,405 MHz	Experimental	1.5 W	Integrated	AX 25	9600 baud FSK			1+ month	turnstile	Alive	Jan 2014
	ChargerSat-1	30405	1U	CC1101	437,405 MHz	Experimental	1 W	Integrated	AX 25	9600 baud FSK				dipole	DOA	Dec 2013
	NPS-SCAT	30389	1U	Microhard MHX-2400	2.4 GHz	Experimental	1 W	Integrated	Proprietary		1200 baud FSK	N/A		patch dipole	Alive	Jan 2014
	Black Knight 1	30398	1U	MHX-125	437,345 MHz	Experimental	1 W	Integrated	Proprietary							
	ORSES	30386	3U			Government										
	ORSTECH (2)	30387+	3U			Government										
	Prismless (8)	30406+	1.0U			Government										
	SENSE (2)	30388+	3U	Insight SCR-100	2.2 GHz	Government								patch	Alive	Dec 2013
	FinFly	30404	3U	L-3 Cadet	425 MHz	Government	1 W	Integrated	Proprietary		QPSK			dipole	Alive	Jan 2014
	Horus	30397	3U	AstroDev	915 MHz	Government								patch	DOA	Dec 2013
	IS1Launch03/DeepP 21 Nov 2013	FUNcube-1	30414	1U	AMSAT	145,936 MHz	Amateur	300 mW	N/A		1200 baud BPSK	70 MB+	2+ months	dipole	Active	Jan 2014
		ZACube-1	30417	1U	Custom	437,345 MHz	Amateur			AX 25	1200 baud AFSK/9600 baud FSK		2+ months	dipole	Alive	Jan 2014
HiNcube		30415	1U	Custom	437,305 MHz	Amateur			AX 25	9600 baud GMSK		2+ months	dipole	DOA	Jan 2014	
First-MOVE		30408	1U	ISIS TRXUV	145,970 MHz	Amateur	230 mW	Integrated	AX 25	1200 baud BPSK		2+ months	dipole	Alive	Jan 2014	
UWE-3		30446	1U	ISIS TRXUV	437,385 MHz	Amateur	230 mW	Integrated	AX 25	9600 baud FSK		2+ months	dipole	Alive	Jan 2014	
Valeo-PH		30428	3U	ISIS ²	145,980 MHz	Amateur	230 mW	Integrated	AX 25	1200 baud BPSK		2+ months	dipole	Alive	Jan 2014	
NEE-02 KRYSAOR		30411	1U		980 MHz	Amateur			AX 25	1200 baud BPSK		2+ months	dipole	Alive	Jan 2014	
CubeBug-2		30440	2U	AstroDev Li-1	437,445 MHz	Amateur	1 W	Integrated	AX 25	1200/9600 baud AFSK/GFSK		2+ months	turnstile	Alive	Jan 2014	
KHUSAT (2)		30424+	3U	Embleer	2.2 GHz	Space Research	1 W	FFGA	Proprietary		1 Mbps FSK		2+ months	patch	Alive	Jan 2014
TRITON-1		30427	3U	ISIS TRXUV	145,815 MHz	Amateur	230 mW	Integrated	AX 25	9600 baud BPSK		2+ months	dipole	Alive	Jan 2014	
IMb-nsat		30428	3U	ISIS TRXUV	145,900 MHz	Amateur	230 mW	Integrated	AX 25	9600 baud BPSK		2+ months	dipole	Alive	Feb 2014	
OPTOS		30420	3U	ISIS Custom	2.405 GHz	Amateur	125 mW				20-500 kbps MSK		2+ months	patch	Alive	Jan 2014
Dove-3		30429	3U+	D3-400-T D3-8200-T	401.3 MHz 8.1 GHz	Experimental	1.6 W 3 W			IP over DVB-S2				monopole patch	Active	Jan 2014
PUPC-SAT-1 ³		30412	1U	Telemetry Beacon	145,840 MHz 437,200 MHz	Amateur	1.5 W 10 mW			AX 25 CW	1200 baud AFSK 12 WPM	N/A		dipole dipole	Alive Jan 2014	
ICUBE-1 ⁴		30413	1U	ISIS TRXUV	145,917 MHz	Amateur	230 mW	Integrated	AX 25	1200 baud BPSK			2+ months	turnstile	Active	Jan 2014
HiNcube-D ⁵		30433	1U	NanoCom U482C	437,325 MHz	Amateur	500 mW			CSP/CW	1200 baud MSK		2+ months	turnstile	Active	Jan 2014
Dove-4 ⁶		30434	3U+	D3-400-T D3-8200-T	401.3 MHz 8.1 GHz	Experimental	1.6 W 3 W			IP over DVB-S2				monopole patch	Active	Jan 2014
GomX-1		30430	2U	NanoCom U482C	437,250 MHz	Amateur	3 W			CSP	4800 baud MSK		2+ months	turnstile	Alive	Mar 2015
NR0-2/9/ElanNa-2 3 Dec 2013		IFEX	30471	1U		437,270 MHz	Experimental	1 W		AX 25	9600 baud FSK		2+ months	dipole	Active	Jan 2014
		MicroSat-2	30469	1U	AstroDev Li-1	437,485 MHz	Experimental	1 W		AX 25	9600 baud GMSK		2+ months	monopole	Active	Jan 2014
	CUNYSAT-1	30470	1U		437,505 MHz	Experimental	1.2 W									
	FIREBRD (2)	30463+	1.0U		437,405 MHz	Amateur	1 W					27+ MB	2+ months	dipole	Active/DOA	Jan 2014
	Allee	30467	3U			Government										
	AeroCube-5 (2)	30465+	1.0U	FreeSpace FGRM CC1101	915 MHz 915 MHz	Experimental	1.3 W	Integrated	Proprietary		500 kbps FSK			patch		
	SMDC-ONE (2)	30472+	3U			Government										
	TecSat-6	30473	3U			Government										
	SNAP	30488	3U			Government										
	Oris-1/ISS 28 Feb 2014	Flock-1 (28)	30513+	3U+	D3-400-T D3-8200-T	401.3 MHz 8.1 GHz	Earth exploration Earth exploration	1.6 W 3 W			IP over DVB-S2				monopole patch	Active
UAPSAT		30568	1U		437,385 MHz	Amateur			AX 25					turnstile		
ArduSat-2		30571	2U	AstroDev Li-1 AstroDev Beeryllium	400 MHz 2.4 GHz	Experimental	2 W 2.3 W							turnstile	DOA	Apr 2014
SkyCube		30567	1U	AstroDev C-2	915 MHz	Government			AX 25		57.6 kbps			dipole	Dead	Apr 2014
LiSat-1		30570	1U		145,850 MHz	Amateur	500 mW		AX 25					dipole	Alive	Apr 2014
IRUA/GPM 27 Feb 2014	INVADER (C-077)	30578	1U	Custom	437,200 MHz	Amateur	800 mW		AX 25	1200 baud AFSK	N/A	2+ months	dipole	Active	Apr 2014	
	ISSAT-2	30573	1U	S-band Ku-band	437,325 MHz	Amateur	100 mW							dipole	Active	Apr 2014
	OPUSAT	30575	1U	Custom	437,150 MHz	Amateur			CW/AX 25	1200 baud AFSK/9600 baud GMSK		2+ months	dipole	Alive	Apr 2014	
CRS-1/ ElanNa-5 18 Apr 2014	SponSat	30681	3U	StenSat	437,100 MHz	Experimental ¹	1 W	Integrated	AX 25	1200 baud AFSK			monopole	Deorbited	Aug 2014	
	TSAT	30682	2U	GlobalStar simplex STX2	1.6 GHz	Experimental ¹	200 mW	PIC18F2620	Proprietary	30bytes/sec			Patch	Deorbited	Aug 2014	
	PhoSat-v2.5	30684	1U		437,425 MHz	Experimental	1 W		AX 25	1200 baud AFSK			monopole	Deorbited	Aug 2014	
	ALL-STAR	30683	3U	Custom SDR	2401.7 MHz	Experimental	1 W		AX 25	1200 baud AFSK			monopole	Deorbited	Aug 2014	
	KiCSat	30685	3U		437,565 MHz	Amateur	1 W		AX 25	1200 baud AFSK			monopole	Deorbited	Aug 2014	
IS1Launch07/DeepP 12 June 2014	AnteSat ⁷	40034	2U	Custom (beacon) Custom (downlink)	437,280 MHz 437,375 MHz 2403 MHz	Amateur	200 mW 1W		CW AX 25	1200 baud FSK/AFSK 500 kbps GFSK	N/A		1.5 months monopole	Alive	Aug 2014	
	AeroCube-0(2)	40015+	0.5U	CC1101	915 MHz	experimental	1.3 W	Integrated	Proprietary			1.5 months	patch	Alive	Aug 2014	
	LEMUR-1 ⁸	40014	3U	AstroDev Lithium	402 MHz	Experimental	2 W	Integrated								
	UgrSat ⁹	40013	3U			Experimental										
	Flock-1 (11)	40027+	3U+	D3-400-T D3-8200-T	401.3 MHz 8.1 GHz	Earth exploration Earth exploration	1.6 W 3 W			IP over DVB-S2	40 Mbps		1.5 months	monopole patch/helix	Active	Aug 2014
	Persus (2)	40039+	6U	AstroDev Lithium Custom transmitter	401 MHz 26.8 GHz	Experimental	1 W 500 mW	Integrated	AX 25 DVB-S2	40 Mbps		1.5 months	Monopole Horn	Alive	Aug 2014	
	QB50P1	40025	2U	ISIS TRXUV FUNcube-3	145,815 MHz 145,950 MHz	Amateur	200 mW 400 mW	Integrated	CW/AX 25	15 WPM/1200 baud BPSK Linear transponder		1.5 months	dipole dipole	Alive	Aug 2014	
	QB50P2	40032	2U	ISIS TRXUV FUNcube-3	145,880 MHz 145,940 MHz	Amateur	200 mW 400 mW	Integrated	CW/AX 25	15 WPM/1200 baud BPSK 9600 lps FSK		1.5 months	dipole dipole	Alive	Aug 2014	
	DTUSat-2	40030	1U	AMSAT-Francephone	2.401 GHz	Amateur	220 mW		CW/FSK							
	POMTIAN-1	40012	1U	Custom	437,675 MHz	Amateur	600 mW		CW/AX 25	9600 baud FSK			1.5 months	dipole	Alive	Aug 2014
	Duchafin-1	40021	1U	ISIS TRXUV	145,980 MHz	Amateur	200 mW	Integrated	CW/AX 25	15 WPM/1200 baud BPSK			1.5 months	dipole	Alive	Aug 2014
	NanosatCBr-1	40024	2U	ISIS TRXUV	145,825 MHz	Amateur	200 mW	Integrated	AX 25	1200 baud AFSK			1.5 months	dipole	Alive	Aug 2014
	PACE	40022	2U	ISIS TRXUV	437,485 MHz	Amateur	200 mW	Integrated	CW/AX 25	15 WPM/1200 baud BPSK			1.5 months	dipole	Alive	Aug 2014
	POPSAT-HP-1	40028	3U		437,405 MHz	Amateur			CXSDS					dipole	Alive	Aug 2014
	PSUV-CB- 6/30/2014	VELOX-1	40057	3U	ISIS TRXUV	145,980 MHz	Amateur	200 mW	Integrated	AX 25	1200 baud BPSK				dipole	Alive
UKube-1		40074	3U	ISIS TRXUV FUNcube-2 CS-CP/UT-STX-01	145,840 MHz 145,915 MHz 2.401 GHz	Amateur	200 mW 300 mW 1 W	Integrated	AX 25	1200 baud BPSK 1200 baud BPSK 2 Mbps QPSK		1 month	dipole dipole patch	Alive	Aug 2014	
ISS - 8/18/2014	Classiq-1 ¹⁰	40117	1U		437,625 MHz	Experimental	1 W		AX 25	1200 baud AFSK		0 days	monopole	DOA	Mar 2015	
	Evocube	40380	1U	Assen AX5012	437,270 MHz	Amateur	1 W	ATD1 ¹¹	AX 25	9600 baud FSK		2+ months	monopole	Alive	Mar 2015	
SMAP/ElanNa-10 31 Jan 2015	GRIFEX	40379	3U	AstroDev Lithium	437,485 MHz	Amateur	1W	Integrated	AX 25	9600/15000 baud GFSK		2+ months	monopole	Alive	Mar 2015	
	FIREBR															

Appendix B: AX.25 Layer 3 Protocol Definitions

HEX	M S B	L S B	Translation
**	yy01yyyy		AX.25 layer 3 implemented.
**	yy10yyyy		AX.25 layer 3 implemented.
0x01	00000001		ISO 8208/CCITT X.25 PLP
0x06	00000110		Compressed TCP/IP packet. Van Jacobson (RFC 1144)
0x07	00000111		Uncompressed TCP/IP packet. Van Jacobson (RFC 1144)
0x08	00001000		Segmentation fragment
0xC3	11000011		TEXNET datagram protocol
0xC4	11000100		Link Quality Protocol
0xCA	11001010		Appletalk
0xCB	11001011		Appletalk ARP
0xCC	11001100		ARPA Internet Protocol
0xCD	11001101		ARPA Address resolution
0xCE	11001110		FlexNet
0xCF	11001111		NET/ROM
0xF0	11110000		No layer 3 protocol implemented.
0xFF	11111111		Escape character. Next octet contains more Level 3 protocol information.
Escape character. Next octet contains more Level 3 protocol information.	00001000		

Table B.1: AX.25 Layer 3 Protocol Definitions

Appendix C: CCSDS Telecommand Additional Information

C.1 Sending Node Architecture

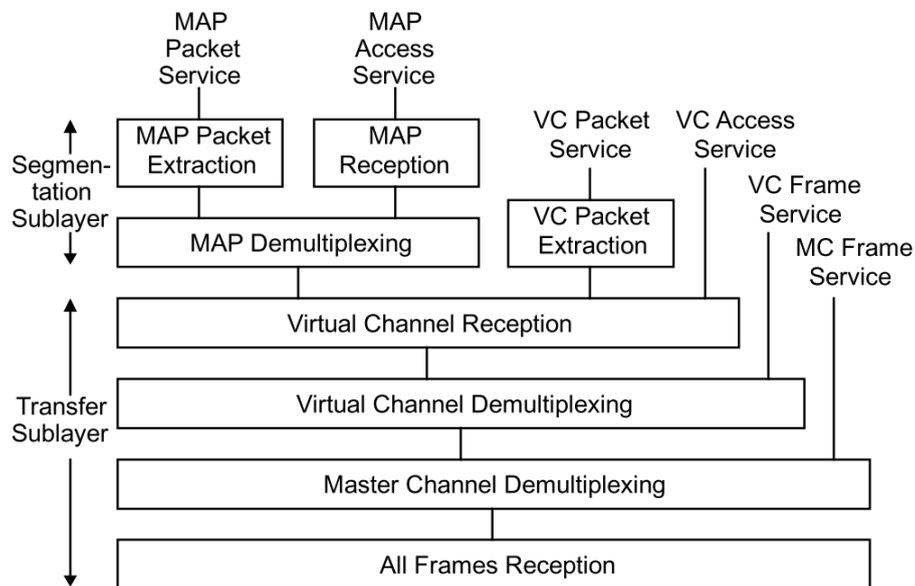


Figure C.1: TC Sending Node. Adapted from [4]

Appendix D: GF(256) Elements

GF(256) elements, $\phi(x) = x^8 + x^4 + x^3 + x^2 + 1$. The field elements are generated by first having setting 0 element then, the primitive α element. For each stage from α^0 , multiply the current stage α value with α and substituting x^8 with $x^4 + x^3 + x^2 + 1$ from equations 4.3 and 4.4.

	Exp	Bin	Dec		Exp	Bin	Dec
0	0	00000000	0				
α^0	1	00000001	1	α^{128}	$\alpha^7 + \alpha^2 + 1$	10000101	133
α^1	α^1	00000010	2	α^{129}	$\alpha^4 + \alpha^2 + \alpha^1 + 1$	00010111	23
α^2	α^2	00000100	4	α^{130}	$\alpha^5 + \alpha^3 + \alpha^2 + \alpha^1$	00101110	46
α^3	α^3	00001000	8	α^{131}	$\alpha^6 + \alpha^4 + \alpha^3 + \alpha^2$	01011100	92
α^4	α^4	00010000	16	α^{132}	$\alpha^7 + \alpha^5 + \alpha^4 + \alpha^3$	10111000	184
α^5	α^5	00100000	32	α^{133}	$\alpha^6 + \alpha^5 + \alpha^3 + \alpha^2 + 1$	01101101	109
α^6	α^6	01000000	64	α^{134}	$\alpha^7 + \alpha^6 + \alpha^4 + \alpha^3 + \alpha^1$	11011010	218
α^7	α^7	10000000	128	α^{135}	$\alpha^7 + \alpha^5 + \alpha^3 + 1$	10101001	169
α^8	$\alpha^4 + \alpha^3 + \alpha^2 + 1$	00011101	29	α^{136}	$\alpha^6 + \alpha^3 + \alpha^2 + \alpha^1 + 1$	01001111	79
α^9	$\alpha^5 + \alpha^4 + \alpha^3 + \alpha^1$	00111010	58	α^{137}	$\alpha^7 + \alpha^4 + \alpha^3 + \alpha^2 + \alpha^1$	10011110	158
α^{10}	$\alpha^6 + \alpha^5 + \alpha^4 + \alpha^2$	01110100	116	α^{138}	$\alpha^5 + 1$	00100001	33
α^{11}	$\alpha^7 + \alpha^6 + \alpha^5 + \alpha^3$	11101000	232	α^{139}	$\alpha^6 + \alpha^1$	01000010	66
α^{12}	$\alpha^7 + \alpha^6 + \alpha^3 + \alpha^2 + 1$	11001101	205	α^{140}	$\alpha^7 + \alpha^2$	10000100	132
α^{13}	$\alpha^7 + \alpha^2 + \alpha^1 + 1$	10000111	135	α^{141}	$\alpha^4 + \alpha^2 + 1$	00010101	21
α^{14}	$\alpha^4 + \alpha^1 + 1$	00010011	19	α^{142}	$\alpha^5 + \alpha^3 + \alpha^1$	00101010	42
α^{15}	$\alpha^5 + \alpha^2 + \alpha^1$	00100110	38	α^{143}	$\alpha^6 + \alpha^4 + \alpha^2$	01010100	84
α^{16}	$\alpha^6 + \alpha^3 + \alpha^2$	01001100	76	α^{144}	$\alpha^7 + \alpha^5 + \alpha^3$	10101000	168
α^{17}	$\alpha^7 + \alpha^4 + \alpha^3$	10011000	152	α^{145}	$\alpha^6 + \alpha^3 + \alpha^2 + 1$	01001101	77
α^{18}	$\alpha^5 + \alpha^3 + \alpha^2 + 1$	00101101	45	α^{146}	$\alpha^7 + \alpha^4 + \alpha^3 + \alpha^1$	10011010	154
α^{19}	$\alpha^6 + \alpha^4 + \alpha^3 + \alpha^1$	01011010	90	α^{147}	$\alpha^5 + \alpha^3 + 1$	00101001	41
α^{20}	$\alpha^7 + \alpha^5 + \alpha^4 + \alpha^2$	10110100	180	α^{148}	$\alpha^6 + \alpha^4 + \alpha^1$	01010010	82
α^{21}	$\alpha^6 + \alpha^5 + \alpha^4 + \alpha^2 + 1$	01110101	117	α^{149}	$\alpha^7 + \alpha^5 + \alpha^2$	10100100	164
α^{22}	$\alpha^7 + \alpha^6 + \alpha^5 + \alpha^3 + \alpha^1$	11101010	234	α^{150}	$\alpha^6 + \alpha^4 + \alpha^2 + 1$	01010101	85
α^{23}	$\alpha^7 + \alpha^6 + \alpha^3 + 1$	11001001	201	α^{151}	$\alpha^7 + \alpha^5 + \alpha^3 + \alpha^1$	10101010	170
α^{24}	$\alpha^7 + \alpha^3 + \alpha^2 + \alpha^1 + 1$	10001111	143	α^{152}	$\alpha^6 + \alpha^3 + 1$	01001001	73
α^{25}	$\alpha^1 + 1$	00000011	3	α^{153}	$\alpha^7 + \alpha^4 + \alpha^1$	10010010	146
α^{26}	$\alpha^2 + \alpha^1$	00000110	6	α^{154}	$\alpha^5 + \alpha^4 + \alpha^3 + 1$	00111001	57
α^{27}	$\alpha^3 + \alpha^2$	00001100	12	α^{155}	$\alpha^6 + \alpha^5 + \alpha^4 + \alpha^1$	01110010	114
α^{28}	$\alpha^4 + \alpha^3$	00011000	24	α^{156}	$\alpha^7 + \alpha^6 + \alpha^5 + \alpha^2$	11100100	228
α^{29}	$\alpha^5 + \alpha^4$	00110000	48	α^{157}	$\alpha^7 + \alpha^6 + \alpha^4 + \alpha^2 + 1$	11010101	213
α^{30}	$\alpha^6 + \alpha^5$	01100000	96	α^{158}	$\alpha^7 + \alpha^5 + \alpha^4 + \alpha^2 + \alpha^1 + 1$	10110111	183
α^{31}	$\alpha^7 + \alpha^6$	11000000	192	α^{159}	$\alpha^6 + \alpha^5 + \alpha^4 + \alpha^1 + 1$	01110011	115
α^{32}	$\alpha^7 + \alpha^4 + \alpha^3 + \alpha^2 + 1$	10011101	157	α^{160}	$\alpha^7 + \alpha^6 + \alpha^5 + \alpha^2 + \alpha^1$	11100110	230
α^{33}	$\alpha^5 + \alpha^2 + \alpha^1 + 1$	00100111	39	α^{161}	$\alpha^7 + \alpha^6 + \alpha^4 + 1$	11010001	209
α^{34}	$\alpha^6 + \alpha^3 + \alpha^2 + \alpha^1$	01001110	78	α^{162}	$\alpha^7 + \alpha^5 + \alpha^4 + \alpha^3 + \alpha^2 + \alpha^1 + 1$	10111111	191
α^{35}	$\alpha^7 + \alpha^4 + \alpha^3 + \alpha^2$	10011100	156	α^{163}	$\alpha^6 + \alpha^5 + \alpha^1 + 1$	01100011	99
α^{36}	$\alpha^5 + \alpha^2 + 1$	00100101	37	α^{164}	$\alpha^7 + \alpha^6 + \alpha^2 + \alpha^1$	11000110	198
α^{37}	$\alpha^6 + \alpha^3 + \alpha^1$	01001010	74	α^{165}	$\alpha^7 + \alpha^4 + 1$	10010001	145
α^{38}	$\alpha^7 + \alpha^4 + \alpha^2$	10010100	148	α^{166}	$\alpha^5 + \alpha^4 + \alpha^3 + \alpha^2 + \alpha^1 + 1$	00111111	63
α^{39}	$\alpha^5 + \alpha^4 + \alpha^2 + 1$	00110101	53	α^{167}	$\alpha^6 + \alpha^5 + \alpha^4 + \alpha^3 + \alpha^2 + \alpha^1$	01111110	126
α^{40}	$\alpha^6 + \alpha^5 + \alpha^3 + \alpha^1$	01101010	106	α^{168}	$\alpha^7 + \alpha^6 + \alpha^5 + \alpha^4 + \alpha^3 + \alpha^2$	11111100	252
α^{41}	$\alpha^7 + \alpha^6 + \alpha^4 + \alpha^2$	11010100	212	α^{169}	$\alpha^7 + \alpha^6 + \alpha^5 + \alpha^2 + 1$	11100101	229
α^{42}	$\alpha^7 + \alpha^5 + \alpha^4 + \alpha^2 + 1$	10110101	181	α^{170}	$\alpha^7 + \alpha^6 + \alpha^4 + \alpha^2 + \alpha^1 + 1$	11010111	215
α^{43}	$\alpha^6 + \alpha^5 + \alpha^4 + \alpha^2 + \alpha^1 + 1$	01110111	119	α^{171}	$\alpha^7 + \alpha^5 + \alpha^4 + \alpha^1 + 1$	10110011	179

α^{44}	$\alpha^7 + \alpha^6 + \alpha^5 + \alpha^3 + \alpha^2 + \alpha^1$	11101110	238	α^{172}	$\alpha^6 + \alpha^5 + \alpha^4 + \alpha^3 + \alpha^1 + 1$	01111011	123
α^{45}	$\alpha^7 + \alpha^6 + 1$	11000001	193	α^{173}	$\alpha^7 + \alpha^6 + \alpha^5 + \alpha^4 + \alpha^2 + \alpha^1$	11110110	246
α^{46}	$\alpha^7 + \alpha^4 + \alpha^3 + \alpha^2 + \alpha^1 + 1$	10011111	159	α^{174}	$\alpha^7 + \alpha^6 + \alpha^5 + \alpha^4 + 1$	11110001	241
α^{47}	$\alpha^5 + \alpha^1 + 1$	00100011	35	α^{175}	$\alpha^7 + \alpha^6 + \alpha^5 + \alpha^4 + \alpha^3 + \alpha^2 + \alpha^1 + 1$	11111111	255
α^{48}	$\alpha^6 + \alpha^2 + \alpha^1$	01000110	70	α^{176}	$\alpha^7 + \alpha^6 + \alpha^5 + \alpha^1 + 1$	11100011	227
α^{49}	$\alpha^7 + \alpha^3 + \alpha^2$	10001100	140	α^{177}	$\alpha^7 + \alpha^6 + \alpha^4 + \alpha^3 + \alpha^1 + 1$	11011011	219
α^{50}	$\alpha^2 + 1$	00000101	5	α^{178}	$\alpha^7 + \alpha^5 + \alpha^3 + \alpha^1 + 1$	10101011	171
α^{51}	$\alpha^3 + \alpha^1$	00001010	10	α^{179}	$\alpha^6 + \alpha^3 + \alpha^1 + 1$	01001011	75
α^{52}	$\alpha^4 + \alpha^2$	00010100	20	α^{180}	$\alpha^7 + \alpha^4 + \alpha^2 + \alpha^1$	10010110	150
α^{53}	$\alpha^5 + \alpha^3$	00101000	40	α^{181}	$\alpha^5 + \alpha^4 + 1$	00110001	49
α^{54}	$\alpha^6 + \alpha^4$	01010000	80	α^{182}	$\alpha^6 + \alpha^5 + \alpha^1$	01100010	98
α^{55}	$\alpha^7 + \alpha^5$	10100000	160	α^{183}	$\alpha^7 + \alpha^6 + \alpha^2$	11000100	196
α^{56}	$\alpha^6 + \alpha^4 + \alpha^3 + \alpha^2 + 1$	01011101	93	α^{184}	$\alpha^7 + \alpha^4 + \alpha^2 + 1$	10010101	149
α^{57}	$\alpha^7 + \alpha^5 + \alpha^4 + \alpha^3 + \alpha^1$	10111010	186	α^{185}	$\alpha^5 + \alpha^4 + \alpha^2 + \alpha^1 + 1$	00110111	55
α^{58}	$\alpha^6 + \alpha^5 + \alpha^3 + 1$	01101001	105	α^{186}	$\alpha^6 + \alpha^5 + \alpha^3 + \alpha^2 + \alpha^1$	01101110	110
α^{59}	$\alpha^7 + \alpha^6 + \alpha^4 + \alpha^1$	11010010	210	α^{187}	$\alpha^7 + \alpha^6 + \alpha^4 + \alpha^3 + \alpha^2$	11011100	220
α^{60}	$\alpha^7 + \alpha^5 + \alpha^4 + \alpha^3 + 1$	10111001	185	α^{188}	$\alpha^7 + \alpha^5 + \alpha^2 + 1$	10100101	165
α^{61}	$\alpha^6 + \alpha^5 + \alpha^3 + \alpha^2 + \alpha^1 + 1$	01101111	111	α^{189}	$\alpha^6 + \alpha^4 + \alpha^2 + \alpha^1 + 1$	01010111	87
α^{62}	$\alpha^7 + \alpha^6 + \alpha^4 + \alpha^3 + \alpha^2 + \alpha^1$	11011110	222	α^{190}	$\alpha^7 + \alpha^5 + \alpha^3 + \alpha^2 + \alpha^1$	10101110	174
α^{63}	$\alpha^7 + \alpha^5 + 1$	10100001	161	α^{191}	$\alpha^6 + 1$	01000001	65
α^{64}	$\alpha^6 + \alpha^4 + \alpha^3 + \alpha^2 + \alpha^1 + 1$	01011111	95	α^{192}	$\alpha^7 + \alpha^1$	10000010	130
α^{65}	$\alpha^7 + \alpha^5 + \alpha^4 + \alpha^3 + \alpha^2 + \alpha^1$	10111110	190	α^{193}	$\alpha^4 + \alpha^3 + 1$	00011001	25
α^{66}	$\alpha^6 + \alpha^5 + 1$	01100001	97	α^{194}	$\alpha^5 + \alpha^4 + \alpha^1$	00110010	50
α^{67}	$\alpha^7 + \alpha^6 + \alpha^1$	11000010	194	α^{195}	$\alpha^6 + \alpha^5 + \alpha^2$	01100100	100
α^{68}	$\alpha^7 + \alpha^4 + \alpha^3 + 1$	10011001	153	α^{196}	$\alpha^7 + \alpha^6 + \alpha^3$	11001000	200
α^{69}	$\alpha^5 + \alpha^3 + \alpha^2 + \alpha^1 + 1$	00101111	47	α^{197}	$\alpha^7 + \alpha^3 + \alpha^2 + 1$	10001101	141
α^{70}	$\alpha^6 + \alpha^4 + \alpha^3 + \alpha^2 + \alpha^1$	01011110	94	α^{198}	$\alpha^2 + \alpha^1 + 1$	00000111	7
α^{71}	$\alpha^7 + \alpha^5 + \alpha^4 + \alpha^3 + \alpha^2$	10111100	188	α^{199}	$\alpha^3 + \alpha^2 + \alpha^1$	00001110	14
α^{72}	$\alpha^6 + \alpha^5 + \alpha^2 + 1$	01100101	101	α^{200}	$\alpha^4 + \alpha^3 + \alpha^2$	00011100	28
α^{73}	$\alpha^7 + \alpha^6 + \alpha^3 + \alpha^1$	11001010	202	α^{201}	$\alpha^5 + \alpha^4 + \alpha^3$	00111000	56
α^{74}	$\alpha^7 + \alpha^3 + 1$	10001001	137	α^{202}	$\alpha^6 + \alpha^5 + \alpha^4$	01110000	112
α^{75}	$\alpha^3 + \alpha^2 + \alpha^1 + 1$	00001111	15	α^{203}	$\alpha^7 + \alpha^6 + \alpha^5$	11100000	224
α^{76}	$\alpha^4 + \alpha^3 + \alpha^2 + \alpha^1$	00011110	30	α^{204}	$\alpha^7 + \alpha^6 + \alpha^4 + \alpha^3 + \alpha^2 + 1$	11011101	221
α^{77}	$\alpha^5 + \alpha^4 + \alpha^3 + \alpha^2$	00111100	60	α^{205}	$\alpha^7 + \alpha^5 + \alpha^2 + \alpha^1 + 1$	10100111	167
α^{78}	$\alpha^6 + \alpha^5 + \alpha^4 + \alpha^3$	01111000	120	α^{206}	$\alpha^6 + \alpha^4 + \alpha^1 + 1$	01010011	83
α^{79}	$\alpha^7 + \alpha^6 + \alpha^5 + \alpha^4$	11110000	240	α^{207}	$\alpha^7 + \alpha^5 + \alpha^2 + \alpha^1$	10100110	166
α^{80}	$\alpha^7 + \alpha^6 + \alpha^5 + \alpha^4 + \alpha^3 + \alpha^2 + 1$	11111101	253	α^{208}	$\alpha^6 + \alpha^4 + 1$	01010001	81
α^{81}	$\alpha^7 + \alpha^6 + \alpha^5 + \alpha^2 + \alpha^1 + 1$	11100111	231	α^{209}	$\alpha^7 + \alpha^5 + \alpha^1$	10100010	162
α^{82}	$\alpha^7 + \alpha^6 + \alpha^4 + \alpha^1 + 1$	11010011	211	α^{210}	$\alpha^6 + \alpha^4 + \alpha^3 + 1$	01011001	89
α^{83}	$\alpha^7 + \alpha^5 + \alpha^4 + \alpha^3 + \alpha^1 + 1$	10111011	187	α^{211}	$\alpha^7 + \alpha^5 + \alpha^4 + \alpha^1$	10110010	178
α^{84}	$\alpha^6 + \alpha^5 + \alpha^3 + \alpha^1 + 1$	01101011	107	α^{212}	$\alpha^6 + \alpha^5 + \alpha^4 + \alpha^3 + 1$	01111001	121
α^{85}	$\alpha^7 + \alpha^6 + \alpha^4 + \alpha^2 + \alpha^1$	11010110	214	α^{213}	$\alpha^7 + \alpha^6 + \alpha^5 + \alpha^4 + \alpha^1$	11110010	242
α^{86}	$\alpha^7 + \alpha^5 + \alpha^4 + 1$	10110001	177	α^{214}	$\alpha^7 + \alpha^6 + \alpha^5 + \alpha^4 + \alpha^3 + 1$	11111001	249
α^{87}	$\alpha^6 + \alpha^5 + \alpha^4 + \alpha^3 + \alpha^2 + \alpha^1 + 1$	01111111	127	α^{215}	$\alpha^7 + \alpha^6 + \alpha^5 + \alpha^3 + \alpha^2 + \alpha^1 + 1$	11101111	239
α^{88}	$\alpha^7 + \alpha^6 + \alpha^5 + \alpha^4 + \alpha^3 + \alpha^2 + \alpha^1$	11111110	254	α^{216}	$\alpha^7 + \alpha^6 + \alpha^1 + 1$	11000011	195
α^{89}	$\alpha^7 + \alpha^6 + \alpha^5 + 1$	11100001	225	α^{217}	$\alpha^7 + \alpha^4 + \alpha^3 + \alpha^1 + 1$	10011011	155
α^{90}	$\alpha^7 + \alpha^6 + \alpha^4 + \alpha^3 + \alpha^2 + \alpha^1 + 1$	11011111	223	α^{218}	$\alpha^5 + \alpha^3 + \alpha^1 + 1$	00101011	43
α^{91}	$\alpha^7 + \alpha^5 + \alpha^1 + 1$	10100011	163	α^{219}	$\alpha^6 + \alpha^4 + \alpha^2 + \alpha^1$	01010110	86
α^{92}	$\alpha^6 + \alpha^4 + \alpha^3 + \alpha^1 + 1$	01011011	91	α^{220}	$\alpha^7 + \alpha^5 + \alpha^3 + \alpha^2$	10101100	172
α^{93}	$\alpha^7 + \alpha^5 + \alpha^4 + \alpha^2 + \alpha^1$	10110110	182	α^{221}	$\alpha^6 + \alpha^2 + 1$	01000101	69
α^{94}	$\alpha^6 + \alpha^5 + \alpha^4 + 1$	01110001	113	α^{222}	$\alpha^7 + \alpha^3 + \alpha^1$	10001010	138
α^{95}	$\alpha^7 + \alpha^6 + \alpha^5 + \alpha^1$	11100010	226	α^{223}	$\alpha^3 + 1$	00001001	9
α^{96}	$\alpha^7 + \alpha^6 + \alpha^4 + \alpha^3 + 1$	11011001	217	α^{224}	$\alpha^4 + \alpha^1$	00010010	18
α^{97}	$\alpha^7 + \alpha^5 + \alpha^3 + \alpha^2 + \alpha^1 + 1$	10101111	175	α^{225}	$\alpha^5 + \alpha^2$	00100100	36
α^{98}	$\alpha^6 + \alpha^1 + 1$	01000011	67	α^{226}	$\alpha^6 + \alpha^3$	01001000	72
α^{99}	$\alpha^7 + \alpha^2 + \alpha^1$	10000110	134	α^{227}	$\alpha^7 + \alpha^4$	10010000	144
α^{100}	$\alpha^4 + 1$	00010001	17	α^{228}	$\alpha^5 + \alpha^4 + \alpha^3 + \alpha^2 + 1$	00111101	61

α^{101}	$\alpha^5 + \alpha^1$	00100010	34	α^{229}	$\alpha^6 + \alpha^5 + \alpha^4 + \alpha^3 + \alpha^1$	01111010	122
α^{102}	$\alpha^6 + \alpha^2$	01000100	68	α^{230}	$\alpha^7 + \alpha^6 + \alpha^5 + \alpha^4 + \alpha^2$	11110100	244
α^{103}	$\alpha^7 + \alpha^3$	10001000	136	α^{231}	$\alpha^7 + \alpha^6 + \alpha^5 + \alpha^4 + \alpha^2 + 1$	11110101	245
α^{104}	$\alpha^3 + \alpha^2 + 1$	00001101	13	α^{232}	$\alpha^7 + \alpha^6 + \alpha^5 + \alpha^4 + \alpha^2 + \alpha^1 + 1$	11110111	247
α^{105}	$\alpha^4 + \alpha^3 + \alpha^1$	00011010	26	α^{233}	$\alpha^7 + \alpha^6 + \alpha^5 + \alpha^4 + \alpha^1 + 1$	11110011	243
α^{106}	$\alpha^5 + \alpha^4 + \alpha^2$	00110100	52	α^{234}	$\alpha^7 + \alpha^6 + \alpha^5 + \alpha^4 + \alpha^3 + \alpha^1 + 1$	11111011	251
α^{107}	$\alpha^6 + \alpha^5 + \alpha^3$	01101000	104	α^{235}	$\alpha^7 + \alpha^6 + \alpha^5 + \alpha^3 + \alpha^1 + 1$	11101011	235
α^{108}	$\alpha^7 + \alpha^6 + \alpha^4$	11010000	208	α^{236}	$\alpha^7 + \alpha^6 + \alpha^3 + \alpha^1 + 1$	11001011	203
α^{109}	$\alpha^7 + \alpha^5 + \alpha^4 + \alpha^3 + \alpha^2 + 1$	10111101	189	α^{237}	$\alpha^7 + \alpha^3 + \alpha^1 + 1$	10001011	139
α^{110}	$\alpha^6 + \alpha^5 + \alpha^2 + \alpha^1 + 1$	01100111	103	α^{238}	$\alpha^3 + \alpha^1 + 1$	00001011	11
α^{111}	$\alpha^7 + \alpha^6 + \alpha^3 + \alpha^2 + \alpha^1$	11001110	206	α^{239}	$\alpha^4 + \alpha^2 + \alpha^1$	00010110	22
α^{112}	$\alpha^7 + 1$	10000001	129	α^{240}	$\alpha^5 + \alpha^3 + \alpha^2$	00101100	44
α^{113}	$\alpha^4 + \alpha^3 + \alpha^2 + \alpha^1 + 1$	00011111	31	α^{241}	$\alpha^6 + \alpha^4 + \alpha^3$	01011000	88
α^{114}	$\alpha^5 + \alpha^4 + \alpha^3 + \alpha^2 + \alpha^1$	00111110	62	α^{242}	$\alpha^7 + \alpha^5 + \alpha^4$	10110000	176
α^{115}	$\alpha^6 + \alpha^5 + \alpha^4 + \alpha^3 + \alpha^2$	01111100	124	α^{243}	$\alpha^6 + \alpha^5 + \alpha^4 + \alpha^3 + \alpha^2 + 1$	01111101	125
α^{116}	$\alpha^7 + \alpha^6 + \alpha^5 + \alpha^4 + \alpha^3$	11111000	248	α^{244}	$\alpha^7 + \alpha^6 + \alpha^5 + \alpha^4 + \alpha^3 + \alpha^1$	11111010	250
α^{117}	$\alpha^7 + \alpha^6 + \alpha^5 + \alpha^3 + \alpha^2 + 1$	11101101	237	α^{245}	$\alpha^7 + \alpha^6 + \alpha^5 + \alpha^3 + 1$	11101001	233
α^{118}	$\alpha^7 + \alpha^6 + \alpha^2 + \alpha^1 + 1$	11000111	199	α^{246}	$\alpha^7 + \alpha^6 + \alpha^3 + \alpha^2 + \alpha^1 + 1$	11001111	207
α^{119}	$\alpha^7 + \alpha^4 + \alpha^1 + 1$	10010011	147	α^{247}	$\alpha^7 + \alpha^1 + 1$	10000011	131
α^{120}	$\alpha^5 + \alpha^4 + \alpha^3 + \alpha^1 + 1$	00111011	59	α^{248}	$\alpha^4 + \alpha^3 + \alpha^1 + 1$	00011011	27
α^{121}	$\alpha^6 + \alpha^5 + \alpha^4 + \alpha^2 + \alpha^1$	01110110	118	α^{249}	$\alpha^5 + \alpha^4 + \alpha^2 + \alpha^1$	00110110	54
α^{122}	$\alpha^7 + \alpha^6 + \alpha^5 + \alpha^3 + \alpha^2$	11101100	236	α^{250}	$\alpha^6 + \alpha^5 + \alpha^3 + \alpha^2$	01101100	108
α^{123}	$\alpha^7 + \alpha^6 + \alpha^2 + 1$	11000101	197	α^{251}	$\alpha^7 + \alpha^6 + \alpha^4 + \alpha^3$	11011000	216
α^{124}	$\alpha^7 + \alpha^4 + \alpha^2 + \alpha^1 + 1$	10010111	151	α^{252}	$\alpha^7 + \alpha^5 + \alpha^3 + \alpha^2 + 1$	10101101	173
α^{125}	$\alpha^5 + \alpha^4 + \alpha^1 + 1$	00110011	51	α^{253}	$\alpha^6 + \alpha^2 + \alpha^1 + 1$	01000111	71
α^{126}	$\alpha^6 + \alpha^5 + \alpha^2 + \alpha^1$	01100110	102	α^{254}	$\alpha^7 + \alpha^3 + \alpha^2 + \alpha^1$	10001110	142
α^{127}	$\alpha^7 + \alpha^6 + \alpha^3 + \alpha^2$	11001100	204	α^{255}	1	00000001	1

Appendix E: Adeunis RF ARF6921D 869.525 MHz Board

E.1 Transmission Power Configuration

P0	P1	Power
0V	0V	+14 dBm (25 mW)
0V	open	+20 dBm (125 mW)
open	0V	+23 dBm (200 mW)
open	open	+27 dBm (500 mW)

Table E.1: Transmission power configuration. Adapted from [8]

E.2 Busy

The RF module is unavailable for use when changing modes. The switching timing signals are given by Figure E.1.

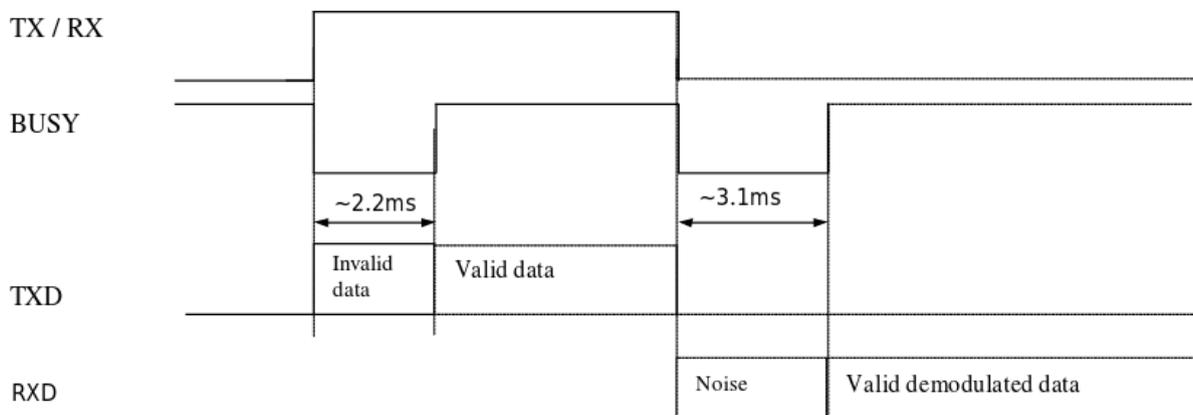


Figure E.1: Switching timing signals. Adapted from [8].

E.3 Specifications

	Parameter	Value	Conditions
Transmitter (Tx)	Maximum power	500mW (+27dBm)	In 50 ohms
	Modulation	FSK +/-40kHz	Version G (1-channel)
	Consumption	550 mA	- @max power in 50 Ohms
	Wake-up time	10ms	From Power_Down mode
	Rx to Tx turn-around time	2ms	
Receiver (Rx)	Sensitivity	-103 dBm (1.6uV)	Version H (2-channel)
	Passband @3dB	60 kHz	Version G (1-channel)
	Consumption	25 mA	
	Wake-up time	10 ms	From Power_Down mode
	Rx to Tx wake-up time	3 ms	
Transceiver	VCC power supply	Regulated 3V	
	Transmission rate	76,8 kBps NRZ i.e. 38.4kHz	G version (1-channel)
	Digital input/output	0/VCC	
	Standby consumption	400 μ A	Take care over the position of C2, P1 and P2
	Channel settling time	2 ms	
	Free field range	3 km	G version (1-channel)
	Temperature	From -20°C to +70°C	

Table E.2: ARF6921D Specifications

References

- [1] Le Roux, J.-H.: *Development of a satellite network simulator tool and simulation of AX.25, FX.25 and a hybrid protocol for nano-satellite communications*. Masters {Thesis}, University of Stellenbosch, 2014.
Available at: <http://hdl.handle.net/10019.1/96114>
- [2] William A. Beech, N., Douglas E. Nielsen, N., Jack Taylor, N., Beech, W., Nielsen, D., Taylor, J., Protocol, L.A., Radio, A.P., William A. Beech, N., Douglas E. Nielsen, N. and Jack Taylor, N.: *AX.25 Link Access Protocol for Amateur Packet Radio*. Tech. Rep., Tucson Amateur Packet Radio Corporation, jul 1998.
Available at:
<https://www.tapr.org/pdf/AX25.2.2.pdf><http://www.tapr.org/pdf/AX25.2.2.pdf>
- [3] Jim McGuire, K., Ivan Galysh, K., Doherty, K., Hank Heidt, N., Neimi, D., Jim McGuire, KB3MPL, Ivan Galysh, KD4HBO, Kevin Doherty, Hank Heidt, N4AFL and Dave Neimi: *Forward Error Correction Extension to AX.25 Link Protocol For Amateur Packet Radio*. Tech. Rep., Stensat Group LLC, 2006.
Available at: http://www.stensat.org/docs/FX-25_01_06.pdf
- [4] CCSDS Secretariat: *TC Space Data Recommendation for Space Data System Standards*. Tech. Rep. 2, CCSDS, Washington, DC, USA, 2010.
- [5] CCSDS Secretariat: *TC Synchronization and Channel Coding*. Tech. Rep., CCSDS, Washington, DC, USA, 2010.
Available at: <https://public.ccsds.org/Pubs/231x0b2c1.pdf>
- [6] Barbosa, T.C., More, R.L., Pereira, T.C., Ferreira, L.H.C., Moreno, R.L., Pimenta, T.C. and Ferreira, L.H.C.: *FPGA Implementation of a Reed-Solomon CODEC for OTN G.709 Standard with Reduced Decoder Area*. *2010 International Conference on Computational Intelligence and Software Engineering*, vol. 1303, no. 2, pp. 1–4, 2010.
Available at:
<http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5600886>
- [7] Lin, S. and Costello, D.J.: *Error Control Coding*. 2nd edn. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 2004. ISBN 0130426725.
- [8] RF, A.: *ARF29 Synchronous User Guide*. Tech. Rep. 0, Adeunis RF, Crolles, France, 2008.
- [9] Klofas, B.: *CubeSat Communications System Table*. Tech. Rep., SRI International, mar 2015.
Available at: <http://www.klofas.com/comm-table/table.pdf>
- [10] Wesdock, John; Patel, Chitra; Bustamante, Eduardo; Michel, Jeremy; Zhang, Y.: *Communications Receiver Performance Degradation Handbook*. Tech. Rep., Joint Spectrum Center, Annapolis, 2010.
Available at: <https://www.ntia.doc.gov/files/ntia/publications/jsc-cr-10-004final.pdf>
- [11] Messier, D.: *Tiny Cubesats Gaining Bigger Role in Space*. 2015.
Available at: <http://www.space.com/29464-cubesats-space-science-missions.html>

- [12] Labs, P.: Welcome to your planet. 2015.
Available at: <https://www.planet.com/>
- [13] Wikipedia: Low Earth orbit. 2015.
Available at: https://en.wikipedia.org/wiki/Low_Earth_orbit
- [14] Chu, V., Sun, W., Sweeting, M., Y, C.V., W, S., N, S.M., Kingdom, U., Chu V Y, Sun W and Sweeting M N: Optimising the LEO satellite communications link through hybrid-ARQ techniques. In: *International Conference on Information, Communications and Signal Processing.*, vol. 1, pp. 9–12. IEEE, 1997. ISBN 0-7803-3676-3.
- [15] James R. Wertz, Wiley J. Larson, Wertz, J.R. and Larson, W.J.: *Space Mission Analysis and Design*. 3rd edn. Microcosm Press, 1999.
- [16] Littman, M.K.: An Adaptive FEC with QoS Provisioning for Real-Time Traffic in LEO Satellite Networks. *Building Broadband Networks*, pp. 2938–2942, 2002.
Available at: <http://dx.doi.org/10.1201/9781420000016.ch10>
- [17] Emmelmann, M. and Bischl, H.: An adaptive MAC layer protocol for ATM-based LEO satellite networks. *2003 IEEE 58th Vehicular Technology Conference. VTC 2003-Fall (IEEE Cat. No.03CH37484)*, pp. 2698–2702 Vol.4, 2003. ISSN 07400551.
Available at:
<http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=1286054>
- [18] Cho, S.: Adaptive Error Control for Hybrid (Satellite-Terrestrial) Networks. *IEEE ICC*, pp. 1013–1017, 1999.
- [19] Ahn, J.-S., Heidemann, J., Rey, M., Hong, S.-w. and Heidemann, J.: An adaptive FEC code control algorithm for mobile wireless sensor networks. *Information Sciences*, vol. 7, no. 4, pp. 489–498, 2005. ISSN 1229-2370.
Available at:
<http://ieeexplore.ieee.org/xpl/login.jsp?tp=&arnumber=6387991&url=http://ieeexplore.ieee.org/iel5/5449605/6387978/06387991.pdf?arnumber=6387991>
<http://network.dongguk.ac.kr/publication/JCN05-2-007.pdf>
- [20] Lamoriniere, C., Nafaa, A. and Murphy, L.: Dynamic switching between adaptive FEC protocols for reliable multi-source streaming. *GLOBECOM - IEEE Global Telecommunications Conference*, 2009.
- [21] Bolot, J.-C., Fosse-Parisis, S. and Towsley, D.: Adaptive FEC-based error control for Internet telephony. *Conference on Computer Communications.*, pp. 1453–1460 vol.3, 1999. ISSN 0743-166X.
Available at: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=752166
- [22] Shimiz, K., Togawatt, N., Ohtsditt, T., Engineering, S. and Sciences, M.: A RECONFIGURABLE ADAPTIVE FEC SYSTEM FOR RELIABLE WIRELESS COMMUNICATIONS. *The 2004 IEEE Asia-Pacific Conference on Circuits and Systems*, pp. 13–16, 2004.
- [23] Blahut, R.E.: *Algebraic Codes for Data Transmission*. Cambridge University Press, 2003. ISBN 9781139435079.

- [24] Wikipedia: Cubesat Space Protocol. jun 2015.
Available at: https://en.wikipedia.org/w/index.php?title=Cubesat_Space_Protocol&oldid=669449648<http://gomspace.com/documents/GS-CSP-1.1.pdf>
- [25] Muri, P. and McNair, J.: A survey of communication sub-systems for intersatellite linked systems and cubesat missions. *Journal of Communications*, 2012.
Available at:
<http://www.ojs.academypublisher.com/index.php/jcm/article/view/jcm0704290308>
- [26] Secretariat, C.: Overview of Space Communications Protocols. Informational Report, Green Book CCSDS 130.0-G-3, Consultative Committee for Space Data Systems, Washington DC, jul 2014.
Available at: <http://public.ccsds.org/publications/archive/130x0g3.pdf>
- [27] Milne, G.W., Schoonwinkel, A., du Plessis, J.J., Mostert, S., Steyn, W.H., vd Westhuizen, K., vd Merwe, D.A., Grobler, H., Koekemoer, J.A. and Steenkamp, N.: SUNSAT-Launch and First Six Month's Orbital Performance. In: *13th Annual AIAA/USU Conference on Small Satellites*. 1999.
- [28] Wikipedia: ns (simulator). 2016.
Available at: [https://en.wikipedia.org/wiki/Ns_\(simulator\)](https://en.wikipedia.org/wiki/Ns_(simulator))
- [29] NS_project: Network Simulator. 2016.
Available at: <https://www.nsnam.org/>
- [30] Puttonen, J. and Kurjenniemi, J.: Satellite Network Simulator 3 (SNS3). 2016.
Available at: <http://satellite-ns3.com/>
- [31] Ltd, O.: OMNeT++ Discrete Event Simulator - Home. 2015.
Available at: <https://omnetpp.org/>
- [32] Riverbed: OPNET IT Guru Academic Edition. 2016.
Available at: http://www.opnet.com/university_program/itguru_academic_edition/
- [33] Simpy 3.0.7. 2015.
Available at: <https://simpy.readthedocs.org/>
- [34] PyEphem. 2015.
Available at: rhodesmill.org/pyephem/
- [35] Weisstein, E.W.: "Erfc." From MathWorld. 2016.
Available at: <http://mathworld.wolfram.com/Erfc.html>
- [36] Campbell, B.A., Jr., P.W.M., Samuel Walter McCandless, J., Bruce A. Campbell and Samuel Walter McCandless, Jr.: *Introduction to Space Sciences and Spacecraft Applications*. Gulf Professional Publishing, 1996. ISBN 0080478735.
Available at: https://books.google.co.za/books/about/Introduction_to_Space_Sciences_and_Space.html?id=e-IeVouismzoC&pgis=1
- [37] Union, I.T.: Attenuation By Atmospheric Gases, Recommendation ITU-R P.676-11 (09/2016). Tech. Rep., ITU-R, Geneva, Switzerland, 2016.

- [38] Moon, T.K.: *Error Correction Coding: Mathematical Methods and Algorithms*. Wiley, 2005. ISBN 0471739146.
Available at: <https://books.google.com/books?id=adxb8CRx5vQC&pgis=1>
- [39] Bertsekas, D.P. and Tsitsiklis, J.N.: *Introduction to Probability*. Athena Scientific books. Athena Scientific, 2002. ISBN 9781886529403.
Available at: <https://books.google.co.za/books?id=bcHaAAAAMAAJ>
- [40] Secretariat, C.: TC Space Data Link Protocol. sep 2003.
Available at: <http://public.ccsds.org/publications/archive/232x0b1s.pdf>
- [41] Kazz, G.J., Greenberg, E., Burleigh, S.C. and Ngu, G.U.: Replacing the CCSDS Telecommand Protocol with the Next Generation Uplink (NGU). *American Institute of Aeronautics and Astronautics*, 2012.
- [42] Clarke, C.: Reed-Solomon error correction. *BBC R&D White Paper, WHP*, 2002.
Available at: https://scholar.google.co.za/scholar?q=reed+solomon+error+correction+bbc&btnG=&hl=en&as_sdt=0%252C5#0
- [43] Wai, K. and Yang, S.: Field programmable gate array implementation of Reed-Solomon code, RS (255,239). *Nyworkshop*, 2005.
Available at: <http://www.ce.rit.edu/~sjyec/papers/nyworkshop-rs-codec.pdf>
- [44] Wai, K., Chung, K., Wai, C., Bush, D. and Bush, D.: FPGA implementation of Reed Solomon codec for 40Gbps Forward Error Correction in optical networks. *Theses*, 2006.
Available at: <http://scholarworks.rit.edu/theses/8048>