



# A system on chip based error detection and correction implementation for nanosatellites

by

**Caleb Pedro Hillier**

*Thesis submitted in fulfilment of the requirements for the degree Master of Engineering in  
Electrical Engineering in the Faculty of Engineering at the Cape Peninsula University of  
Technology*

Supervisor: Dr. Vipin Balyan

Bellville

October 2018

CPUT Copyright Information

This dissertation/thesis may not be published either in part (in scholarly, scientific or technical journals), or as a whole (as a monograph), unless permission has been obtained from the university

## DECLARATION

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



10/10/2018

---

Signed

---

Date

Caleb Hillier  
CPUT Bellville  
October 2018

## ABSTRACT

This thesis will focus on preventing and overcoming the effects of radiation in RAM on board the ZA cube 2 nanosatellite. The main objective is to design, implement and test an effective error detection and correction (EDAC) system for nanosatellite applications using a SoC development board. By conducting an in-depth literature review, all aspects of single-event effects are investigated, from space radiation right up to the implementation of an EDAC system. During this study, Hamming code was identified as a suitable EDAC scheme for the prevention of single-event effects.

During the course of this thesis, a detailed radiation study of ZA cube 2's space environment is conducted. This provides insight into the environment to which the satellite will be exposed to during orbit. It also provides insight which will allow accurate testing should accelerator tests with protons and heavy ions be necessary. In order to understand space radiation, a radiation study using ZA cube 2's orbital parameters was conducted using OMERE and TRIM software. This study included earth's radiation belts, galactic cosmic radiation, solar particle events and shielding. The results confirm that there is a need for mitigation techniques that are capable of EDAC.

A detailed look at different EDAC schemes, together with a code comparison study was conducted. There are two types of error correction codes, namely error detection codes and error correction codes. For protection against radiation, nanosatellites use error correction codes like Hamming, Hadamard, Repetition, Four Dimensional Parity, Golay, BCH and Reed Solomon codes. Using detection capabilities, correction capabilities, code rate and bit overhead each EDAC scheme is evaluated and compared. This study provides the reader with a good understanding of all common EDAC schemes.

The field of nanosatellites is constantly evolving and growing at a very fast speed. This creates a growing demand for more advanced and reliable EDAC systems that are capable of protecting all memory aspects of satellites. Hamming codes are extensively studied and implemented using different approaches, languages and software. After testing three variations of Hamming codes, in both Matlab and VHDL, the final and most effective version was Hamming  $[16, 11, 4]_2$ . This code guarantees single error correction and double error detection. All developed Hamming codes are suited for FPGA implementation, for which they are tested thoroughly using simulation software and optimised.

## ACKNOWLEDGEMENTS

The author would like to thank the following people for their contribution which led to the success of this thesis, and without whose support its completion would not have been possible:

- **Dr. Vipin Balyan** (supervisor), for his invaluable support, advice and guidance throughout this thesis.
- **Dr. Yaseen**, for all his help in selecting a thesis topic, as well as all advice and guidance offered.
- **Prof. Francois Rocaries** and **Dr. Ifriky Tadadjeu**, for the advice and guidance upon their review of my proposal.
- **Ashley van Oudtshoorn**, for all her love, support and for reminding me to stay focused through all the ups and downs.
- **My family** and **friends**, for their love, support, and prayers.
- **Prof. Robert van Zyl** for his support and for presenting me with the opportunity to pursue a master's degree.
- **French South African Institute of Technology (F'SATI)**, for their generous support through NRF financial assistance, equipment and additional courses to assist me during the completion of this thesis.
- **God**, who assisted me through it all.
- The financial assistance of the **National Research Foundation** towards this research is acknowledged. Opinions expressed in this thesis and the conclusions arrived at, are those of the author, and are not necessarily to be attributed to the National Research Foundation.

## LIST OF PUBLICATION

1. Hillier, C. & Balyan, V., 2018. Review Paper: Error Detection and Correction onboard Nanosatellites. *2018 International Conference on Advanced Computation and Telecommunication (ICACAT)*. [Accepted]
2. Hillier, C. & Balyan, V., 2018. Effect of space radiation on LEO nanosatellites. *Journal of Applied Engineering Science (JAES)*. [Under review].
3. Hillier, C. & Balyan, V., 2019. Error Detection And Correction On-Board Nanosatellites Using Hamming Codes. *Journal of Electrical and Computer Engineering*. [Accepted]

## TABLE OF CONTENTS

Declaration .....	i
Abstract .....	ii
Acknowledgements.....	iii
List of publication .....	iii
Table of Figures.....	vii
Table of Tables .....	viii
Appendices.....	ix
Glossary .....	x
Chapter 1.....	1
Introduction .....	1
1.1 Statement of the research problem .....	1
1.2 Background.....	1
1.3 Review of literature.....	2
1.4 Research questions.....	12
1.5 Objectives of the research.....	12
1.6 Delineation of the research.....	12
1.7 The significance of the research.....	13
1.8 Thesis layout.....	14
Chapter 2.....	15
Space Radiation .....	15
2.1 Introduction .....	15
2.2 ZA-CUBE 2 orbital parameters .....	16
2.3 Understanding radiation .....	17
2.4 Earth radiation belts .....	21
2.5 Solar cosmic radiation .....	26
2.6 Galactic cosmic radiation.....	28
2.7 Shielding .....	29
2.8 Summary.....	35
Chapter 3.....	36

Error correcting codes .....	36
3.1 Introduction .....	36
3.2 Error detection and correction .....	37
3.3 Error detection and correction schemes .....	41
3.4 EDAC Selection .....	48
Chapter 4.....	51
Hamming code .....	52
4.1 Introduction .....	52
4.2 Overview of Hamming code .....	55
4.3 Design process .....	60
4.4 Implementation.....	70
4.5 VHDL optimization of Hamming $[16, 11, 4]_2$ .....	72
Chapter 5.....	77
Simulation and Test results of Hamming code in VHDL.....	77
5.1 Introduction .....	77
5.2 Software tests and reports.....	78
5.3 Optimisation .....	83
5.4 Hardware tests.....	86
Chapter 6.....	90
Conclusion .....	90
6.1 Findings .....	90
6.2 Outcomes.....	92
6.3 Recommendations .....	93
References .....	95
Appendices.....	98
8.1 Simulink model of Hamming $[7, 4, 3]$ (proof of concept) .....	98
8.2 Matlab code for Hamming $[16, 11, 4]_2$ .....	99
8.3 Explanation and code flow of Hamming $[16, 11, 4]_2$ in Matlab .....	101
8.4 Explanation and code flow of Hamming $[16, 11, 4]_2$ in VHDL.....	102
8.5 VHDL code for Hamming $[16, 11, 4]_2$ – main .....	103

8.6	VHDL code for Hamming $[16, 11, 4]_2$ - encoder.....	104
8.7	VHDL code for Hamming $[16, 11, 4]_2$ – decoder.....	106
8.8	Gate level VHDL code for Hamming $[16, 11, 4]_2$ – encoder .....	108
8.9	Gate level VHDL code for Hamming $[16, 11, 4]_2$ – decoder .....	109
8.10	VHDL code for Hamming $[16, 11, 4]_2$ - testbench .....	111

## TABLE OF FIGURES

Figure 1-1: ZA cube 1, a 1U CubeSat (CPUT 2017).....	1
Figure 1-2: The Earth's Magnetosphere (Miller 2012) .....	5
Figure 1-3: ERBs including 2 Van Allen Probes satellites (Zell 2013).....	5
Figure 1-4: SAA Using STK SEET (System Tool Kit (STK) 2017) .....	6
Figure 1-5: Orbital position of OBC386 Ramdisk memory upsets (Bentoutou 2012)...	6
Figure 1-6: Block diagram of TMR-based EDAC (Bentoutou 2012).....	9
Figure 2-1: ZACube-2 Conceptual Layout (Villiers & Zyl n.d.) .....	15
Figure 2-2: Omere orbital parameters: initialisation (left) and output file (right) .....	16
Figure 2-3: Heliophysics and Space Weather (Bensusen et al. 2013) .....	17
Figure 2-4: Effects of heavy ions (left) & protons (right) - (Halbert 2006).....	18
Figure 2-5: Magnetic field (Jensen Cain) at 550km in 2018.....	21
Figure 2-6: Orbital average integral and differential fluxes of trapped particles.....	22
Figure 2-7: Maximum trapped electrons (differential) .....	23
Figure 2-8: Maximum trapped electrons (integral) .....	23
Figure 2-9: Maximum trapped protons (differential) .....	24
Figure 2-10: Maximum trapped protons (integral).....	24
Figure 2-11: Orbital minimum trapped electrons – AE8 – Jensen Cain – Differential	25
Figure 2-12: Orbital minimum trapped protons – AP8 – Jensen Cain – Differential ..	25
Figure 2-13: Solar Particle (Proton) – Setup.....	26
Figure 2-14: Integral and differential fluences of solar protons .....	26
Figure 2-15: Solar Particle (Ion) Z = 2 to Z = 17 – Setup.....	27
Figure 2-16: Integral and differential fluence of solar ions Z = 2 (He) to Z = 17 (Cl) ..	27
Figure 2-17: Heavy ion integral (Z = 1 to Z=17).....	28
Figure 2-18: Heavy ion differential (Z = 1 to Z =17) .....	28
Figure 2-19: Dose (rad) vs thickness.....	29
Figure 2-20: Dose (rad) vs thickness (zoom in) .....	30
Figure 2-21: Orbital dosage graph for 2mm Aluminium shield .....	30
Figure 2-22: Fluxes of incident and transmitted trapped protons in ERBs .....	31
Figure 2-23: Fluxes of incident and transmitted solar peak and mean heavy ions ....	31
Figure 2-24: Integral LET spectrum diagram for 98° and 2mm Al shield .....	32
Figure 2-25: TRIM set-up windows.....	33
Figure 2-26: Protons (+ hydrogen) of 10 MeV depth into 2mm aluminium .....	33
Figure 2-27: Protons (+ hydrogen) of 19 MeV depth into 2mm aluminium.....	34
Figure 3-1: Classification of error control systems.....	39
Figure 3-2: EDAC layout (FEC).....	40
Figure 3-3: General construction of a codeword.....	40
Figure 3-4: Barcode using 2-out-of-5.....	42



Figure 3-5: Likelihood of error occurring in a day .....	51
Figure 4-1: Classification of Hamming code .....	53
Figure 4-2: General layout of Hamming code .....	55
Figure 4-3: Parity VS data, Hamming (7,3) (left) & Hamming (15,11) (right) .....	57
Figure 4-4: Gate/graphical expression of Hamming encoder formula .....	58
Figure 4-5: Gate/graphical expression of Hamming decoder formula .....	59
Figure 4-6: Parity relationship to data, extended Hamming (8, 4) .....	59
Figure 4-7: Overview of the design process .....	60
Figure 4-8: Overview of Matlab code (flow chart) .....	70
Figure 4-9: Overview of VHDL code (flow chart) .....	71
Figure 4-10: I/O overview of VHDL code (hamming_11_16_main.vhd) .....	72
Figure 4-11: RTL overview of the encoder (hammen16.vhd).....	73
Figure 4-12: RTL overview of the decoder (hammde16.vhd).....	73
Figure 4-13: RTL overview of the optimised decoder (hammde16.vhd).....	75
Figure 4-14: Quartus Prime advisor optimization options .....	76
Figure 4-15: Timing (left) and Resource (right) Optimization Advisor .....	76
Figure 4-16: Resource Optimization Advisor breakdown.....	76
Figure 5-1: Full RTL overview of VHDL code (hamming_11_16_main.vhd) .....	78
Figure 5-2: ModelSim simulation of Hamming [16, 11, 4] SECC capabilities .....	79
Figure 5-3: Resource usage report on non-optimised Hamming (16, 11, 4) .....	80
Figure 5-4: ALM for Intel Stratix series (Intel 2018) .....	81
Figure 5-5: Timing report – Path summary for non-optimised Hamming (16, 11, 4)..	82
Figure 5-6: Timing report – Waveform for non-optimised Hamming (16, 11, 4) .....	82
Figure 5-7: Resource usage report on optimised Hamming (16, 11, 4) .....	83
Figure 5-8: Timing report (TimeQuest) for resource optimised Hamming (16, 11, 4)	84
Figure 5-9: Timing report (TimeQuest) for timing optimised Hamming (16, 11, 4) ....	85

## TABLE OF TABLES

Table 1-1: Effects of charged particles in a space environment. (Holbert 2007) .....	3
Table 1-2: Effects of radiation on CMOS devices. (Wall & Macdonald 1993).....	3
Table 1-3: Summary of well-known EDAC schemes .....	8
Table 2-1: Possible SEE as a function of component technology and family .....	19
Table 3-1: ECC history.....	37
Table 3-2: Error detection and correction capabilities of common EDAC schemes ..	41
Table 3-3: Hamming code classification and parameters .....	43
Table 3-4: Hadamard code classification and parameters.....	44

Table 3-5: Golay codes classification and parameters .....	45
Table 3-6: BCH code classification and parameters .....	46
Table 3-7: Reed Solomon code classification and parameters .....	47
Table 3-8: EDAC scheme comparison .....	49
Table 3-9: Memory errors that were observed on Alsat-1 during a 7 year period.....	50
Table 4-1: Hamming code classification and parameters .....	52
Table 4-2: Systematic and non-systematic codewords.....	54
Table 4-3: Bit layout of Hamming code (non-systematic) .....	56
Table 4-4: Bit layout of extended Hamming code (16, 11).....	59
Table 4-5: Construction of the Hamming [7, 4, 3] codeword.....	61
Table 4-6: Calculated performance aspects of Hamming [7, 4, 3] .....	61
Table 4-7: Construction of the Hamming [8, 4, 4] codeword.....	63
Table 4-8: Calculated performance aspects of Hamming [8, 4, 4] .....	63
Table 4-9: Construction of the Hamming [16, 11, 4] codeword .....	66
Table 4-10: Calculated performance aspects of Hamming [16, 11, 4] .....	66
Table 4-11: Deriving gate-level code from VHDL code and RTL viewer .....	74
Table 5-1: Cyclotron parameters at iThemba labs (iThemba n.d.).....	89
Table 6-1: Evaluation of Hamming code.....	91
Table 6-2: Summary of developed codes .....	91
Table 6-3: Original VS optimised Hamming compression.....	92

## APPENDICES

8.1	Simulink model of Hamming [7, 4, 3] (proof of concept)
8.2	Matlab code for Hamming [16, 11, 4] <sub>2</sub>
8.3	Explanation and code flow of Hamming [16, 11, 4] <sub>2</sub> in Matlab
8.4	Explanation and code flow of Hamming [16, 11, 4] <sub>2</sub> in VHDL
8.5	VHDL code for Hamming [16, 11, 4] <sub>2</sub> – main
8.6	VHDL code for Hamming [16, 11, 4] <sub>2</sub> - encoder
8.7	VHDL code for Hamming [16, 11, 4] <sub>2</sub> – decoder
8.8	Gate level VHDL code for Hamming [16, 11, 4] <sub>2</sub> – encoder
8.9	Gate level VHDL code for Hamming [16, 11, 4] <sub>2</sub> – decoder
8.10	VHDL code for Hamming [16, 11, 4] <sub>2</sub> - testbench
8.11	ICACAT acceptance letter
8.12	J.A.E.S. submission acknowledgement

## GLOSSARY

<b>Abbreviations</b>	<b>Definition</b>
A/D	Analog to Digital
ADCS	Attitude Determination and Control System
ALM	Adaptive Logic Module
ALUT	Adaptive Look-Up Tables
ARQ	Automatic Repeat-reQuest
ASIC	Application Specific Integration Circuit
CGR	Cosmic Galactic Radiation
CLK	Clock
COTS	Commercial off-the-shelf
DEC	Double Error Correction
DED	Double Error Detection
DMA	Direct Memory Access
DSP	Digital Signal Processing
DUT	Device Under Test
ECC	Error Correcting Code
ECSS	European Cooperation for Space Standardization
EDAC	Error Detection and Correction
ERB	Earth Radiation Belts
FEC	Forward Error Correction
FF	Flip Flops
FPGA	Field Programmable Gate Array
GCR	Galactic Cosmic Radiation
HARQ	Hybrid Automatic Repeat-reQuest
HDL	Hardware Descriptive Language
HVD	Horizontal Vertical Diagonal
I/O	Input / Output
LEO	Low Earth Orbit
LUT	Look-Up Table
MDPC	Multi-Dimensional Parity-check Code
MEC	Multiple Error Correction
MED	Multiple Error Detection
MMU	Memory Management Unit
NASA	National Aeronautics and Space Administration
NRF	National Research Fund

OBC	On-board Computer
PCB	Printed Circuit Board
POR	Power On Reset
RTL	Register Transfer Level
S/W	Software
SAA	South Atlantic Anomaly
SANSA	South Africa National Space Agency
SCR	Solar Cosmic Radiation
SEC	Single Error Correction
SED	Single Error Detection
SEE	Single Event Effects
SEL	Single Event Latch-up
SEU	Single Event Upset
SoC	System on Chip
SPE	Solar Particle Events
SRAG	Space Radiation Analysis Group
SRAM	Static Random Access Memory
TID	Total Ionizing Dose
TMR	Triple Modular Redundancy
TR	Temporal Redundancy
AIS	Automatic Identification System
ESL	Electronic Systems Laboratory
ESP	Emission of Solar Protons
HCD	Human capacity development
RAM	Random Access Memory
SEFI	Single Event Functional Interrupt
SEGR	Single Event Gate Rupture
SHE	Single Event Hard Errors
SET	Single Event Transient
SSO	Sun Synchronous Orbits
TRIM	Transport of Ions in Matter
VDE	VHF Data Exchange
VHDL	VHSIC Hardware Description Language

# CHAPTER 1.

## INTRODUCTION

### 1.1 Statement of the research problem

This thesis will focus on preventing and overcoming the effects of radiation in RAM of nanosatellites. The issues that will be addressed are the single event upsets (SEU) and multiple event upsets (MEU) caused by space radiation. This study will include the protection of memory found within the satellites OBC and develop an effective EDAC based on a SoC (FPGA) development board specifications. All EDAC designs will be tested thoroughly using simulation software.

### 1.2 Background

Nanosatellites are small satellites often referred to as CubeSat (Figure 1-1) due to their physical appearance. A nanosatellite can weigh between 1 kg to 10 kg. Owing to their fast development time and low manufacturing costs, these satellites have become extremely popular for university programs around the world.



**Figure 1-1: ZA cube 1, a 1U CubeSat (CPUT 2017)**

In the past few years, nanosatellites have evolved into the ideal platform to test new technologies, discover more about space and for developing the skills of future engineers. Most nanosatellites are made up of off-the-shelf components. This is done to ensure fast development and to minimise the cost.

During the history of satellites, many SEU and MEU have been recorded. These upsets are mainly owing to radiation. The South African Sumbandila microsatellite is an extreme example of the damage these events can cause as it was rendered unresponsive after being exposed to a severe amount of radiation (Keith Campbell 2012).

In order to ensure data integrity and to prevent disasters like Sumbandila, methods such as radiation-hardened devices and EDAC systems have been developed. This thesis focuses on an EDAC-based solution, as digital/programmed solutions are more cost effective and can be implemented using existing systems.

### **1.3 Review of literature**

This section's main objective is to provide a solid foundation and show the results of research already done in the field of EDAC schemes.

#### **1.3.1 Research background**

The research conducted during the completion of this thesis touches on a number of aspects of engineering and scientific phenomena. EDAC systems have been around for quite some time and certain EDAC schemes have been implemented and tested extensively. There are a number of articles, journals and thesis that are focused at EDAC's in general. However, this thesis is a more focused study on finding the best suited EDAC solution for nanosatellites.

Based on my existing knowledge and knowledge gained completing an undergraduate degree in electrical engineering at Cape Peninsula University of Technology (CPUT) I will research, develop and implement an EDAC scheme, designed specifically for the nanosatellites orbiting at low-earth orbits (LEO).

Most research was conducted using the IEEE Explore database. The IEEE Explore database aim is to allow full-text access to the world's highest-quality technical literature in engineering and technology.

#### **1.3.2 Search string**

Based on the obtained information and the problem statement, certain research topics were identified. The following list shows the main search strings used during conducted research:

- Space radiation
- Glitches and upsets
- Geomagnetism
- EDAC schemes
- Implementing EDAC systems

#### **1.3.3 Space radiation**

Space radiation is a general term given to ionizing radiation found in space. This form of radiation is made up of highly energized particles which are mainly protons and heavy ions. The sources of space radiation are identified by NASA's Space Radiation Analysis Group (SRAG) in the following manner: "There are three naturally occurring sources of space radiation: trapped radiation, galactic cosmic radiation (GCR), and solar particle events (SPE)" (Langford 2014). According to sources on radiation, SPE

has the biggest impact on satellites. The effects of radiation are often minor, however there are some extreme examples of incidents occurring that have cause space missions to fail (Keith Campbell 2012).

Radiated solar particles are responsible for electronic malfunctions, deterioration of materials and surface charging and discharging. Due to the severe impact radiation can have on a satellite, radiation is considered to be a fundamental and important factor when it comes to the design and operations of satellites (Maki 2009).

The effects of charged particles in a space environment is summarized using Table 1-1. Table 1-1 shows the three main sources of the radiation in space along with their effects on CMOS devices.

**Table 1-1: Effects of charged particles in a space environment. (Holbert 2007)**

Spacecraft Charging	Total Ionizing Dose	Displacement Damage	Single Event Effects
<ul style="list-style-type: none"> <li>· Surface Charging from Plasma</li> <li>· Deep Dielectric from High Energy Electrons</li> </ul>	<ul style="list-style-type: none"> <li>· Trapped Protons and Electrons</li> <li>· Solar Protons</li> </ul>	<ul style="list-style-type: none"> <li>· Protons</li> <li>· Electrons</li> </ul>	<ul style="list-style-type: none"> <li>· Protons: both Trapped and Solar</li> <li>· Heavy Ions: both Galactic Cosmic Rays and Solar Events</li> </ul>

**Table 1-2: Effects of radiation on CMOS devices. (Wall & Macdonald 1993)**

Space Radiation Environments and their Effects on CMOS Devices		
Radiation Source	Particle Types	Primary Effects in Devices
Trapped radiation belts	Electrons	Ionization damage
	Protons	Ionization damage; SEE in sensitive devices
Galactic cosmic rays	High-energy charged particles	Single-event effects (SEEs)
Solar flares	Electrons	Ionization damage
	Protons	Ionization damage; SEE in sensitive devices
	Lower energy/heavy-charged particles	SEE

It is important to ensure that all electronic devices and components used to make up any space system have been tested and are able to resist the dose of radiation they might be exposed to. This is essential to ensure data integrity and reliability.

#### **1.3.4 Glitches and upsets**

A glitch is defined as a sudden, usually temporary malfunction or fault of equipment. Glitches experienced by a satellite can be caused by many factors, such as hardware problems, corrupted software or space weather. The exact cause of glitches and upsets is often hard to determine, especially when satellites are exploring unknown areas of space and interplanetary space.

The first warnings for single event upsets were first brought to light by Wallmark and Marcus in 1962 (Wallmark & Marcus 1962). SEU are errors that occur in electric and digital circuits. According to NASA these SEU occur: "when charged particles lose energy by ionizing the medium through which they pass, leaving behind a wake of electron-hole pairs" (NASA/SP 2012). MEU occurs when two or more bits are upset by a single ion. These upsets are usually soft errors and can be corrected by on-board EDACs or prevented using hardened devices.

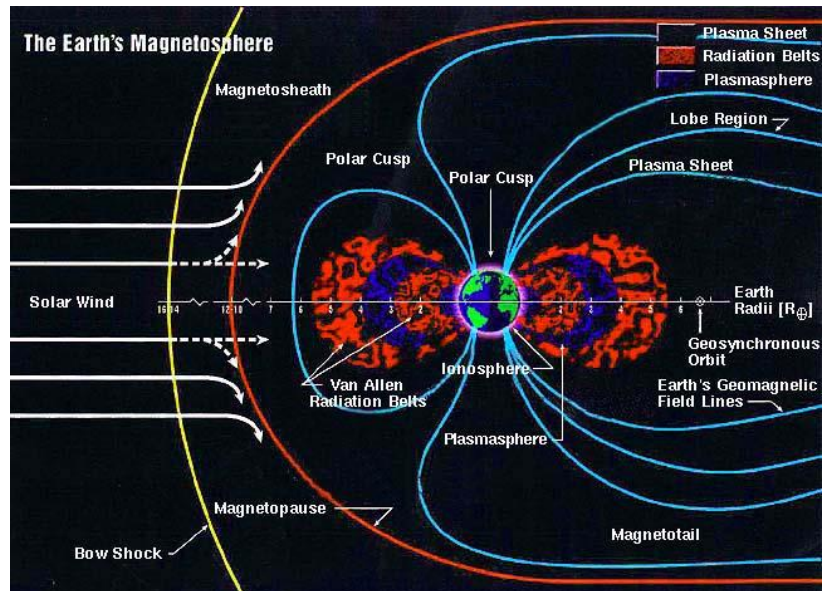
There are numerous examples of satellites being affected by glitches and upsets. The Magellan spacecraft on route to Venus suffered both power-panel and star-tracker upsets after being exposed to a solar flare (Odenwald 2001). Intelsat officials reported on the 13<sup>th</sup> of January 2011 that their Intelsat's Galaxy 15 telecommunications satellite was unable to receive any commands from earth for eight months due to electrostatic discharge that caused a major software error (Choi 2011).

The source of these types of glitches can normally be determined by analysing the housekeeping data of the satellite. The referred articles are relevant to this project as solving the problem of glitches and upsets is ultimately the aim of the thesis.

#### **1.3.5 Geomagnetism**

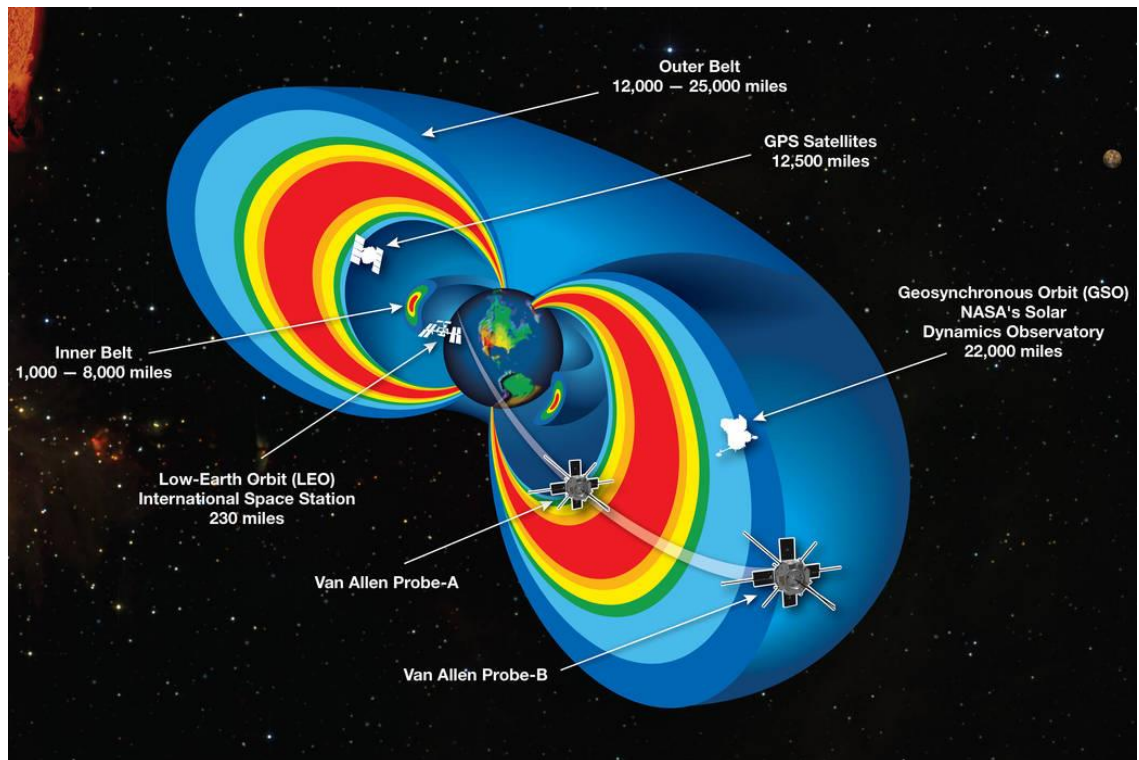
Geomagnetism refers to the earth's magnetic field. This field expands into outer space from the earth's core, affecting orbiting spacecraft and also deflects particles from outer space. Figure 1-2 provides a well-laid-out illustration of the magnetic fields produced by the earth, as well as the factors that influence it.





**Figure 1-2: The Earth's Magnetosphere (Miller 2012)**

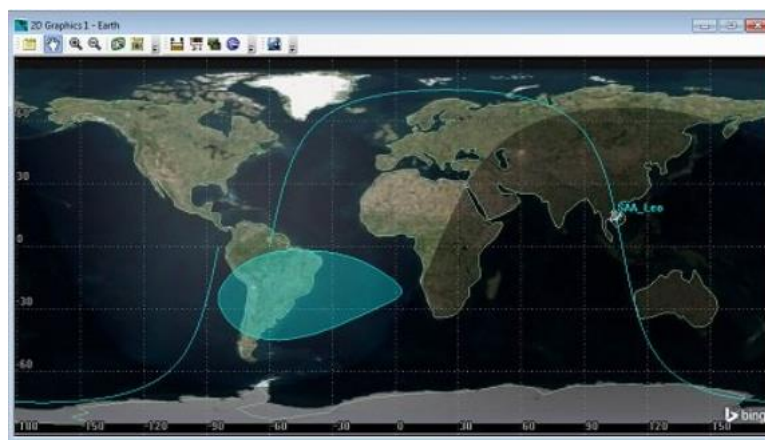
Geomagnetism is directly linked to radiation, as its fields cause particles to be trapped, which results in belts, like the Van Allen Radiation Belts. The Van Allen Radiation Belts contain highly energized particles which can have devastating results on unprotected satellites. In 2012, NASA launched two Van Allen Probes spacecrafts with the mission to “study two extreme and dynamic regions of space known as the Van Allen Radiation Belts that surround Earth” (Zell 2015). Figure 1-3 shows both the inner and outer radiation belts.



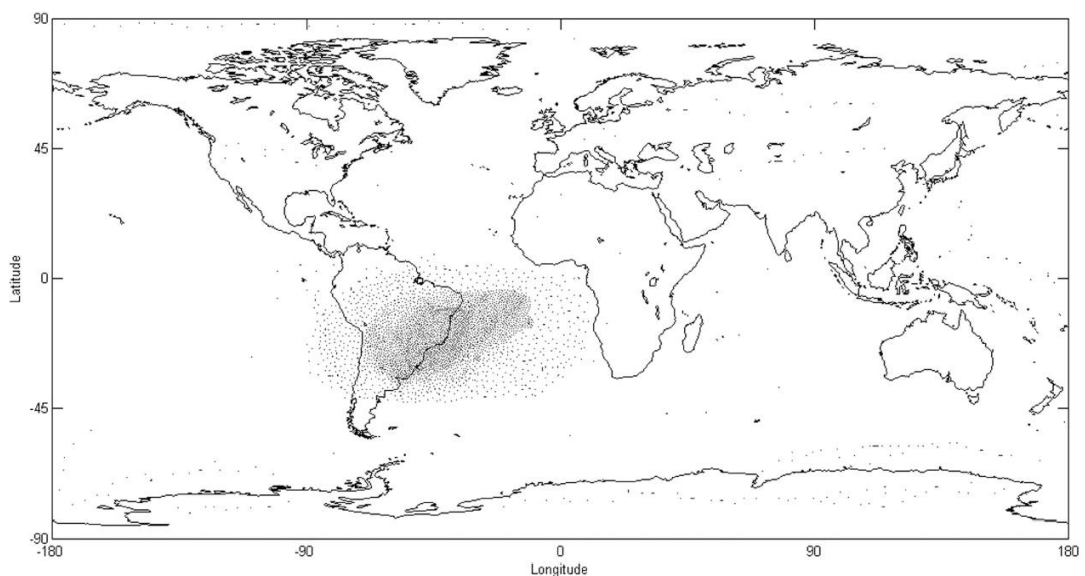
**Figure 1-3: ERBs including 2 Van Allen Probes satellites (Zell 2013)**

When discussing radiation with regards to nanosatellites, the South Atlantic Anomaly (SAA) is the main culprit of single error upsets (SEU) and multiple error upsets (MEU). The South Atlantic Anomaly (Figure 1-4) is the area where the earth's magnetic field is at its weakest. This means it is the area where the Van Allen Radiation belt is the closest to earth.

From an article written by Y. Bentoutou, the effect of radiation on board the Alsat-1 spacecraft is shown clearly in Figure 1-5. The orbital location of each upset that occurred from the 29<sup>th</sup> November 2002 to 12<sup>th</sup> October 2009 is plotted. It can be noted that the majority (+/- 80%) of SEU fell within the SAA (Bentoutou 2012).



**Figure 1-4: SAA Using STK SEET (System Tool Kit (STK) 2017)**



**Figure 1-5: Orbital position of OBC386 Ramdisk memory upsets (Bentoutou 2012)**

From the provided information on Geomagnetism, its connection and relevance to nanosatellites design and implementation become clear. It is important that methods and schemes are designed that will allow Nano-satellites to withstand and operate normally within environments produced by SAA and others.

### **1.3.6 EDAC schemes**

EDAC systems are responsible for ensuring reliable data transfer between the satellites onboard computer and its local memory. An EDAC system is a software solution to prevent the effects of radiation on a satellite. There are a number of EDAC schemes that have been developed during the past few years. Below is a list of some of these EDAC schemes that have been developed (Ahmad et al. 2013):

- 2 of 5 Code
- Berger Code
- Parity Code (Bilal et al. 2013)
- Hamming Code (Jindal 2006)
- Hadamard Code
- Repetition Code
- Four Dimensional Parity Code (Bilal et al. 2013)
- Golay Code (Kanemasu 1999)
- BCH Code (Poolakparambil et al. 2011)
- Reed Solomon Code (Parvathi 2015)

However, each scheme has its own advantages and disadvantages. These advantages and disadvantages are summarized in Table 1-3: Summary of well-known EDAC schemes.

**Table 1-3: Summary of well-known EDAC schemes**

EDAC scheme	SED	SEC	DED	DEC	MED	MEC	Description
Parity Code	Yes	No	No	No	No	No	This scheme is considered the simplest and most basic error detection scheme. Using a parity bit the scheme determines whether the string of bits is even or odd. This is done by evaluating the 1s contained in the string. Creating two variants, even and odd parity bit.
2 of 5 Code	Yes	No	No	No	No	No	Most popular was the 2-out-of-5 code, which allows decimal digits to be represented using five bits. This code was implemented in barcodes. The m-out-of-n code makes use of codeword weightings (m) and length (n) to perform error detection. The weighting value normally represents the sum of the 1's within a codeword.
Berger Code	Yes	No	No	No	No	No	This unidirectional error detecting code is only capable of flipping ones into zeroes or only zeroes into ones, such as in asymmetric channels. Used mainly in telecommunications. Berger Code counts all the ones or zeroes within the information data (k bits long) and then attaches the binary equivalent of the sum to the information forming the codeword (n + k bits).
Hamming Code	Yes	Yes	No	No	No	No	This scheme adds additional parity bits (r) to the sent information (k). The codeword can be calculated as $n = 2^r - 1$ . This means information data can be calculated by $k = 2^r - r - 1$ . Using a parity check matrix and a calculated syndrome, the scheme can self-detect and self-correct any SEE that occur during transmission.
Extended Hamming Code	Yes	Yes	Yes	No	No	No	The original scheme allows SECSSED, but with an addition of one bit, an extended Hamming version allows DECSSED.
Hadamard Code	Yes	Yes	Yes	Yes	No	No	Based on unique mathematical properties namely Hadamard matrixes, this linear code allows both DED and double error correction (DEC). This code was used in 1971 by NASA space probe Mariner 9, to send photos of Mars back to Earth (Malek n.d.).
Repetition Code	Yes	Yes	Yes	Yes	No	No	This code is one of the most basic codes as it simply resends a message several times. This result is low performance and transfer rates makes the code less than ideal.
Four Dimensional Parity Code	Yes	Yes	Yes	Yes	No	No	Also referred to as multidimensional parity-check code (MDPC), this code makes uses of multiple parity bits. This means the code basically arranges a message into a grid and then generating parity rows according to horizontal, vertical and cross diagonally. Ideally used for DDR RAM protection.
Golay Code	Yes	Yes	Yes	Yes	Yes	Yes	This code is a perfect linear error correction and makes use of a look-up table. This code has the following parameters [24, 12, 8] & [23, 12, 7].
BCH Code	Yes	Yes	Yes	Yes	Yes	Yes	This cyclic code is constructed using polynomials over a finite field. This code generally uses a linear-feedback shift register (LFSR) to encode the message block and uses syndromes polynomial to determine the error location during decoding. This results in Bose Chaudhuri Hocquenghem (BCH) codes being complex and difficult to implement while requiring significant processing time.
Reed Solomon Code	Yes	Yes	Yes	Yes	Yes	Yes	This non-binary cyclic code is based on univariate polynomials over finite fields. The error locator polynomial is then found using both the syndrome polynomial and Euclidian algorithm. Errors can then be located and corrected by applying the Chien Search Algorithm and Forney algorithm. This results in Reed Solomon (RS) codes being complex and difficult to implement, while requiring significant processing time.

Out of all the EDAC schemes mentioned in Table 1-3, Hamming and Reed Solomon Code are most commonly used in modern-day satellites (Bentoutou 2012).

An additional EDAC technique has to be mentioned as it is normally implemented together with a software EDAC such as Hamming code. This EDAC technique is referred to as Triple modular redundancy (TMR). TMR's definition is dependent on the section/hardware for which TMR is implemented. In simple terms, TMR consists of having three identical devices that perform the exact same operation but communicate through a voter. The voter compares all received information to ensure the information matches and that the right result is returned. Figure 1-6 shows a block diagram of a TMR based EDAC system which is implemented for RAM (Bentoutou 2012). TMR, however, is not the ideal solution as three times the necessary hardware is needed to perform a single operation. This ultimately adds to the satellite cost, complexity and computing time.

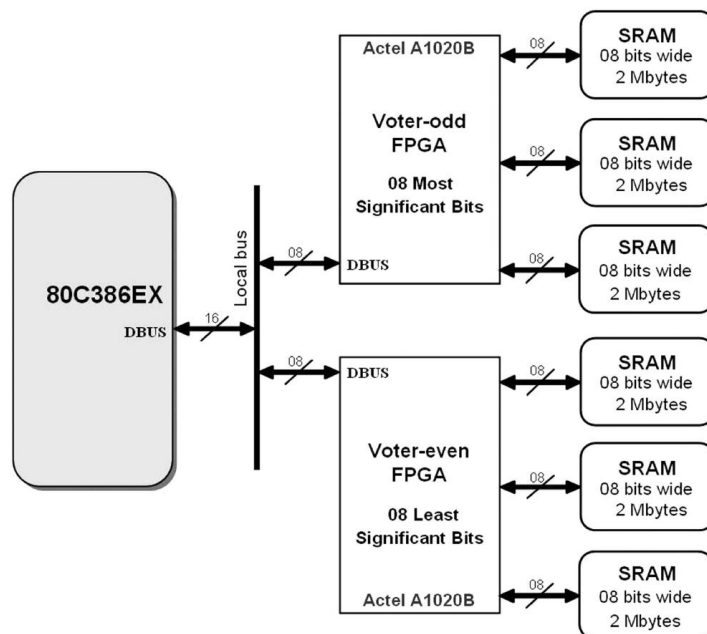


Figure 1-6: Block diagram of TMR-based EDAC (Bentoutou 2012)

### 1.3.7 Implementing EDAC systems

Before getting started it is important to know what EDAC systems are currently being implemented onboard nanosatellites and if additional hardware is needed in order to implement these EDAC schemes.

First, there are mainly two types of memory that need protection against upsets, namely program memory (SRAM) and Ramdisk. SRAM is faster than Ramdisk and is directly linked to the satellite's OBC, as it is typically used as the CPU cache. This implies that the integrity of the information stored within the SRAM is vital to the lifespan and health of the satellite. Ramdisk, on the other hand, refers to the memory that serves as a disk drive, mostly used to store image files and memory-intensive information.

Most commonly used EDAC schemes implemented onboard nanosatellites for SRAM protection is Hamming code and TMR. These are the most popular schemes because they are relatively easy to implement and have short encoding and decoding delays. RS codes, on the other hand, are more popular for large sets of information, as used in Ramdisk. For this reason, RS codes are known as block codes. RS is popular mainly due to its EDAC capabilities, as it is able to ensure MED-MEC.

From the literature review, it was clear that implementing EDAC systems on FPGAs is quite popular. In a paper by V. Tawar, a 4-dimensional parity EDAC scheme was designed, tested and synthesized on Xilinx FPGA Device XC3S500E-4FG320 (Tawar & Gupta 2015). A Horizontal Vertical Diagonal (HVD) EDAC was developed for Xilinx (Singh et al. 2013)(Road 2015). Other examples are Orthogonal codes written in Verilog using Altera Quartus-II software (Reshmi et al. 2015) and Hamming code implemented on FPGA using Verilog (Jindal 2006) and Xilinx Spartan-3 FPGA (Hosamani & Karne 2014).

From the finding of this search string, it is clear that EDAC schemes can be implemented and efficiently tested using FPGA design software. It can also be noted that the coding language needed is VHDL or Verilog.

### 1.3.8 Summary

To summarise, it is obvious that there is a need for designing effective and reliable EDAC systems for the nanosatellites. While researching this topic as a whole, information seemed hard to find, however, once broken up into small topics and research areas, information was found.

Space radiation has caused numerous mission failures. This was shown and proved in the section titled Space radiation. Through further research, it became apparent that some failures are owing to the SEU and MEU (see section 1.3.4 Glitches and upsets). These upsets are almost impossible to predict but there are certain areas of space where upsets are more frequent, for example, the Van Allen Radiation belts. Within the context of nanosatellites which are mostly low-earth orbiting (LEO) special attention needs to be paid to the SAA (see section 1.3.5 Geomagnetism).

It was found that there are a number of EDAC schemes and techniques currently used. Most commonly Hamming, RS codes and TMR (see section 1.3.6 EDAC schemes). It was also found that the most effective, non-evasive method of implementing an EDAC system would be to implement the system using an FPGA (see section 1.3.7 Implementing EDAC systems). Using the summarized information as a starting point this thesis will take a more detailed look at the design, development and implementation of an effective SoC based EDAC for nanosatellites.

From the literature survey, it is clear that there is a need for research in the area of EDACs. This field is new and constantly evolving as nanosatellites provide a platform from which the boundaries of space and technology are constantly being pushed. This thesis, upon completion, will contribute to further research and technological improvements.

## **1.4 Research questions**

Once this thesis has been concluded, the following questions would have been answered:

1. Do satellites in LEO need EDAC systems?
2. What degree of protection does ZA cube 2 need against space radiation?
3. What EDAC scheme provides the best protection against SEU?
4. Can a functional EDAC scheme be coded and tested within Matlab?
5. Can a functional EDAC scheme be coded and tested within VHDL?
6. Can the timing and resource usage of the designed EDAC system be evaluated?
7. To what degree of confidence can the designed EDAC system be tested and functionality be proven?

## **1.5 Objectives of the research**

### **1.5.1 Primary objective**

To develop and test an efficient EDAC system design for SoC implementation onboard a nanosatellite.

### **1.5.2 Secondary objectives:**

1. The EDAC system should detect and correct more than 90% of all bit errors.
2. The EDAC system should provide protection (both read and write) in under 8 ns.
3. The EDAC system should be capable of single error correction and double error detection.

## **1.6 Delineation of the research**

By the thesis title “A system on chip based error detection and correction for nanosatellites” it is clear this thesis is aimed at nanosatellites. This limits the following:

- The power consumption of the proposed system.
- Computational power available.
- The number of memory banks to protect.

The title also implies the following:

- This thesis is focused on low-earth orbiting (LEO) satellites (300 km and 800 km above the earth's surface).



## 1.7 The significance of the research

This section will indicate the importance and significance of the research conducted in this thesis. This will be done by establishing why this thesis is important, whom and what industries will benefit, as well as the effect this thesis could have on space exploration and science.

### 1.7.1 Benefits

The benefits of a fully functional prototype are endless. Once incorporated into a functional nanosatellite, this EDAC system will ensure data integrity and reliability. This will ultimately improve the lifespan and capabilities of a satellite.

For example [Benefit (*Affected sectors/studies*): Explanation]:

- Earth monitoring (*Environmental studies*): Satellites that previously switched off while passing through the SAA can continue normal operations.
- Reliable data (*Exploring the unknown*): Readings from sensors and the results of experiments conducted on board satellites will be more accurate and trustworthy.
- Satellite lifespan (*All space missions*): By protecting the OBC, CPU and cache, the satellite's health information will be used to protect against SEE. This will ultimately decrease the chance of major system failures.

### 1.7.2 Target group

This thesis is aimed at the nanosatellite industry. The group targeted are designers and engineers who would like to ensure data integrity and reliability during space missions. Another targeted group would be satellites that need to be turned on and functional at all times, for example, when passing over the South Atlantic or during solar flares.

## 1.8 Thesis layout

A brief overview of each chapter will be given in the descriptions below, to make navigating through this thesis easier and more efficient.

**Chapter 2: Space Radiation:** In this chapter, the ZA-Cube 2 nanosatellite launched by French South African Institute of Technology (FSATI) is used as a case study. A detailed look at the space environment which ZA-Cube 2 will be exposed to is simulated using Omere Software. The results obtained will be discussed and concluded.

**Chapter 3: Error correcting code:** This chapter introduces error control codes, giving the reader some perspective and background on the topic. A number of error detection and correction codes such as Parity, Hamming, Golay, BCH and Reed Solomon Codes are mentioned and discussed.

**Chapter 4: Hamming code:** This code is identified as the foundation code of this thesis. An overview, as well as a focused study on the encoding and decoding aspect of this code, is presented and explained. Different Hamming variation are also mentioned and discussed.

**Chapter 5: Simulation and Testing results of Hamming code in VHDL:** Using the simulation tool like ModelSim, together with resource reports and timing analysis tools, the performance of the Hamming code will be thoroughly discussed and concluded. The hardware implementation of the Hamming code in Quartus Prime will be introduced and the optimization steps will be shown.

**Chapter 6: Conclusion:** All important outcomes and findings made during the different chapters of this thesis will be summarised and concluded. Based on the knowledge gained during the completion of this thesis future recommendations will be made.

## CHAPTER 2. SPACE RADIATION

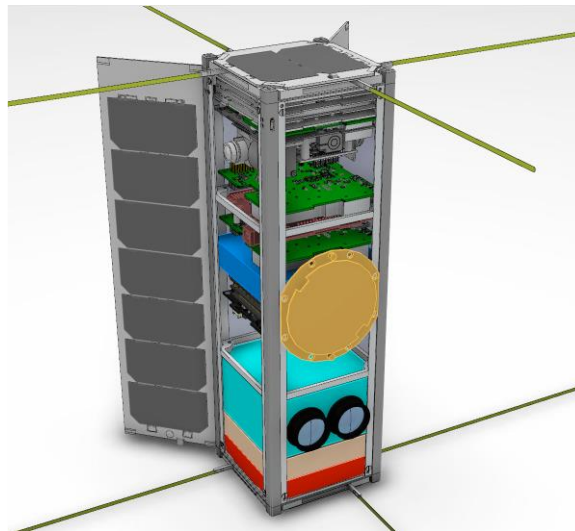
### 2.1 Introduction

In order to understand space radiation, a study was conducted using the orbital parameters of nanosatellite ZA Cube 2. The ZA Cube missions are a series of nanosatellite missions being developed by FSATI in collaboration with CPUT. The aim of these missions is to develop South Africa's space industry and space experience. The first satellite of this program was launched in November of 2013 and is currently still active. The name of this initial nanosatellite is the ZACube-1 (CPUT 2017).

The successor to ZACube-1 will be ZA Cube 2 (Figure 2-1). The set launch date for ZACube-2 was May 2018. ZA Cube 2's primary focus is on maritime domain awareness (MDA) applications (Villiers & Zyl n.d.). This satellite has the following objectives:

- Technology demonstration of AIS/VDE message reception using the primary payload
- Technology demonstration of a medium resolution imager payload
- Human capacity development (HCD)
- Flight heritage for F'SATI/CPUT hardware and also hardware from technology partner ESL

These mission objectives will be achieved using the VHF AIS/VDE receiver and the Medium resolution CMOS imager acting as the primary and secondary payloads (CPUT 2016).



**Figure 2-1: ZACube-2 Conceptual Layout (Villiers & Zyl n.d.)**

This will be a radiation study, using OMERE and TRIM software. This study will consider the three main sources of space radiation, namely: earth radiation belts (ERB), galactic cosmic radiation (GCR), and solar particle events (SPE). This study is important to prevent SEE and to assist in the correct selection of error control codes, components and shielding. Using the proposed orbital parameters set by FSATI this document will simulate and explore the environment in which the ZACube-2 will be orbiting. From the results obtained in this document, conclusions will be formulated and recommendations made.

## 2.2 ZA-CUBE 2 orbital parameters

Using the information provided by FSATI, the following orbital parameters were established and entered into OMERE software (Figure 2-2).

Orbital parameters:

- Date of Launch: **01/05/2018**  
Time: **12:00 - Midday** (Assumption)
- Inclination: **98deg** (inclination for SSO orbit)
- Apogee and Perigee: **550km** (orbital altitude under 600km (Villiers & Zyl n.d.) )
- Orbit type: **Circular orbit**
- Segment Duration: **4 years** (life spanned 3 to 5 years)

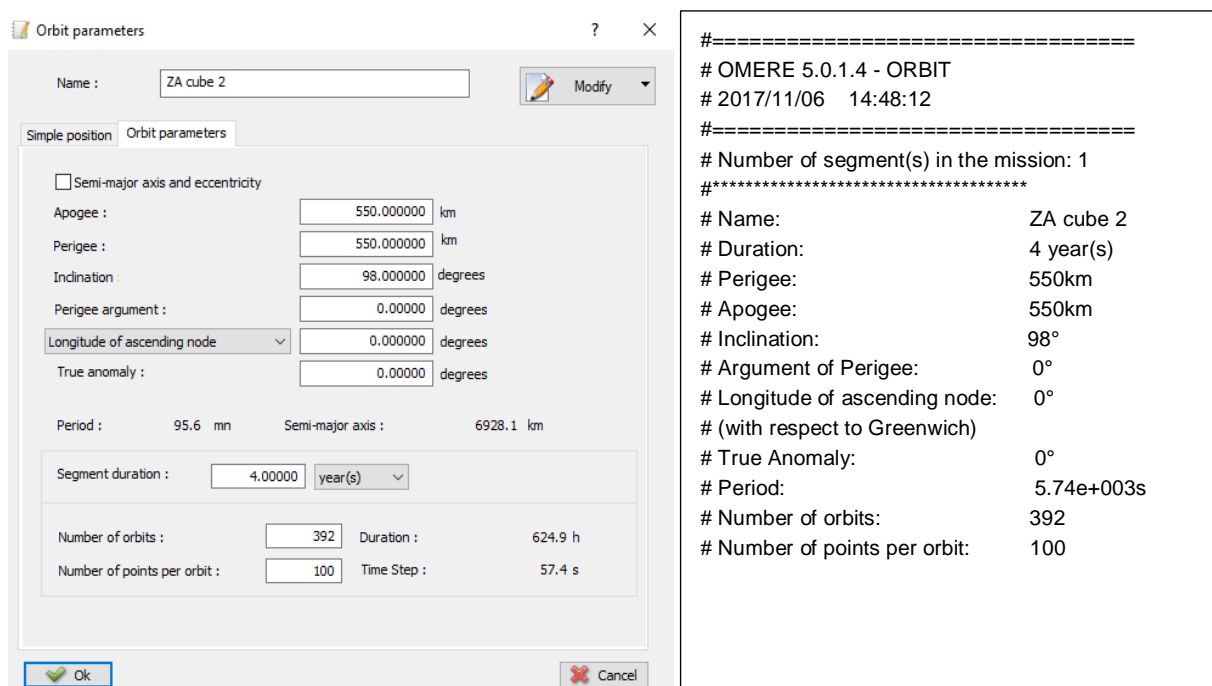


Figure 2-2: Omere orbital parameters: initialisation (left) and output file (right)

## 2.3 Understanding radiation

With the continuous improvements and developments of technology, we are becoming more and more dependent on technology, especially space technology. It is a known fact that electronic systems are affected by radiation. This could cause data to be unreliable and can result in major failures. It is necessary for all engineers in the space industry to have some basic knowledge of the effects radiation can cause, namely, SEE.

### 2.3.1 Single event effects

SEE is electrical noise induced by the natural space environment (high energy ionising particles). This “noise” results in data corruption, transient disturbances and high current conditions which could lead to unwanted functional interruption or in the worst case, catastrophic failures. SEE are caused mainly by space radiation or energetic particles.

Main causes of Single Event Effects are:

- Galactic cosmic rays
- Cosmic solar particles (heavily influenced by solar flares)
- Trapped protons in radiation belts

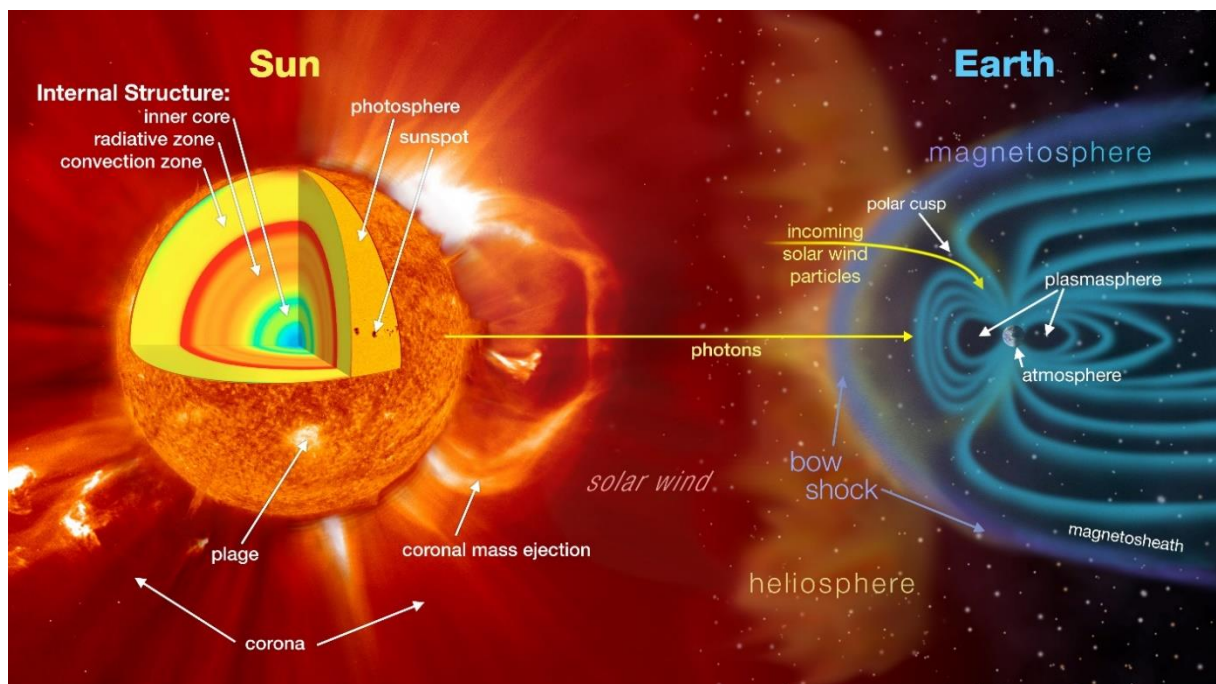


Figure 2-3: Heliophysics and Space Weather (Bensusen et al. 2013)

These SEE effect many types of devices and technologies. Single Event Effects includes Single Event Upset (SEU), Single Event Gate Rupture (SEGR) and others. Vulnerability to SEE has increased drastically, as devices like IC's operating speeds and density increases. When radiation strikes a device, it results in the collection of charges and hence changes the electrical performance. These SEE largely impact the reliability of electronic circuits used in the space environment.

The section to follow, with the help of Figure 2-4, shows how particles can cause SEE within devices:

**Cosmic rays:** Heavy ions cause direct ionization which results in SEE. When an ion particle travels through a device and deposits sufficient charge, an event such as a memory bit flip or transient may occur.

**Radiation belts and solar flares:** Protons are capable of causing a nuclear reaction near a sensitive node, thus creating an indirect ionization effect, potentially causing an SEE. They could also cause direct ionization in highly sensitive devices.

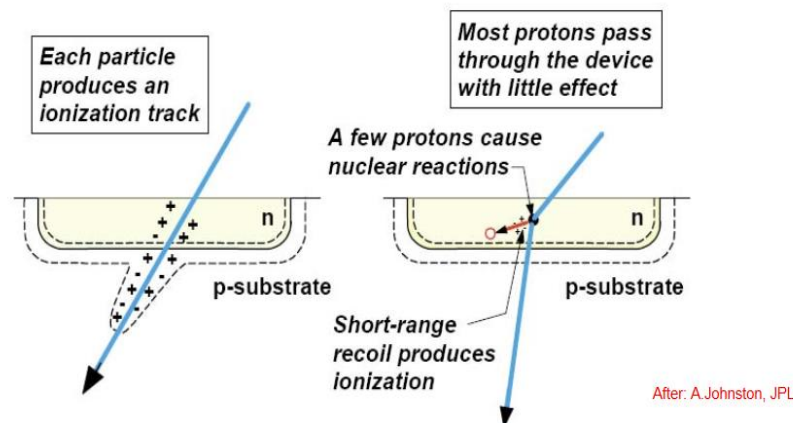


Figure 2-4: Effects of heavy ions (left) & protons (right) - (Halbert 2006)

### 2.3.2 Effects of radiation in terms of SEE

SEE are usually split into two categories. These categories are listed and shown below:

**Non-destructive:** Events which momentarily or permanently change the state of a device or cell/node without affecting its functionality. For example:

- Single Event Upset (SEU) – Bit flips in memory cell and registers
- Single Event Functional Interrupt (SEFI) – Temporal device functionality loss, recovered by a power cycle
- Single Event Transient (SET) - momentary variation in current or voltage to or from a device

**Destructive:** Events which interrupt device function and permanently damage the device without external interaction. For example:

- Single Event Latch-up (SEL) - Can cause circuit lockup, recovered by a power cycle
- Single Event Burnout (SEB) - Localized current in the body of a device, turning on parasitic bipolar transistor
- Single Event Gate Rupture (SEGR) - Dependent on the angle of incidence and on the electric field in the gate oxide
- Single Event Hard Errors (SHE) – A rare case which renders a single cell unable to change state

For further insight into types of SEE, their effects and the components affected, refer to the ECSS-E-ST-10-12C document produced by ECSS. Table 2-1 identifies the components and the possible effect SEE could cause. For a more extensive table see section 9.3 of the ECSS E-ST-10-12C document. (ECSS 2008)

**Table 2-1: Possible SEE as a function of component technology and family**

Component type	Technology	Family	Function	SEL	SESB	SEGR	SEB	SEU	MCU/SMU	SEDR	SEHE	SEFI	SET	SED	
Transistors	Power MOS					X	X								
ICs	CMOS or BiCMOS or SOI	Digital	SRAM	X*				X	X		X				
			DRAM/SDRAM	X*	X			X	X		X	X			
			FPGA	X*				X		X		X			X
			EEPROM/Flash EEPROM	X*							X		X		X
			μP/μcontroller	X				X				X	X		X
		Mixed	ADC	X*				X				X	X	X	
		Signal	DAC	X*				X				X	X	X	
	Linear		X*							X			X		
	Bipolar	Digital						X					X		
		Linear						X					X		

### 2.3.3 Mitigation techniques

Mitigation techniques are methods used to reduce the severity and seriousness of SEE. There are a number of different ways to mitigate SEE.

The methods are categorised in the following levels:

- System level: For example, TMR.
- Circuit level: For example, radiation hardened devices protecting FFs and SRAM cells.
- Software level: For example, 3 processing + voting.
- Chip-level: For example, EDAC codes.

From a mechanical point of view, it is possible to prevent SEE. This is done by using materials that have a natural resistance against radiation and by using purified fabrication materials. Large satellites make use of radiation hardened technologies and devices which are resistant to damage and malfunctions caused by ionizing radiation. These devices and components are called RadTolerant. However, these devices are not the ideal option for nanosatellites due to cost and availability in third world countries. A more practical and logical approach to prevent SEE is to exploit the satellites' PCB and component layout. For example, placing sensitive components behind the battery banks which are able to block most radiation.

Prevention is one approach to handling SEE, but error detection and correction techniques can prove just as effective when implemented correctly. Redundancy on a system, software and at chip level can be highly effective. Error Detection and Correction Codes (EDAC) such as Hamming and Reed Solomon (RS) codes have been used extensively in telecommunication and during deep space missions. EDAC codes are preferred, as they are implemented in software and do not require too much additional power. These types of approaches are more feasible and suited for the nanosatellite industry.



## 2.4 Earth radiation belts

Earth radiation belts (ERB) are belts formed in space around the earth due to the earth's magnetic field. These belts contain and trap energised particles. The most predominant ERB are the Van Allen radiation belts.

### 2.4.1 Magnetic field

Using the flux mapping options it is possible to plot particle fluxes (AE8 and AP8 modes) and magnetic fields in 2D or 3D views. These views and plots are effective in helping establish where magnetic fields are strongest and where protons and electrons are mostly concentrated.

The magnetic fields displayed in Figure 2-5, range from 19.5 KnT to 48.7 KnT. This plot was set up to simulate the magnetic fields at the altitude of 550km (+/- the altitude of ZA cube 2) for the year 2018. The magnetic field selected for the simulation is Jensen Cain. From Figure 2-5 it is clear that the magnetic fields are strongest around the poles and weakest around the equator, especially the south Atlantic anomaly (SAA) area.

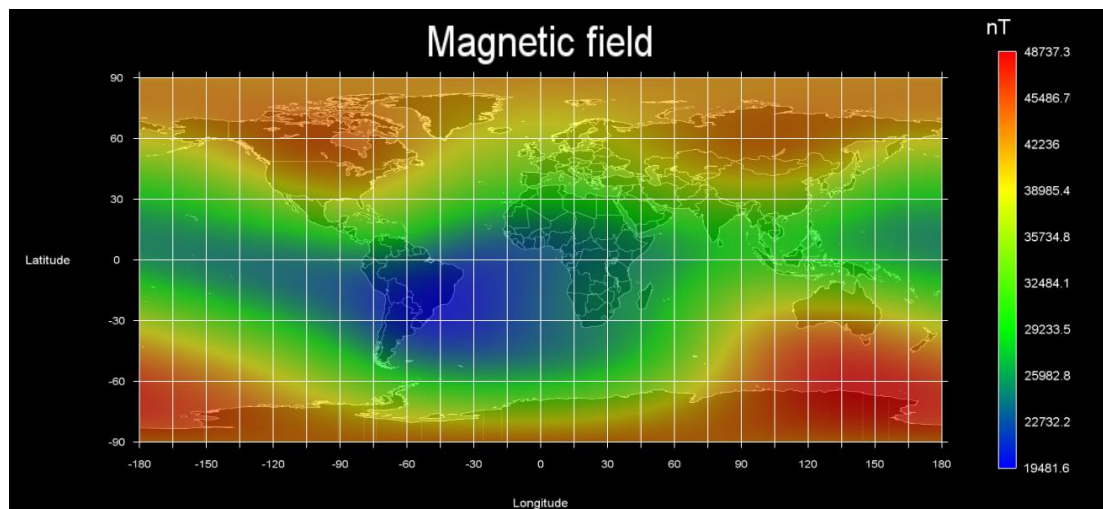
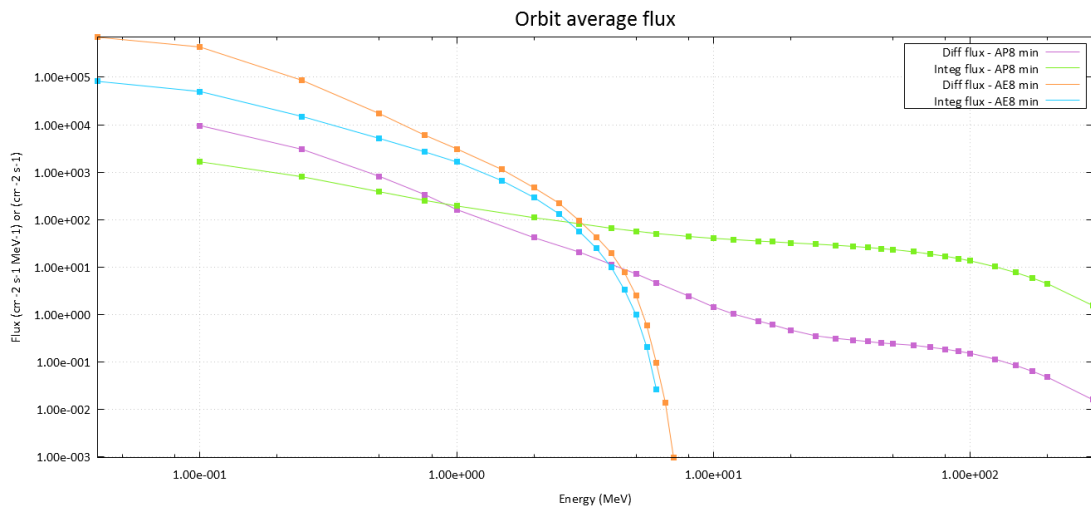


Figure 2-5: Magnetic field (Jensen Cain) at 550km in 2018

## 2.4.2 Trapped particles

ZA cube 2 will travel through the Van Allen radiation belts (Zell 2015)(Zell 2013). For this reason, AP8 and AE8 models were selected when simulating the trapped particles. NASA's AP8 and AE8 models have been the European standard since the seventies (D.Heynderickx 2002) and allow for max and min models to be simulated. Mean flux was selected as an output for both the Proton and Electron models. The mean flux was selected as the flux spectrum is first calculated at each orbit point and then averaged.

Figure 2-6 shows the average integral and differential fluxes of the trapped particles within the defined orbit. From the graph, it is clear that the maximum integral flux for trapped electrons is at  $8.15 \times 10^4 \text{ cm}^{-2} \cdot \text{s}^{-1}$  flux at an energy of around 40 KeV, this decays to the minimum flux of  $2.61 \times 10^{-2} \text{ cm}^{-2} \cdot \text{s}^{-1}$  with a maximum of 6 MeV energy. The maximum integral flux for trapped protons, on the other hand, is  $1.65 \times 10^3 \text{ cm}^{-2} \cdot \text{s}^{-1}$  flux at an energy of around 100 KeV, this decays to the minimum flux of  $1.55 \text{ cm}^{-2} \cdot \text{s}^{-1}$  with a maximum of 300 MeV.



**Figure 2-6: Orbital average integral and differential fluxes of trapped particles**

### 2.4.3 Trapped electrons

Figure 2-7 and Figure 2-8 show the maximum trapped electrons at an altitude of 550km. Figure 2-7 (differential) and Figure 2-8 (integral) show that the trapped electrons of around  $1 \text{ MeV.cm}^2.s$  to  $1 \times 10^5 \text{ MeV.cm}^2.s$  for integral and differential. The trapped electrons are concentrated in a channel just below the North Pole and above the South Pole, as well as the SAA region.

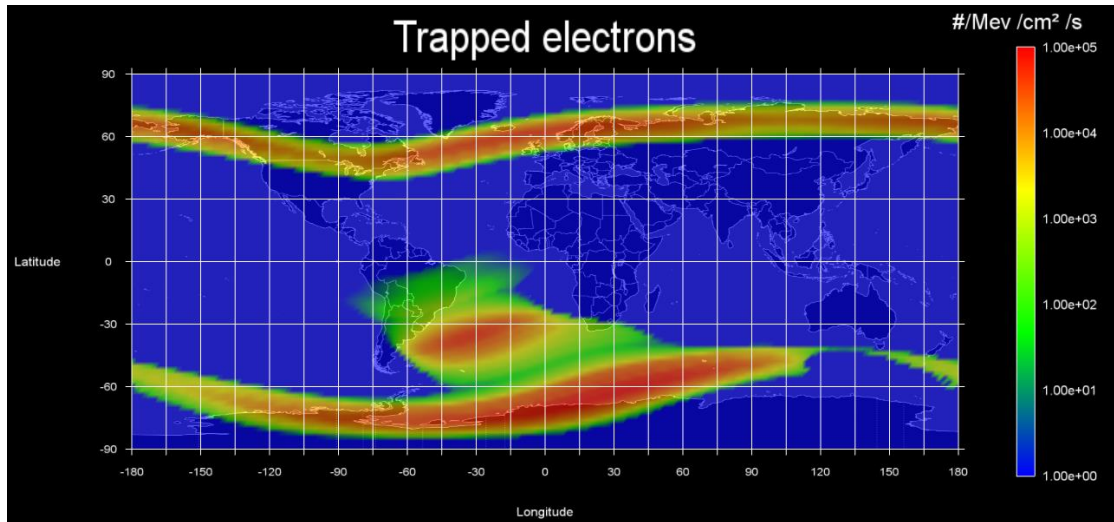


Figure 2-7: Maximum trapped electrons (differential)

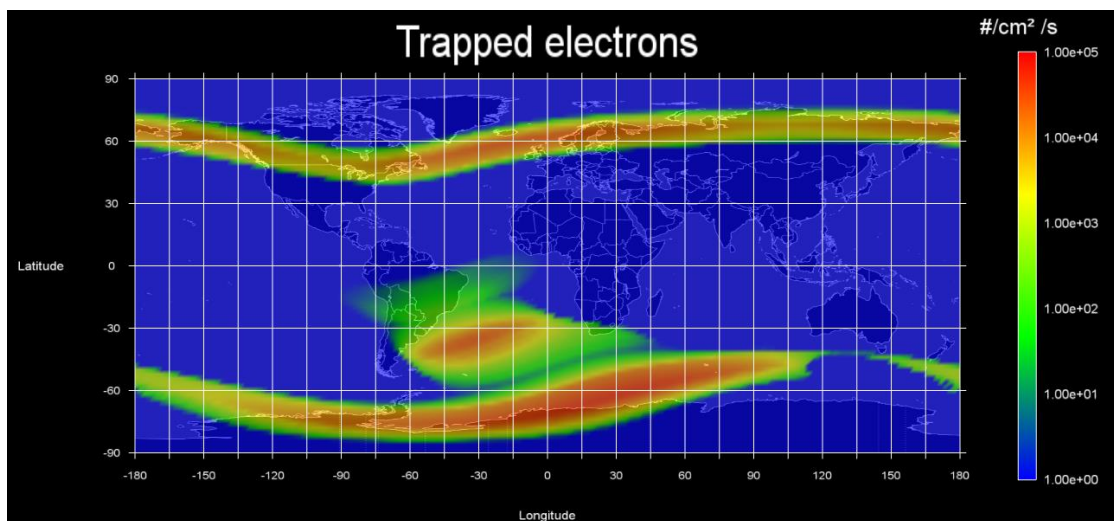


Figure 2-8: Maximum trapped electrons (integral)

#### 2.4.4 Trapped protons

Figure 2-9 and Figure 2-10 show the maximum trapped protons at an altitude of 550km. Figure 2-9 (differential) and Figure 2-10 (integral) show that the trapped protons of around  $1 \text{ MeV.cm}^2.s$  to  $1 \times 10^5 \text{ MeV.cm}^2.s$  for integral and differential. These protons are concentrated intensively around the SAA. It is important to note regions of concentrated protons and electrons as they can cause SEE and in some cases the failure of a satellite mission.

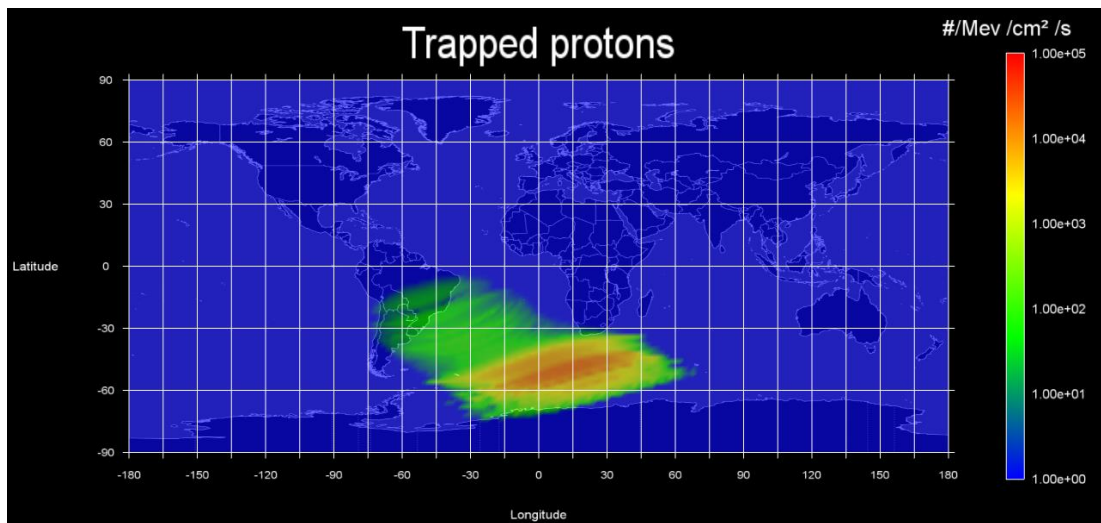


Figure 2-9: Maximum trapped protons (differential)

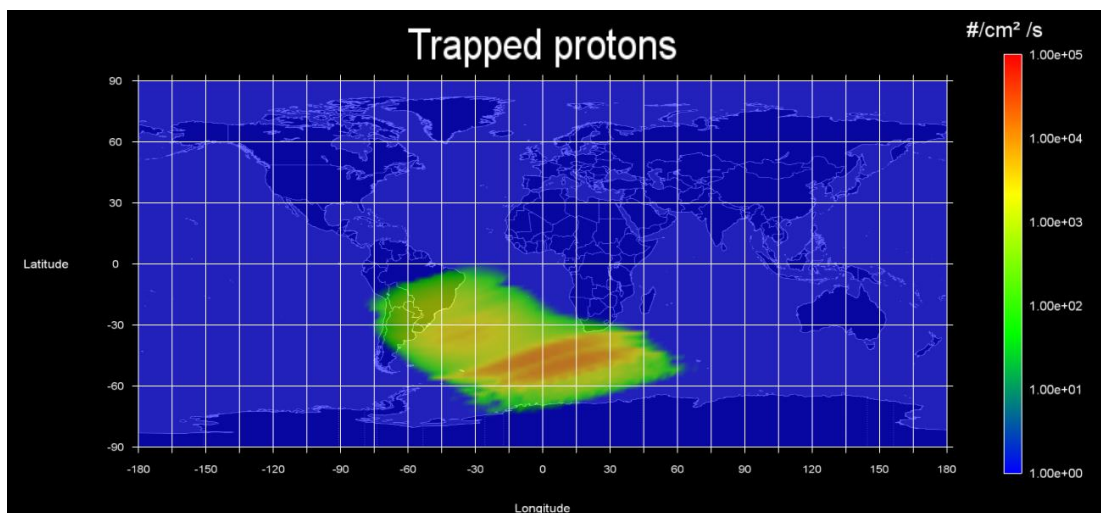


Figure 2-10: Maximum trapped protons (integral)



## 2.4.5 Orbital trapped particles

The figures to follow shows the differential intensity of the electrons (Figure 2-11) and protons (Figure 2-12) during a specific LEO. Their intensities reach about  $10 \text{ MeV.cm}^2$  during the high-intensity regions, such as the SAA, and are slight in Figure 2-11 while passing the North and South Pole trapped electron channels. Satellites travelling in this orbit should be able to withstand trapped electrons of around  $10 \text{ MeV.cm}^2$  or new orbital parameters need to be implemented.

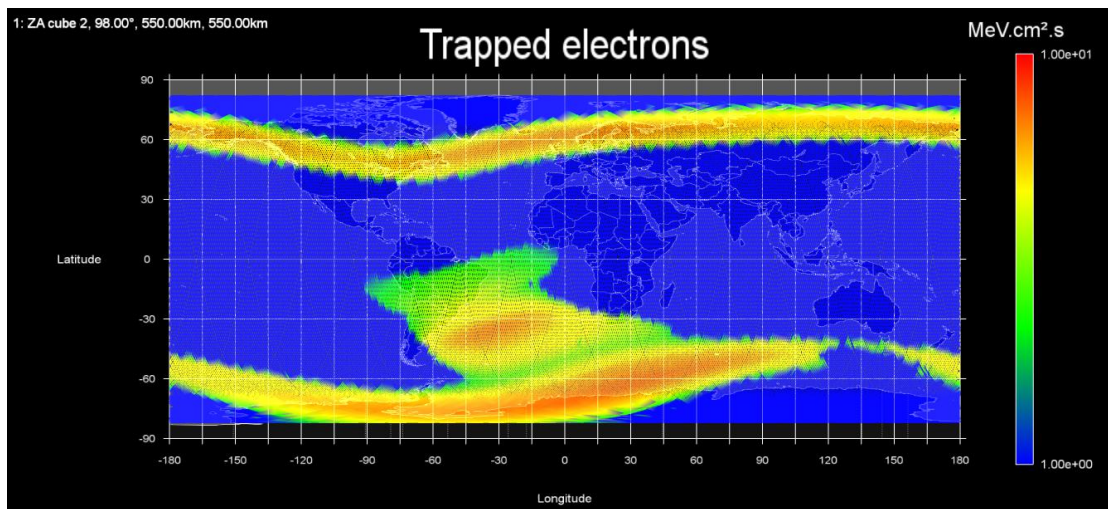


Figure 2-11: Orbital minimum trapped electrons – AE8 – Jensen Cain – Differential

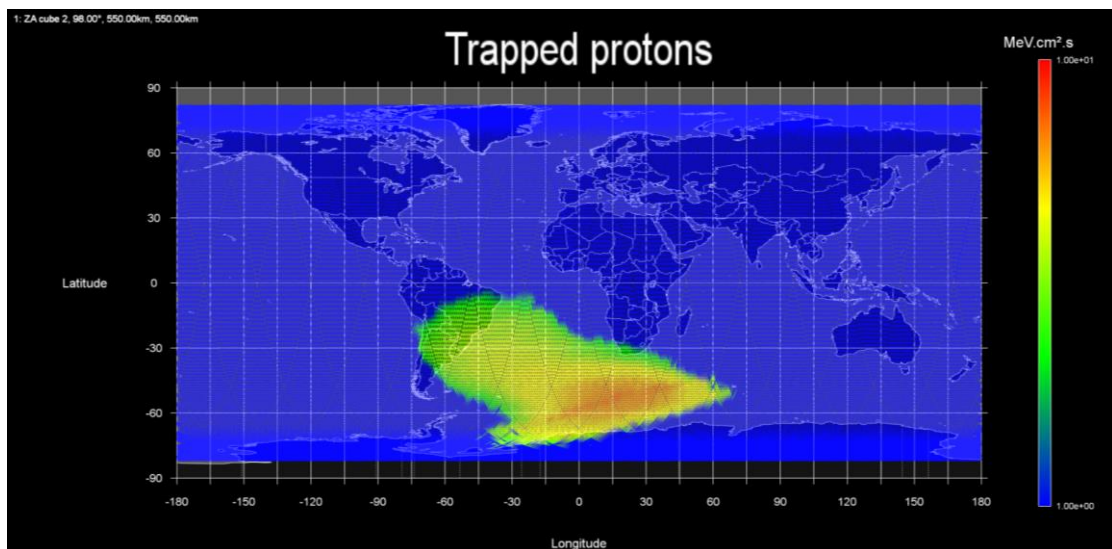


Figure 2-12: Orbital minimum trapped protons – AP8 – Jensen Cain – Differential

## 2.5 Solar cosmic radiation

Solar cosmic radiation (SCR) is one of the main sources of radiation in space. Using OMERE this radiation source is simulated on fluxes and fluences vs energy graphs.

### 2.5.1 Protons

The proton model used is ESP, this model was selected to comply with the European Cooperation for Space Standardization (ECSS). The ECSS ensures uniformity and standardizations. For this simulation, a confidence level of 90% and a solar active period of four years was used.

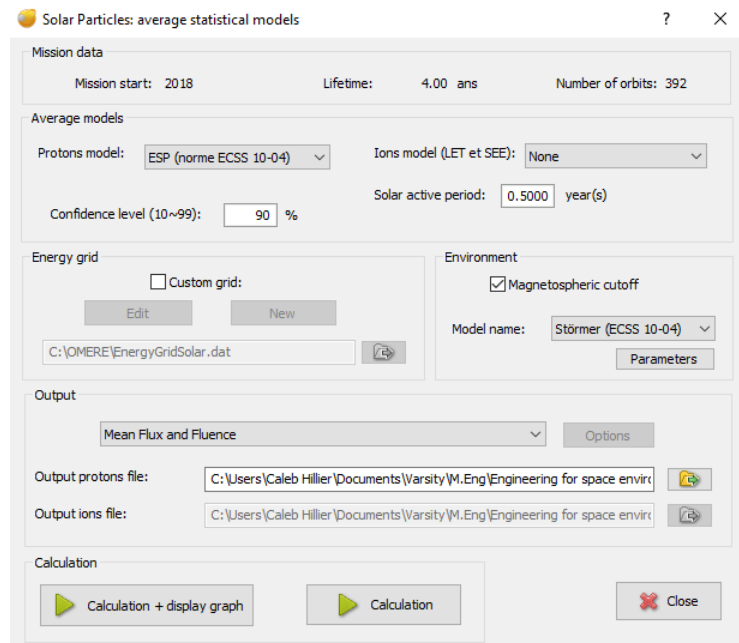


Figure 2-13: Solar Particle (Proton) – Setup

Using the plotted graph in Figure 2-14, the solar protons that ZACube-2 will be exposed to can be seen. The integral flux maximum is  $1.65 \times 10^2$  protons.cm<sup>-2</sup>.s<sup>-1</sup> at an energy level of 1 MeV and a minimum flux of  $4.66 \times 10^{-3}$  protons.cm<sup>-2</sup>.s<sup>-1</sup> at an energy level of  $3 \times 10^2$  MeV.

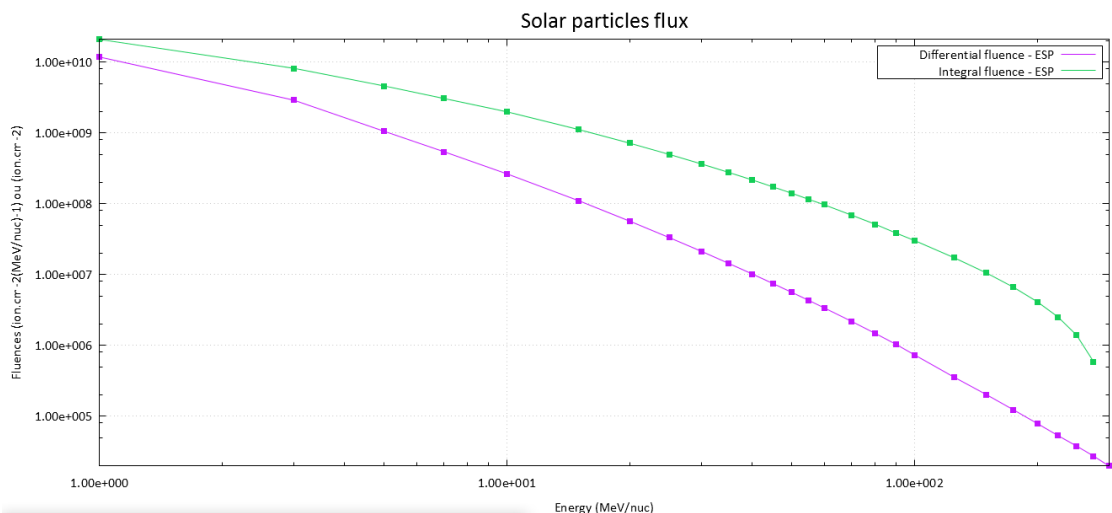


Figure 2-14: Integral and differential fluences of solar protons

## 2.5.2 Ions

Using the OMERE software it is possible to plot specific solar ions according to selected elements. As shown in Figure 2-15, elements Z=2 (He/Helium) to Z=17 (Cl/Chlorine) are taken into account during a solar active period of 4 years. Figure 2-16 shows the integral and differential flux of the selected elements vs the energy range.

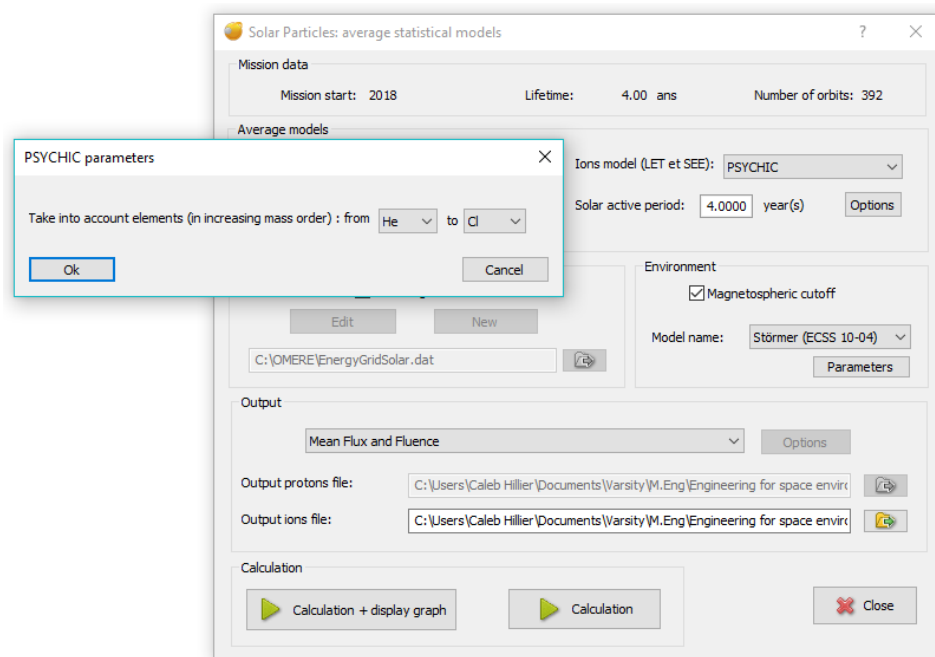


Figure 2-15: Solar Particle (Ion) Z = 2 to Z = 17 – Setup

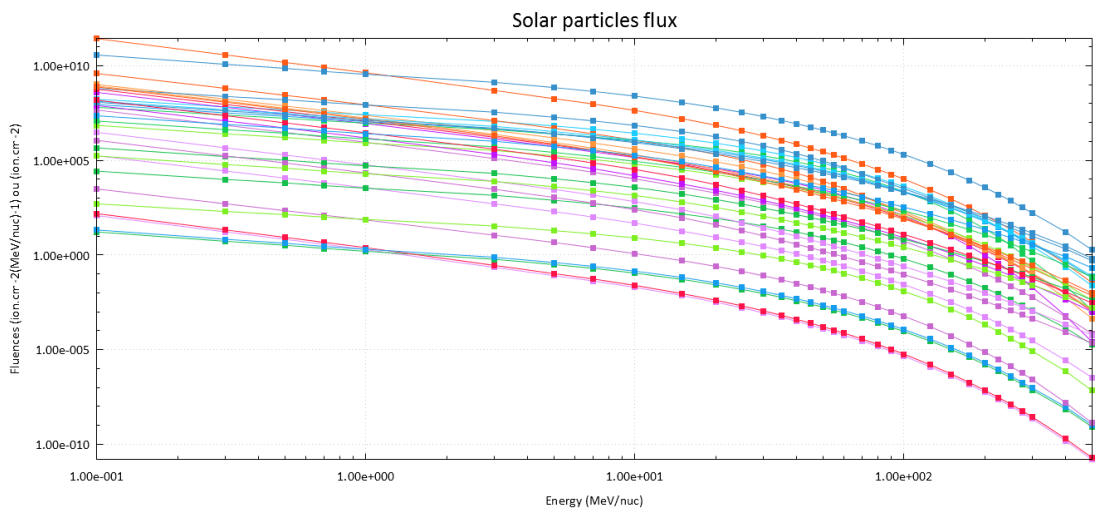


Figure 2-16: Integral and differential fluence of solar ions Z = 2 (He) to Z = 17 (Cl)

## 2.6 Galactic cosmic radiation

Galactic cosmic radiation (GCR) is one of the main sources of space radiation. Using OMERE, the heavy ions that ZACube-2 will be exposed to can be found using the generated graphs shown below. The figures show the energy verse flux graphs for galactic cosmic radiation during a solar minimum. It can be noticed that the flux level decreases as the atomic numbers of the GCR heavy ions increase from  $Z=1$  to  $Z=17$ . This observation can be said for both integral (Figure 2-17) and differential flux (Figure 2-18). The differential flux tends to peak at  $5.78 \times 10^{-4} \text{ ion.cm}^{-2}.\text{s}^{-1} \cdot (\text{MeV/nuc})^{-1}$  this is a well-known feature and should be noted for shielding purposes.

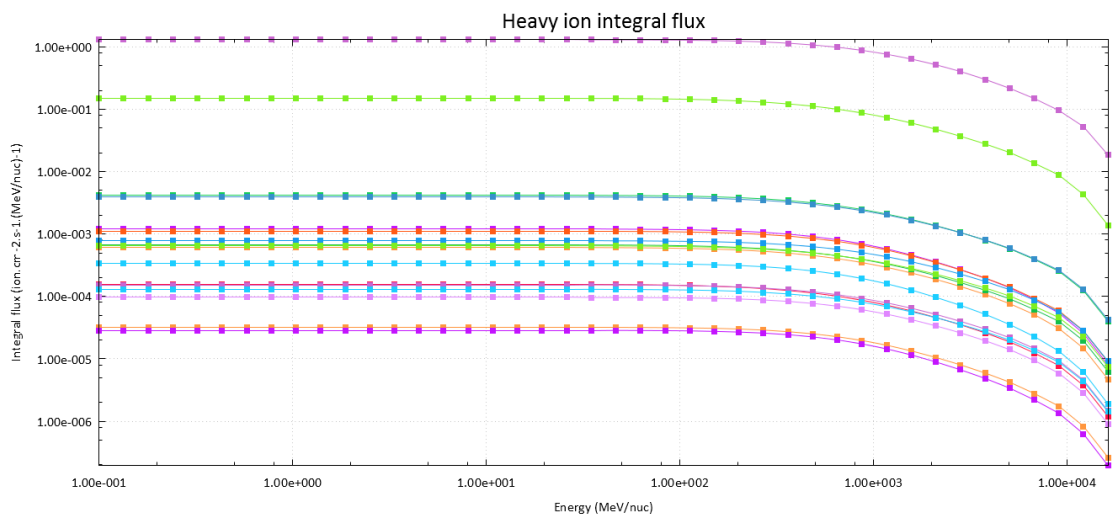


Figure 2-17: Heavy ion integral ( $Z = 1$  to  $Z=17$ )

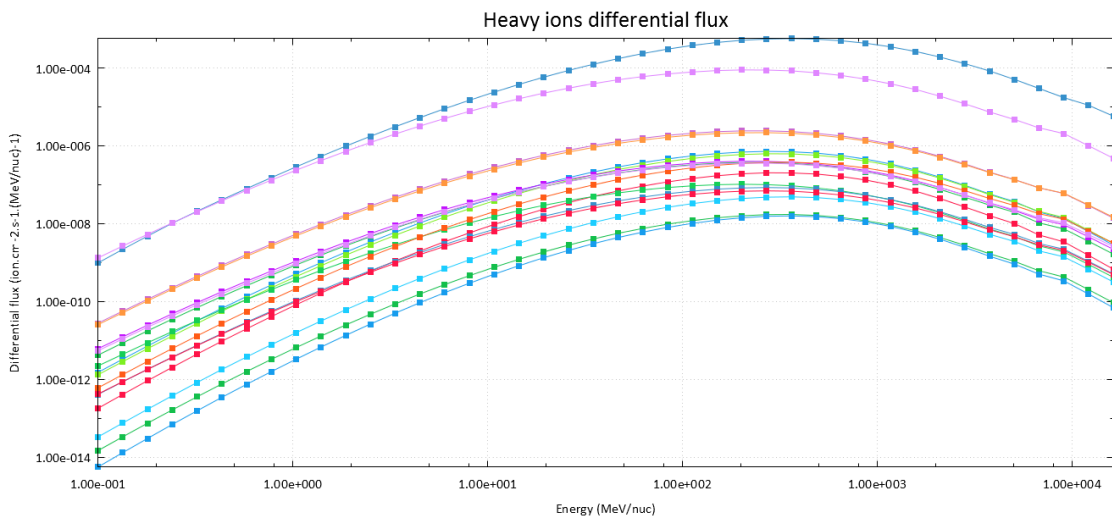


Figure 2-18: Heavy ion differential ( $Z = 1$  to  $Z=17$ )



## 2.7 Shielding

Shielding is used to protect satellites against energized protons and heavy ions in space. The type of material and coatings thickness influences how radiation tolerant a satellite is.

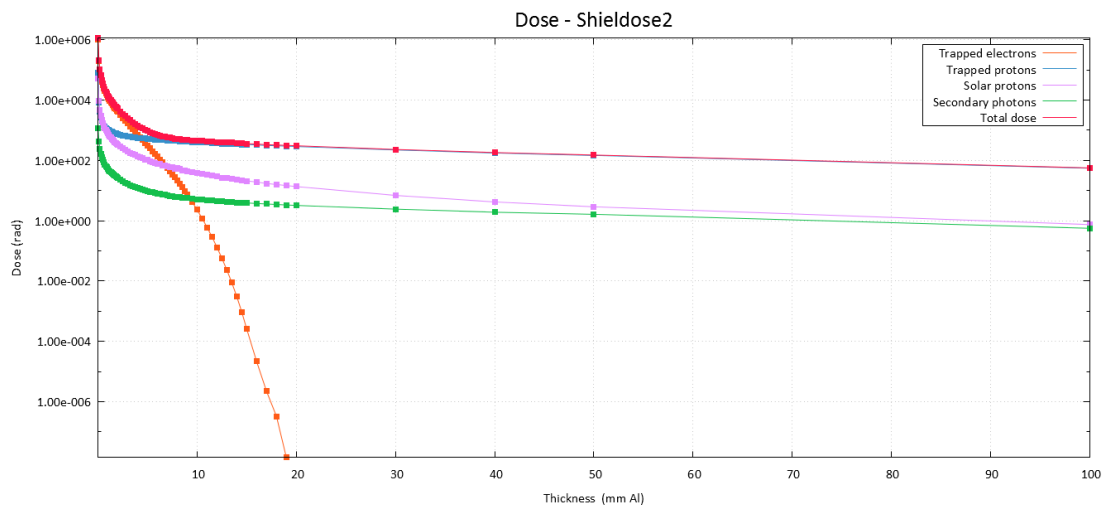
### 2.7.1 Dose

The dose is defined as the total amount of ionizing radiation absorbed by a material. In this case study, the ZA cube 2 nanosatellite structure consists mainly of aluminium. Aluminium was used to reduce weight and not for its shielding properties, however the aluminium structure does provide a minimal amount of protection.

The graph below plots the different sources of radiation against the thickness of aluminium. Typically 2 mm Al is used to shield a nanosatellite, however, this thickness should be carefully chosen by considering graphs (like Figure 2-19), cost and weight.

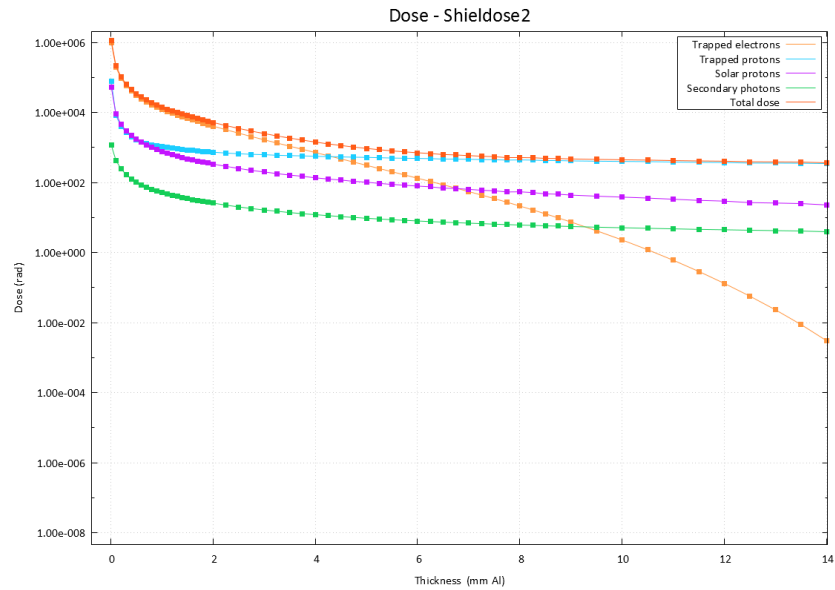
From the graph shown in Figure 2-19, the following can be noted:

- 20mm Al will block majority of the radiation, but will add too much weight.
- 10mm Al will block a large portion of radiation, but will still add too much weight.
- 5mm Al will block most of the high-level radiation but will still add too much weight.
- 2mm Al only blocks the high levels of radiation (Figure 2-20). This thickness will be the best trade-off when looking at weight and cost vs radiation.



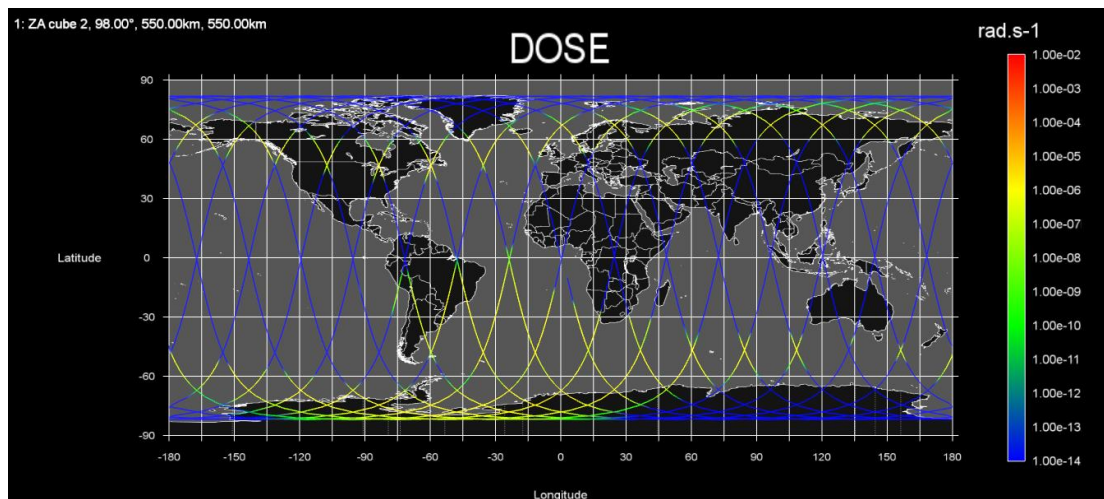
**Figure 2-19: Dose (rad) vs thickness**

It is important to note that 2mm Al is used in most of the simulations as this is the standard thickness of ZA cube 2's casing. Aluminium is not used for its radiation prevention properties but because it is light weighted and cheap.



**Figure 2-20: Dose (rad) vs thickness (zoom in)**

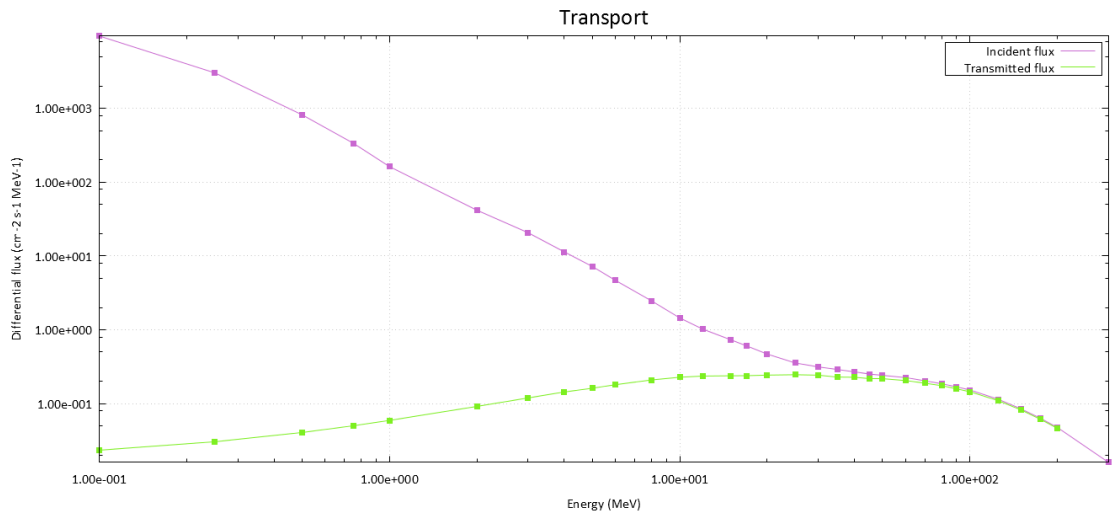
Figure 2-21 shows the orbital dosage graph for a nanosatellite with a 2mm Al structure. Orbital rotations plotted were kept to a minimum in order to reduce the computational power needed to run the simulation. When comparing these results to the flux mapping graphs it is clear that the satellite structure will provide some shielding when travelling through the high radiation regions of space, like the SAA. From this, we can conclude that 2mm Al is able to provide some radiation protection but not enough.



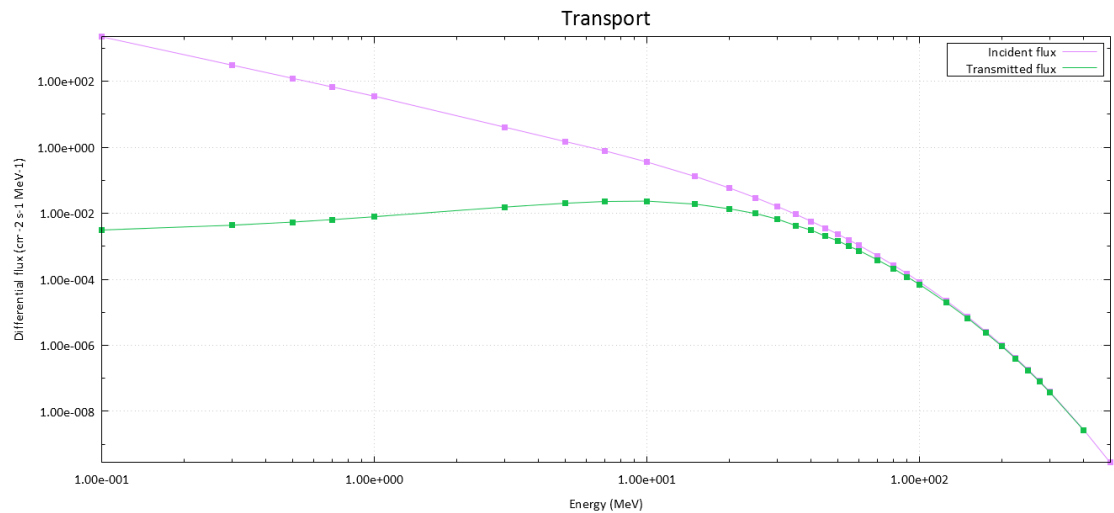
**Figure 2-21: Orbital dosage graph for 2mm Aluminium shield**

## 2.7.2 Transport

The figures to follow compare the incident flux and transmitted flux for trapped protons (Figure 2-22) and solar peak and mean heavy ions for hydrogen (Figure 2-23). The transmitted flux is the resulting flux after implementing 2mm Al shielding. Figure 2-22 shows 2mm Al has a very high absorption of the low energized proton but basically nothing for energies above 20 MeV. From Figure 2-23 it is also clear that 2mm Al shielding will help protect the satellite against heavy ions of energies below 30 MeV but is useless for energies above 30 MeV.



**Figure 2-22: Fluxes of incident and transmitted trapped protons in ERBs**



**Figure 2-23: Fluxes of incident and transmitted solar peak and mean heavy ions**

### 2.7.3 Linear energy transfer

Figure 2-24 is the calculated LET spectra graph. The graphs to follow shows the flux vs Liner-energy-transfer (LET). This integral LET spectrum was plotted specifically for ZACube-2 orbital parameters of 98° degrees inclination with 2mm Al shield.

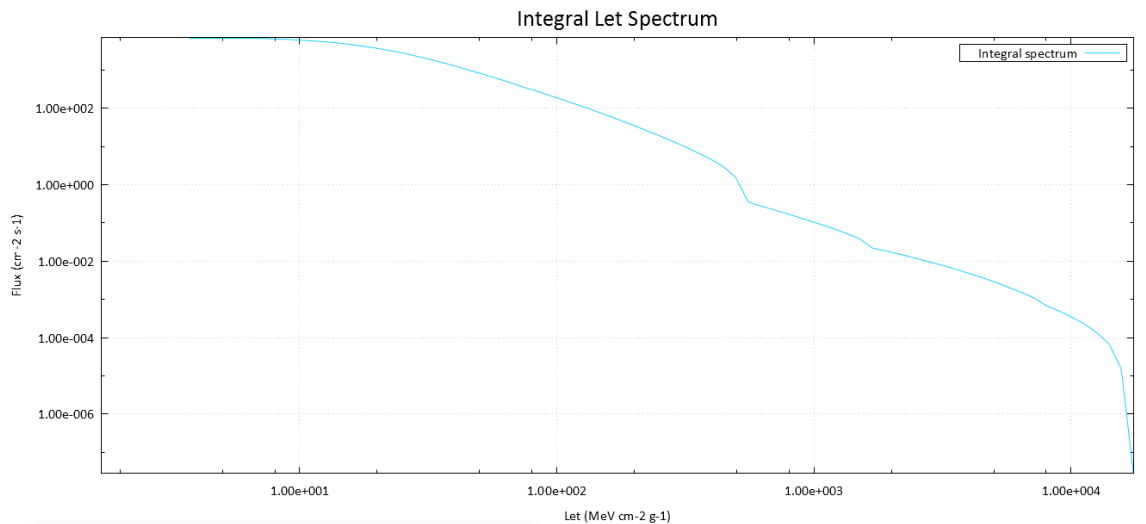


Figure 2-24: Integral LET spectrum diagram for 98° and 2mm Al shield

### 2.7.4 TRIM – The transport of ions into a matter

TRIM was created by James F. Ziegler and was described by him in the following manner (James Ziegler 2013):

“TRIM (the Transport of Ions in Matter) is the most comprehensive program included. TRIM will accept complex targets made of compound materials with up to eight layers, each of different materials. It will calculate both the final 3D distribution of the ions and also all kinetic phenomena associated with the ion's energy loss: target damage, sputtering, ionization, and phonon production. All target atom cascades in the target are followed in detail. The programs are made so they can be interrupted at any time and then resumed later. Plots of the calculation may be saved, and displayed when needed.”

Using TRIM, protons of different energies will be aimed at Aluminium and other materials. The results will help establish how effective the material is at shielding the satellite. These results will be shown in the sections to follow.

### 2.7.4.1 TRIM Set-up

Using the set-up tables shown by Figure 2-25, different aspects to test can be selected. In this document the following scenario was simulated and tested:

- Protons (+ hydrogen) of 10 MeV depth into 2mm Aluminium
- Protons (+ hydrogen) of 19 MeV depth into 2mm Aluminium

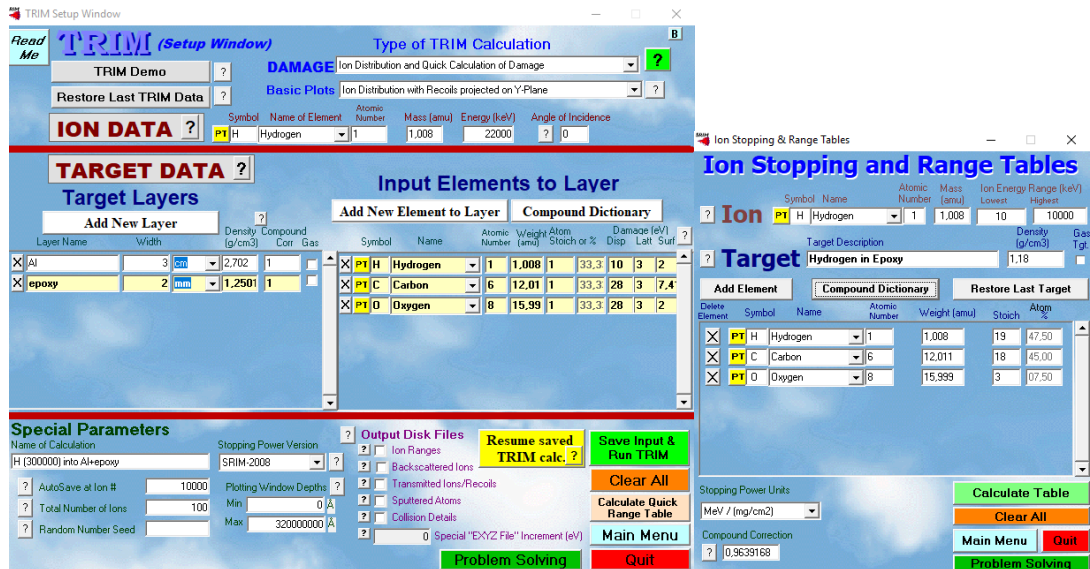


Figure 2-25: TRIM set-up windows

### 2.7.4.2 Protons of 10 MeV depth into Aluminium

When aiming a proton of 10 MeV at Aluminium, the proton is able to penetrate to about 0.7mm into the 2mm Al shield. This interaction can be seen in Figure 2-26.

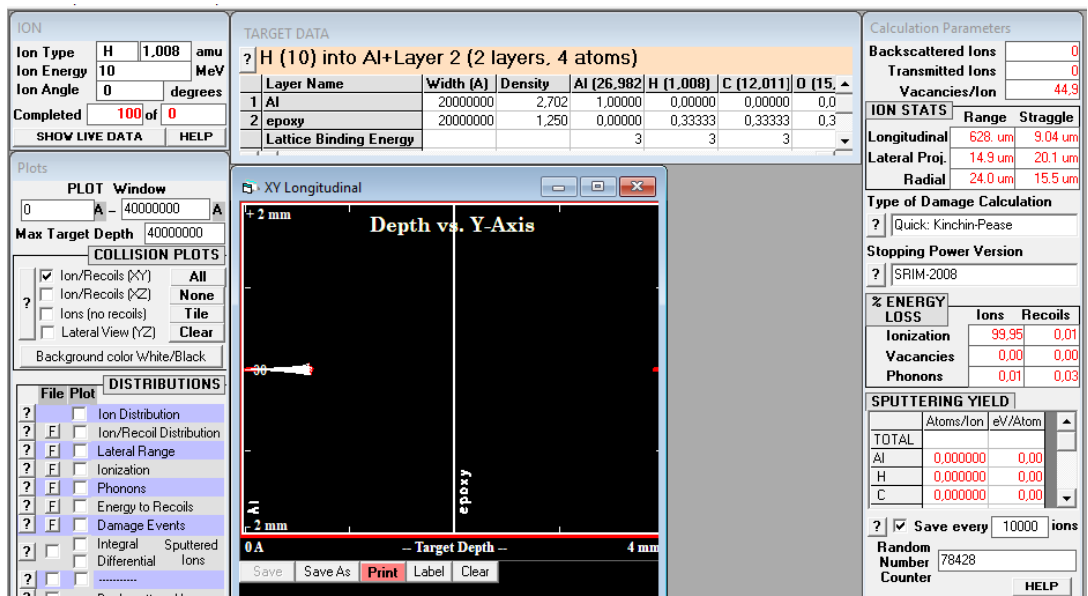


Figure 2-26: Protons (+ hydrogen) of 10 MeV depth into 2mm aluminium

### 2.7.4.3 Protons of 19 MeV depth into Aluminium

When aiming a proton of 19 MeV at Aluminium, the proton is almost able to make it through the 2mm Al shield. This interaction can be seen in Figure 2-27.

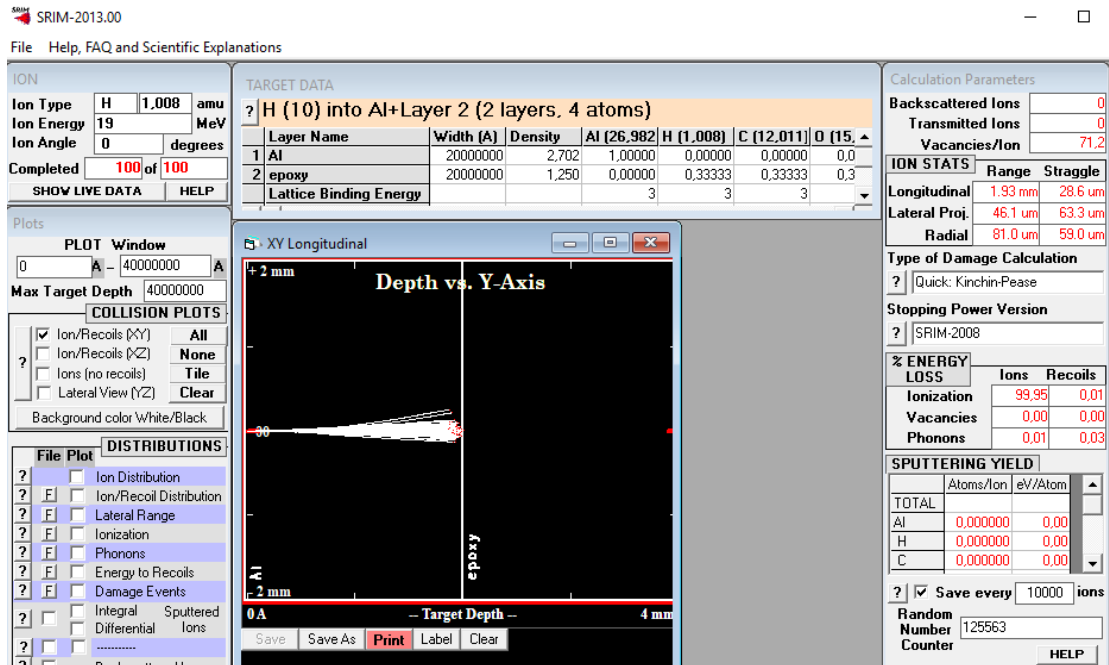


Figure 2-27: Protons (+ hydrogen) of 19 MeV depth into 2mm aluminium

## 2.8 Summary

In this document, the orbital mission for ZA cube 2 was simulated using OMERE software and different shielding was tested using TRIM. From the research done during the initial stages of this assignment, we know that space radiation causes SEE. The main cause of these events are protons and heavy ions. There are three main sources of space radiation, namely earth radiation belts (ERB), solar cosmic radiation (SCR) and galactic cosmic radiation (GCR).

Using the mentioned software, the following radiation environments that the ZA Cube 2 will be exposed to were simulated and analysed:

- Differential and integral spectra for the trapped particle in ERB
- Differential and integral spectra for the trapped particle in energetic solar particle events
- Differential and integral spectra for the trapped particle in GCR
- Transport curves for 2mm Al shielding at 98 degrees inclination
- LET curve for ZA cube 2

For all simulations involving shielding the thickness of 2 mm Al was used. It was proven that 2mm Al is only effective up to 20 MeV for protons shielding (Figure 2-22) and 30 MeV for heavy ions shielding (Figure 2-23 - hydrogen). It is also important to note that various orbits and Al thickness do not affect the LET. This is important to know when trying to prevent and predict single event effects. Using TRIM, the observations made in the paragraph above were re-enforced. Using this software a proton (positively charged hydrogen) was aimed and shot at 2mm of aluminium. The maximum energy that the 2mm Al was able to stop was just over 19 MeV.

As a final conclusion a 2mm Al shield is not able to block all the radiation that ZA cube 2 will be exposed to, however, the majority of low energized particles will be stopped. In order to ensure that SEE does not occur, additional mitigation techniques are needed to protect sensitive and vulnerable devices. These techniques could be Triple modular redundancy (TMR), software EDAC schemes, and others.

## **CHAPTER 3.**

### **ERROR CORRECTING CODES**

#### **3.1 Introduction**

Error correcting is used to prevent errors from occurring. Initially, error correcting was done manually and formed part of the quality check in most industries. However, once digital systems started to be implemented the need for error correcting codes became apparent. Unlike previously where an error would just result in a customer complaint, errors in digital systems can cause complete failure. This introduces the need for error correcting codes (ECC).

Error correcting codes allow systems to perform stand-alone error checking. Such systems ensure reliability and prevent any failures from occurring. ECC often go unnoticed as they run in the background of almost all digital-related devices. These error correcting codes are also often referred to as error-correction and detections (EDAC) codes.

##### **3.1.1 Definitions**

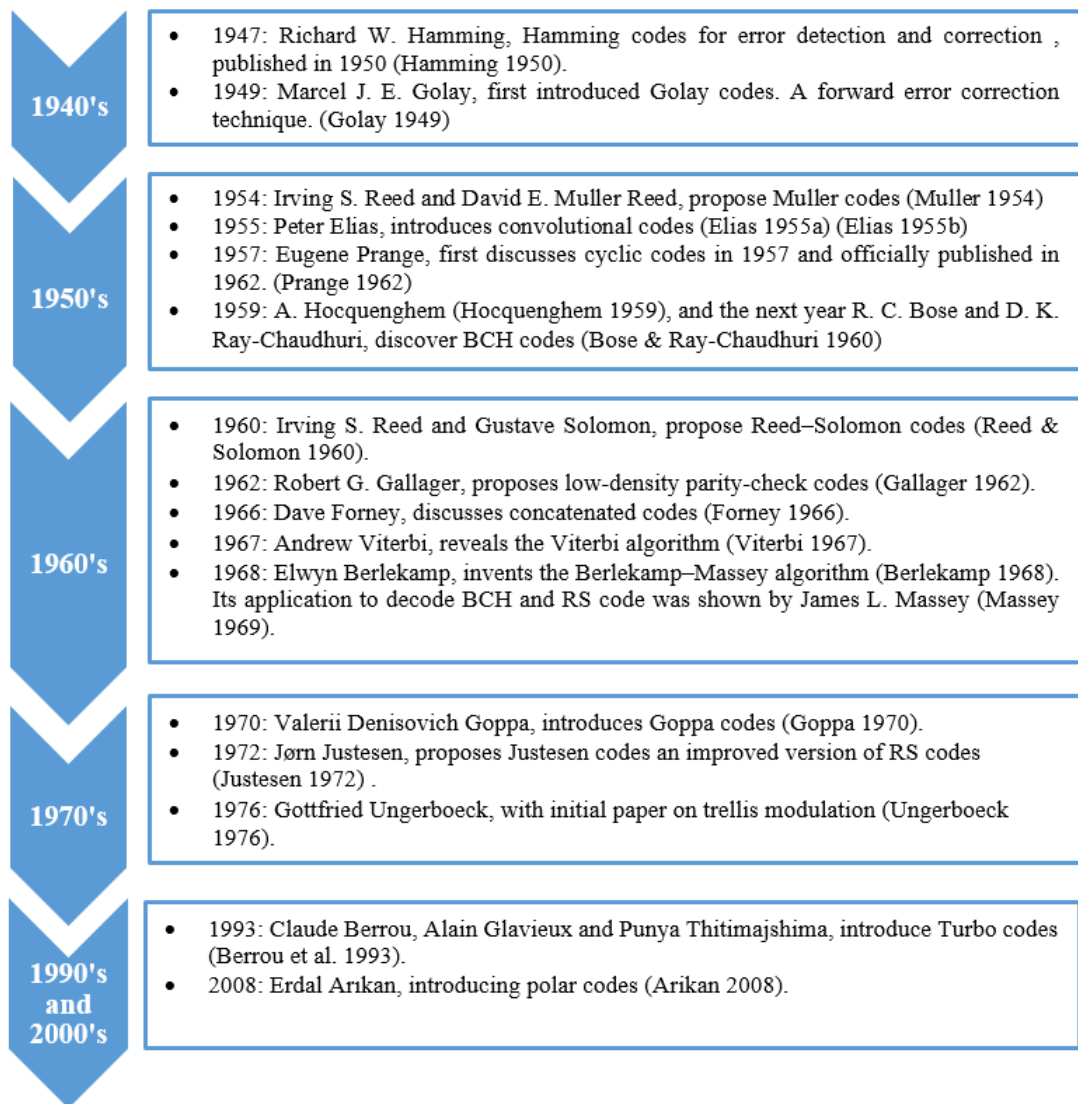
Error detection is the detection of errors caused by interference, radiation and noise during data transmission and storage. Error correction is the recovery processes that ensure any corrupted data is corrected and reconstructed to the original error-free form.

##### **3.1.2 History**

Error control codes as mentioned previously, started when digital systems and computers were introduced in the early to mid-nineteenth century. The development of these EDAC schemes was spearheaded by companies and individuals associated with telecommunication and computer science industries. In Table 3-1 a basic timeline is given, showing major developments in information theory that are linked to ECC.



**Table 3-1: ECC history**



## 3.2 Error detection and correction

### 3.2.1 Key terms and definitions

EDAC coding techniques can be split into block codes and convolutional codes. The main difference between the two is the memory. Block codes need no memory while convolutional codes need memory. Block coding converts the information signal of k-bit size to a codeword of n-bit after appending redundancy bits. Convolutional coding encodes a stream of data rather than a block of data. This means that convolutional codes are dependent not on only the current bits of data but also previous bits of data.

Once the information data (k – bits long) has been encoded, additional bits have been added to form a new data set called codeword (n – bits long). This codeword can be systematic or non-systematic. A codeword is considered systematic when the information data is unaltered in the codeword. However, should the information data be manipulated or some bits rearranged, the codeword is considered non- systematic. For example (d – shows information data and p –shows added bits):

$$\begin{aligned} \text{Systematic} &= [d0\ d1\ d2\ d3\ d4\ d5\ d6\ d7\ p0\ p1\ p3] \\ \text{Non-systematic} &= [p0\ p1\ d0\ p3\ d1\ d2\ d3\ d4\ d5\ d6\ d7] \end{aligned}$$

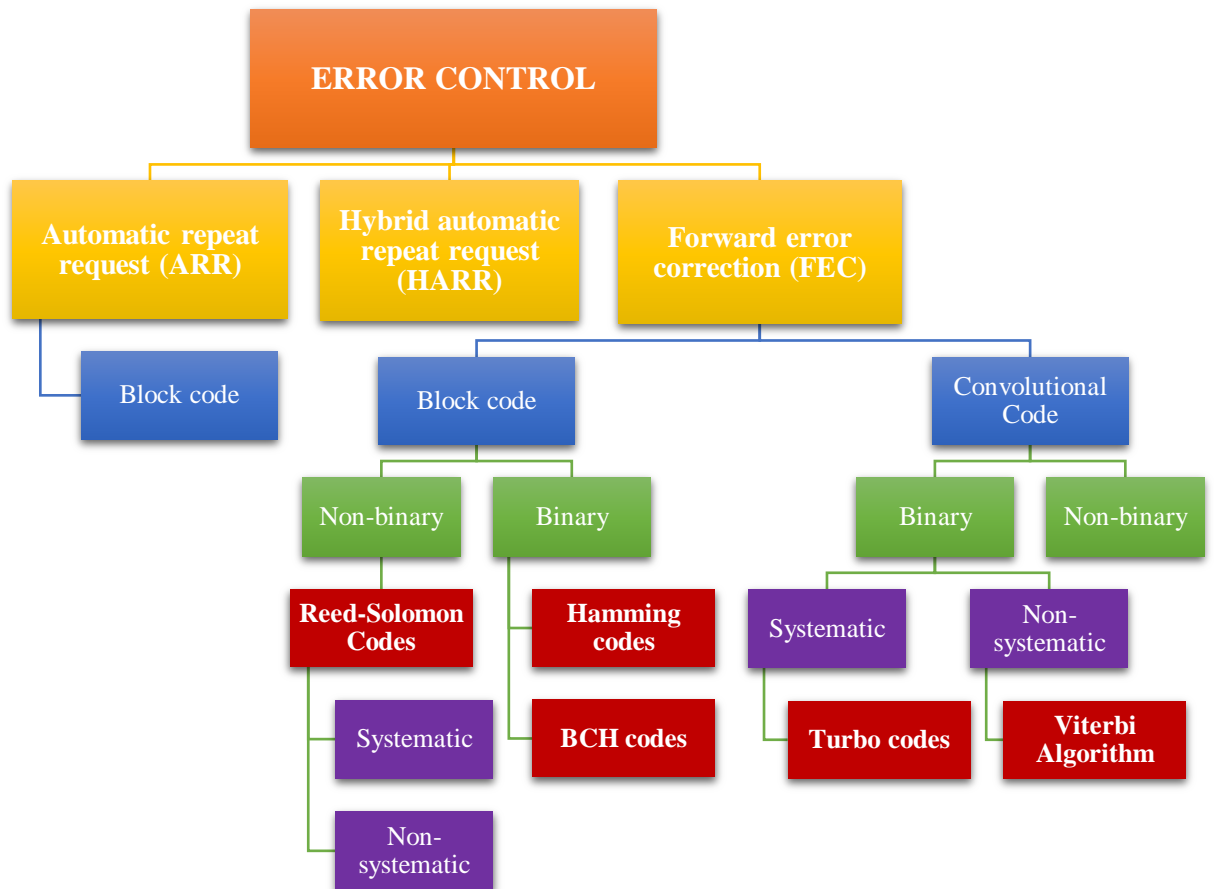
Distance or minimum Hamming distance is often used to help describe an EDAC scheme. Minimum Hamming distance is described as the number of coordinates in which two words of the same length differ.

### 3.2.2 Classification

Error control can be implemented using different techniques. These techniques can be split into three main categories, namely:

1. Automatic repeat request (ARR): This technique is sometimes referred to as backward-error correction. Using error detection schemes, the system detects an error in the data received and requests that the data is retransmitted. ARR and slight variations are used in Internet Protocol (IP), local area networks (LAN - up to 1 Gbits/s), shortwave radio and live video transmissions.
2. Forward-error correction (FEC): This technique uses both error detection and correction. The transmitted data is encoded in a redundant manner to form a codeword and then decoded. If an error occurs during transmission, the added redundant bits are used to correct the error. This error correction is done without requesting the data to be sent again. FEC is popular when re-transmissions of data is costly or impossible, such as satellite data transmissions. Most EDAC schemes are FEC.
3. Hybrid automatic repeat request (HARR): This technique is a combination of ARR and FEC. The technique uses FEC for small errors but implements ARR if the implemented FEC EDAC scheme is not able to handle the error or cause an unacceptable delay in transmission. HARR is mainly used when reliable high-speed data transmission is needed.

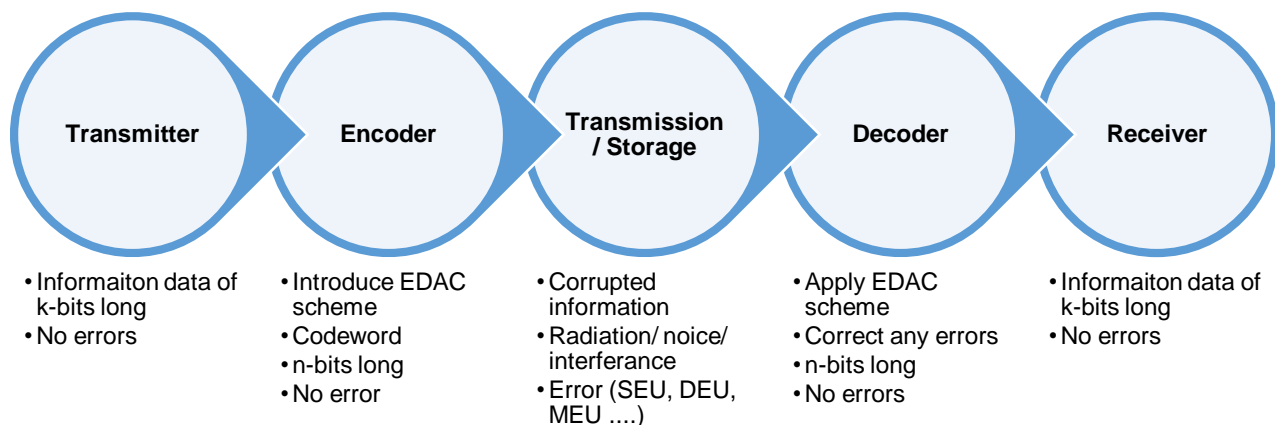
Figure 3-1 shows a chart detailing the classification of error control schemes.



**Figure 3-1: Classification of error control systems**

### 3.2.3 Implementation

Error control in digital systems has been developing for years. As mentioned previously the two main error control methods are ARR and FEC. ARR methods are basically an error detection system that requests data to be resent when an error is found. FEC, however, needs both error detection and error correction. In all EDAC systems, the process is almost the same, only the method differs. This process is shown in Figure 3-2.

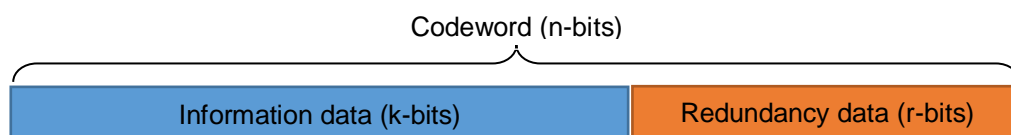


**Figure 3-2: EDAC layout (FEC)**

EDAC systems ensure no information transmitted or stored is corrupted. These systems usually follow the process shown in Figure 3-2. The transmitter (sender) sends errorless information of n-bit long to the encoder. The encoder then applies the chosen EDAC scheme and encodes the received information data to form a codeword of n-bits. The now-processed information is then ready for transmission/storage.

The type of transmission or storage is dependent on the application, for example, optic fibre, telephone wire, radio frequency, memory, etc. Transmission/storage is where error and event upsets occur. The cause of these errors can be noise, interference, radiation, etc. In some cases the cause is unknown, making errors significantly more severe as they cannot be predicted or expected. Errors are defined as bit flips in the data or memory. These errors are categorized according to the number of bits affected. For example, a single bit flip is called single event effect, two bits flipped is a double event effect. As the amount of affected bits increases, so does the severity of these errors. This influences the complexity and selection of the EDAC scheme implemented.

The transmitted information which potentially contains errors, is sent to the receiver. The receiver then decodes the information data it received or requested, by following the decoding steps of the selected EDAC scheme. This process decodes the message and corrects the errors using the added bits which makes up the codeword. This decoding process can often add a significant delay time to the retrieving process. The decoded codeword returns the transmitted data to its errorless form of n-bits long.



**Figure 3-3: General construction of a codeword**

### 3.3 Error detection and correction schemes

EDAC schemes are best described using common parameters and by their capabilities. In the sections to follow, common EDAC schemes are mentioned with a short description.

This is done using Table 3-2 and the following parameters:

- Block length ( $n$ ): Refers to the number of bits that make up a single block. These blocks are also referred to as codewords. A block is made up of information bits and redundancy bits.
- Message length ( $k$ ): Refers to the number of bits that make up the information data contained within a codeword.
- Code rate: It is a measure of how much data ( $k$  – bits) is being transferred in each codeword ( $n$  – bits). It is defined as the ratio of raw data bits to the encoded bit stream. Higher code rates normally imply that a scheme is more efficient.
- Distance ( $d_{\min}$ ): Also referred to as the minimum Hamming distance. It is defined as the number of coordinates in which two words of the same length differ.
- Alphabet ( $\Sigma$ ): Describes the modelled type used to encode the data. Most commonly  $\Sigma$  of size  $q = 2$  is used and implies that the scheme is a binary block code.  $\Sigma$  holds more significance when describing polynomial block codes, as it represents the prime powers and finite fields used during encoding.
- Notation: Describes the entire scheme in a generalized manner :  $(n, k, d_{\min})_q$

**Table 3-2: Error detection and correction capabilities of common EDAC schemes**

	SED	SEC	DED	DEC	MED	MEC
Parity Code	Yes	No	No	No	No	No
m-out-of-n Codes	Yes	No	No	No	No	No
Berger Code	Yes	No	No	No	No	No
Hamming Code	Yes	Yes	Yes	No	No	No
Hadamard Code	Yes	Yes	Yes	Yes	No	No
Repetition Code	Yes	Yes	Yes	Yes	No	No
Four Dimensional Parity Code	Yes	Yes	Yes	Yes	No	No
Golay Code	Yes	Yes	Yes	Yes	Yes	Yes
BCH Code	Yes	Yes	Yes	Yes	Yes	Yes
Reed Solomon Code	Yes	Yes	Yes	Yes	Yes	Yes

### 3.3.1 Error detection

#### Parity Code

This scheme is considered the simplest and most basic error detection scheme. When implemented only to detect single bit flips this scheme is effective, however, it is unable to correct any errors, determine the error's location or detect more than one error. This scheme functions on one of the simplest coding principles, namely, odd and even parity (Hamming 1950). The code determines whether the information data (of fixed block size) is either odd or even and then attaches a single "parity" bit to the data string. For example, information data "1100" is odd, therefore Parity code will append a "0", outputting a codeword of "11000". When an error occurs in the codeword, the codeword will no longer be odd. This means the appended "0", should have been a "1" indicating an error.

#### M-out-of-N Codes

These codes are also known as constant weight codes. Variations of the m-out-of-n codes were used in many write-once applications like PROM. Most popular was the 2-out-of-5 code, which allows decimal digits to be represented using five bits. This code was implemented in barcodes (Figure 3-4). The m-out-of-n code makes use of codeword weightings (denoted by m where  $m \leq n$ ) and length (n) to perform error detection (Ramabadran 1990). The weighting value normally represents the sum of the 1s within a codeword. When an error occurs within an n length data set, the weighting would change. This means if an error occurs that flips a zero to a one the weighting would increase and should an error flip a one to a zero the weighting would decrease.



Figure 3-4: Barcode using 2-out-of-5

#### Checksum Codes

Checksum Codes have many variations but can only perform error detection. These polynomial codes use the summing of bits to determine if an error occurred. The information data is broken up into segments. These segments are then summed column by column. The summed bits are then appended to the information data and sent together to the receiver. The receiver then performs the same summing procedure. Should no error have occurred the result will contain only zeros, or else an error has occurred during transmission. These codes have been extensively analysed based on unidirectional errors and are used in Cyclic Redundancy Check (Saxena & McCluskey 1990).

### Berger Code

J. M. Berger initially developed the Berger error detection code for the telecommunication industry in 1961. This systematic detection code allows the detection of all unidirectional errors. This means that the code is only able to detect when errors flip from ones into zeroes or zeroes flip to ones. Berger Code counts all the ones or zeroes within the information data ( $k$  bits long) and then attaches the binary equivalent of the sum to the information forming the codeword ( $n + k$  bits). The check bits of the scheme can be calculated by  $r = \lceil \log_2(k + 1) \rceil$  and then rounding the result off to the highest closest integer (Metra et al. 1995). This code is able to detect numerous bit flips as long as the flips are in the same direction, meaning all errors are flipping 1s to 0s or 0s to 1s. The receiver counts the number of ones or zeros in the information data and then compares the result with the number shown by the check bits. There are a number of improved versions of this code which have been developed over the years.

### 3.3.2 Error detection and correction

#### Hamming Code

Hamming code is a linear block error detection and correction scheme developed by Richard Hamming in 1950 (Hamming 1950). This scheme adds additional parity bits ( $r$ ) to the sent information ( $k$ ). The codeword can be calculated as  $n = 2^r - 1$ . This means information data can be calculated by  $k = 2^r - r - 1$ . Using a parity check matrix and a calculated syndrome, the scheme can self-detect and self-correct any SEE errors that occur during transmission. Once the location of the error is identified the code inverts the bit, returning it to its original form. Hamming codes form the basis of other more complex error correction schemes. The original scheme allows SECDED, but with an addition of one bit, an extended Hamming version allows SECDED (Gil et al. 2014). Table 3-3, shows the Hamming code's classification and parameters.

**Table 3-3: Hamming code classification and parameters**

Hamming Code	
Developed by	Richard Hamming
Type	Linear block code
Block length	$n = 2^r - 1$ where $r \geq 2$
Message length	$k = 2^r - r - 1$
Code rate	$1 - (r / (2^r - 1))$
Distance	$D_{\min} = 3$
Alphabet ( $\Sigma$ )	$q = 2$
Notation	$[2^r - 1, 2^r - r - 1, 3]_2$

## Hadamard Code

The Hadamard code (or Walsh–Hadamard code) is a linear block error detection and correction scheme named after Jacques Hadamard, as they are based on the Hadamard matrixes he defined in 1893. The Hadamard Code was cemented in history when it was implemented onboard the Mariner 9 satellite, launched by NASA in 1971. The scheme enabled the Mariner 9 to transmit images of Mars back to Earth. The bit overhead of this scheme are extremely large, however, it ensures data reliability in extremely noisy environments. The Hadamard code was improved to form the punctured Hadamard code, which has a slightly better code rate. Table 3-4, shows the Hadamard code's classification and parameters.

**Table 3-4: Hadamard code classification and parameters**

<b>Hadamard Code</b>	
Named after	Jacques Hadamard
Type	Linear block code
Block length	$n = 2k$
Message length	$k$
Code rate	$k / 2k$
Distance	$D_{\min} = 2k - 1$
Alphabet ( $\Sigma$ )	$q = 2$
Notation	$[2k, k, 2k-1]_2$
<b>Punctured Hadamard code</b>	
Named after	Jacques Hadamard
Type	Linear block code
Block length	$n = 2k$
Message length	$k + 1$
Code rate	$(k + 1) / 2k$
Distance	$D_{\min} = 2k - 1$
Alphabet ( $\Sigma$ )	$q = 2$
Notation	$[2k, k, 2k-1]_2$

## Repetition Code

This code is considered the most basic EDAC scheme. Repetition Code takes the simple approach of repeating information data a number of times with the idea that an error cannot affect all the information sent. The decoder then uses majority decision to compare all repeated information it received. The decoder then decides which repetition information is error free. The obvious drawback to this code is the transmission time delay due to the increased bit overhead size.



## Golay Codes

Golay codes are linear block error detection and correction schemes developed by Marcel J. E. Golay in 1949. There are two main Golay codes: perfect Golay code and the extended Golay code. The perfect Golay code allows the correction of a 3-bit error and 6-bit error detection. The extended version of Golay code uses an additional parity bit to increase the minimum distance by 1. This allows correction of a 3-bit error and 7-bit error detection. Table 3-5 shows the Golay codes' classification and parameters.

**Table 3-5: Golay codes classification and parameters**

<b>Perfect Golay code</b>	
Named after	Marcel J. E. Golay
Type	Linear block code
Block length	$n = 23$
Message length	$k = 12$
Code rate	$k / n = 12/23 = 0.522$
Distance	$D_{min} = 7$
Alphabet ( $\Sigma$ )	$q = 2$
Notation	$[23, 12, 7]_2$
<b>Extended Golay code</b>	
Named after	Marcel J. E. Golay
Type	Linear block code
Block length	$n = 24$
Message length	$k = 12$
Code rate	$k / n = 12/24 = 0.5$
Distance	$D_{min} = 8$
Alphabet ( $\Sigma$ )	$q = 2$
Notation	$[24, 12, 8]_2$

## Four-Dimensional Parity Code

Four-dimensional parity code works on establishing parity bits for a memory block (of size  $h \times v$ ). This means generating parity rows according to horizontal, vertical and cross diagonally. This allows the scheme to correct up to two-bit errors and detect up to five. A number of variations of this scheme have been developed through the years, for example, additional parity bit to increase the schemes correction ability and by changing the cross diagonally to forward and backward diagonal.

## BCH Code

The BCH or Bose–Chaudhuri–Hocquenghem codes were first proposed in 1959 by A. Hocquenghem and then separately developed by R. Bose and D. K. Ray-Chaudhuri in 1960. BCH codes are capable of multiple bit error detection and correction. This code generally uses a linear-feedback shift register (LFSR) to encode the message block and uses syndromes to determine the error location during decoding. BCH codes are extremely flexible when it comes to parameter selection, block length and code rate. These parameters are selected according to application type and the manner in which BCH codes are implemented. Table 3-6 shows the BCH code's classification and parameters that exist where  $m \geq 3$  and  $t \leq 2^{m-1}$ .

**Table 3-6: BCH code classification and parameters**

BCH Code	
Developed by	A. Hocquenghem, R. Bose and D. K. Ray-Chaudhuri
Type	Cyclic Linear block code
Block length	$n = 2^m - 1$
Check bits	$r = n - k \leq m \cdot t$
Message length	$k = n - r$
Distance	$D_{\min} \geq 2 \cdot t + 1$

## Reed Solomon Code

Reed Solomon Code is a polynomial linear block error detection and correction scheme developed by I. S. Reed and G. Solomon in 1960. This scheme is used in many applications such as Blu-ray Discs, storage systems like RAID 6 and for satellite communication.

This scheme treats data blocks as finite fields. Finite fields contain a set of pre-determined elements. These finite fields are ideal for error detection and correction as they are based on prime numbers, which allow unique values to be produced when the finite fields are manipulated. The most common of these fields is the base (2) finite field, well known as Galois Field (GF). GF (2<sup>n</sup>) fields are generated using an irreducible polynomial rather than an integer. RS code then uses three polynomials to produce the wanted codeword, name the GF, generator and encoding polynomial.

The receiver takes the codeword and creates a syndrome polynomial S(x). Using S(x) together with the Euclidian algorithm the error locator polynomial can be found, then using the Chien Search Algorithm the error can be located or pin-pointed. Once the

error's location is found, the Forney algorithm is applied and the error (or errors) can be corrected.

The explained approach is one of many to implement RS code. Table 3-7, shows Reed Solomon codes' classification and parameters.

**Table 3-7: Reed Solomon code classification and parameters**

<b>Reed Solomon Code</b>	
Developed by	I. S. Reed and G. Solomon
Type	Polynomial Linear block code
Block length	$n$
Message length	$k$
Distance	$D_{min} = n - k + 1$
Alphabet ( $\Sigma$ )	$q = pm \geq n$ ( $p$ prime) Often $n = q - 1$ .
Notation	$[n, k, n - k + 1]_2$

### 3.4 EDAC Selection

The selection of the EDAC scheme is normally solely dependent on its application. Each EDAC scheme mentioned has different advantages, disadvantages, capabilities, resource demands, complexity, etc. There are however a number of aspects that these schemes have in common, which are:

- Error detection capabilities ( $E_d$ )
  - This is the number of error bits the EDAC scheme is capable of detecting.
  - The detection capabilities are generally dependent on the minimum Hamming distance ( $d_{min}$ ) of a codeword but this is not always true.
  - Generally defined as:  $E_d = d_{min} - 1$  **3-1**
- Error correction capabilities ( $E_c$ )
  - This is the number of error bits the EDAC scheme is capable of correcting.
  - The correction capabilities are also generally dependent on the minimum Hamming distance ( $d_{min}$ ) of a codeword.
  - Generally defined as:  $E_c = (E_d) / 2$  **3-2**
- Code rate
  - It is a measure of how much data ( $k$  – bits) is being transferred in each codeword ( $n$  – bits). It is defined as the ratio of raw data bits to the encoded bit stream.
  - Defined as:  $Code\ rate = k / n$  **3-3**
- Bit overhead
  - It is the amount of redundant data in each codeword ( $n$  – bits). It is defined by the ratio of parity bits ( $r$  – bits) to data bits ( $k$  – bits).
  - Defined as:  $Bit\ overhead = r / k$  **3-4**
- Time and space complexity
  - Time complexity refers to the delay and response times that are associated with the implementation of different EDAC schemes.
  - Space complexity refers to the total algorithm space requirements after taking into account the program size and memory needed to implement the code effectively.
  - These complexities, however, can only be monitored accurately once implemented in hardware, which would allow live simulations and active testing to be conducted.

### 3.4.1 EDAC scheme evaluation

Table 3-8 shows a summary of a study conducted which compares common EDAC schemes. The table allows a graphical overview of each scheme according to codeword length, information data and minimum Hamming distance versus the error detection and correction capabilities, code rate (%) and bit overhead.

**Table 3-8: EDAC scheme comparison**

Schemes (n, k, D <sub>min</sub> )			E <sub>d</sub>	E <sub>c</sub>	Code rate (%)	Bit overhead
Hadamard Code						
16	5	8	7	3	31,25%	220,00%
32	6	16	15	7	18,75%	433,33%
64	7	32	31	15	10,94%	814,29%
Repetition Code						
5	1	-	4	2	20,00%	400,00%
6	1	-	5	2	16,67%	500,00%
Golay Code						
23	12	7	6	3	52,17%	91,67%
11	6	5	4	2	54,55%	83,33%
BCH Code m=3						
22	16	5	4	2	72,73%	37,50%
14	8	5	4	2	57,14%	75,00%
BCH Code m=4						
23	16	5	4	2	69,57%	43,75%
16	8	5	4	2	50,00%	100,00%
BCH Code m=5						
26	16	5	4	2	61,54%	62,50%
18	8	5	4	2	44,44%	125,00%
Reed Solomon Code						
6	2	5	4	2	33,33%	200,00%
8	4	5	4	2	50,00%	100,00%
10	6	5	4	2	60,00%	66,67%
12	8	5	4	2	66,67%	50,00%
20	16	5	4	2	80,00%	25,00%
Four-dimensional parity code						
Memory 32 x 8		-	5	2	75,29%	32,81%
340	256					
Memory 32 x 16		-	5	2	83,66%	19,53%
612	512					
Memory 128 x 8		-	5	2	78,77%	26,95%
1300	1024					
Memory 128 x 16		-	5	2	87,52%	14,26%
2340	2048					
Hamming						
7	4	3	1	1	57,14%	75,00%
15	11	3	1	1	73,33%	36,36%
31	26	3	1	1	83,87%	19,23%
63	57	3	1	1	90,48%	10,53%
Extended Hamming						
8	4	4	2	1	50,00%	100,00%
16	11	4	2	1	68,75%	45,45%
64	58	4	2	1	90,63%	10,34%
1024	1017	4	2	1	99,32%	0,69%

### 3.4.2 EDAC scheme selection

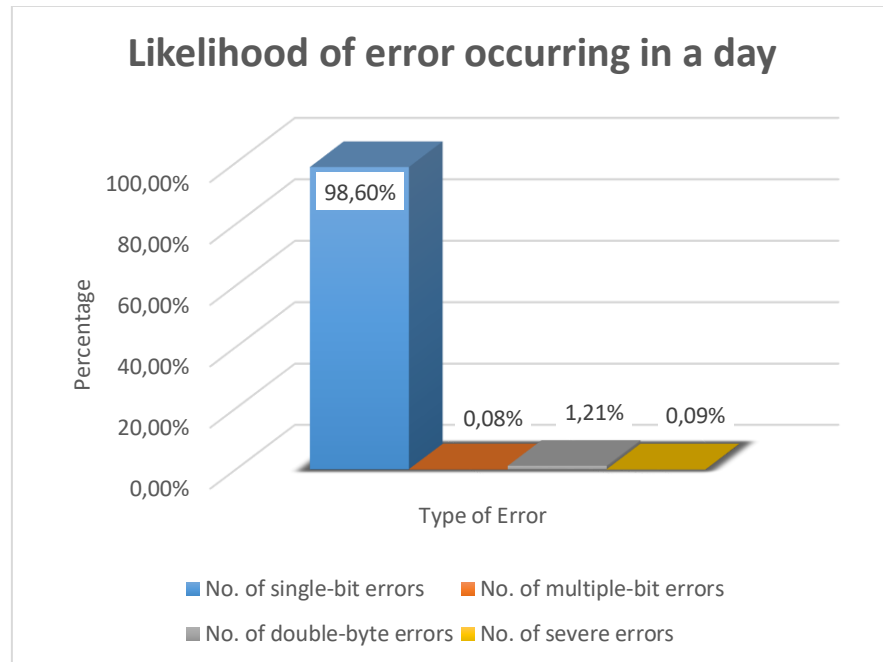
The EDAC scheme selection is almost solely dependent on its application. By referring to the title of this thesis it is clear the application is for EDAC onboard a nanosatellite. This implies that a solution is needed to detect and correct errors induced by radiation onboard a nanosatellites during its LEO. Continuing with the case study done previously in this thesis, the ZA-cube 2 (orbit of about 550 km) is used.

Ideally the effects of SEU and MEU should have been monitored in ZA-cube 1, unfortunately, this was not done and therefore this information is unavailable. However, this data was gathered by the Alsat-1, which was the first Algerian microsatellite launched into LEO. This information was published in a paper titled: “A Real-Time EDAC System for Applications Onboard Earth Observation Small Satellites” by Y. Bentoutou (Bentoutou 2012). This data is a good indication of what ZA-cube 2 will be exposed to, as the Alsat-1 is also SSO and LEO (ZA-cube 2 orbit being roughly 100 km closer to earth). This means the radiation dosages for ZA-cube 2 should be slightly lower than that experienced by the Alsat-1. However, the data from the Alsat-1 can be considered as a worst case scenario for ZA-cube 2. This data is shown in Table 3-9 (Bentoutou 2012).

**Table 3-9: Memory errors that were observed on Alsat-1 during a 7 year period**

Event upset observation for Alsat-1	
System monitored	OBC 386 RAMDISK Memory
EDAC code	R-S (256,252)
Memory size	32 Mbytes
Hybrid	SYS84000RKXLI-70 (4M x 8-bit)
Device	Samsung SEC KM684000BLT-5L: 512K x 8-BIT SRAM
Observation period	2510 days November 29, 2002— October 12, 2009
Bits monitored	268435456
No. of single-bit errors	244150 (98.6%)
No. of multiple-bit errors	217 (0.08%)
No. of double-byte errors	2996 (1.21%)
No. of severe errors	230 (0.09%)

Using the data provided in Table 3-9, it becomes apparent that SEU is by far more likely to occur as a result of radiation. SEU on average cause +/- 97 bit flips to occur during a single day (Figure 3-5). Obviously, the majority (about 80%) of these errors occur while the nanosatellite is traversing through the SAA which could range roughly between 5 to 10 minutes.



**Figure 3-5: Likelihood of error occurring in a day**

Using the information obtained in Figure 3-5, Hamming code will be used as the EDAC scheme implemented to deal with SEU. Hamming code allows both error detection and correction of single bit upset and can be extended to detect double bit flips. Once a double bit flip is detected the information can be erased and a request to resend the information can be made. Should the nanosatellite be travelling through the SAA it is possible to implement TMR that can deal with double-byte errors and severe errors. Hamming code was selected mainly as it meets the EDAC requirements, implementation complexity is minimum and cost of additional hardware is kept minimum (COTS devices can be used).

## CHAPTER 4. HAMMING CODE

### 4.1 Introduction

Hamming code is a linear block error detection and correction scheme developed by Richard Hamming in 1950 (Hamming 1950). This scheme adds additional parity bits ( $r$ ) to the sent information ( $k$ ). The codeword bits can be calculated according to  $n = 2r - 1$ . This means the amount of information data bits can be calculated by  $k = 2r - r - 1$ . Using a parity check matrix and a calculated syndrome, the scheme can self-detect and self-correct any SEE errors that occur during transmission. Once the location of the error is identified and located the code inverts the bit, returning it to its original form. Hamming codes form the basis of other more complex error correction schemes. The original scheme allows SECSSED, but with an addition of one bit, an extended Hamming version allows SECDED (Gil et al. 2014). Table 4-1 shows the original Hamming code's classification and parameters:

**Table 4-1: Hamming code classification and parameters**

Hamming Code		Hamming (7, 4)
Developed by	Richard Hamming	
Type	Linear block code	
Code rate	$1 - (r / (2r - 1))$	
Alphabet ( $\Sigma$ )	$q = 2$	
Block length	$n = 2r - 1$ where $r \geq 2$	$n = 7$
Message length	$k = 2r - r - 1$	$k = 4$
Distance	$D_{min} \leq n - k + 1$	$D_{min} = 3$
Notation	$[2r - 1, 2r - r - 1, D_{min}]_2$	$[7, 4, 3]_2$

#### 4.1.1 History

The Hamming error detection and correction code was first introduced by Richard Hamming in 1950. R. Hamming's initial paper titled "Error Detecting and Error Correcting Codes", was published in the 29<sup>th</sup> volume 2<sup>nd</sup> issue of The Bell System Technical Journal. The paper's main aim was to allow large-scale computing machines to perform a large number of operations without a single error in the end result (Hamming 1950). With the paper concentrating on systematic codes, R. Hamming proposed an EDAC scheme which is now commonly known as the Hamming code.

Richard Wesley Hamming (February 11, 1915 – January 7, 1998) was an American mathematician and pioneer who worked in the fields of computer science and telecommunication. He received his doctoral degree in 1942 for the thesis he wrote

*Hillier, C. & Balyan, V., 2019. Error Detection And Correction On-Board Nanosatellites Using Hamming Codes. Journal of Electrical and Computer Engineering. [Accepted]*

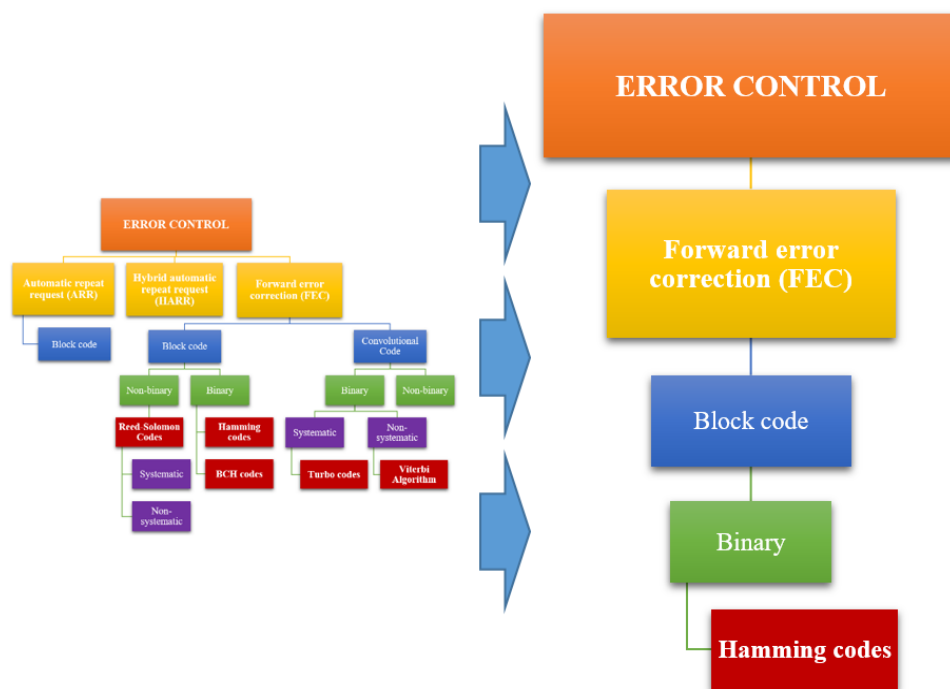


titled: “Some Problems in the Boundary Value Theory of Linear Differential Equations”. During his career, he was considered to be a teacher, mathematician, professor and a scientist. Hamming is known mostly for his work in error detection and correction (Hamming codes), but is also known for developing the Hamming window, Hamming numbers, Hamming distance and for his time spent as president of the Association for Computing Machinery (ACM). He received many awards for his contributions to science but the most prestigious of these is the one he received in 1988 from the IEEE. This award was named after him: “For exceptional contributions to information sciences and systems”. He was also the first recipient of this award (Morgan 1998).

The Hamming code initially started as a solution for the IBM machines that crashed when an error occurred. It then progressed to automatically correcting errors that occur due to punched card readers. When the code was developed it was considered a breakthrough but no one in 1950 could have predicted it being used to ensure data reliability onboard nanosatellites. To this day new applications are being found that would not have been possible without Hamming codes or at the very least the foundation set by Hamming codes.

#### 4.1.2 Classifications

EDAC codes are classified mainly according to the approach the code takes when performing error detection and correction. Hamming codes fall under the following classifications: Error detection and correction codes => Forward error correction => Block => Binary codes. This is shown graphically in Figure 4-1.



**Figure 4-1: Classification of Hamming code**

**Forward error correction (FEC):** refers to codes capable of both error detection and correction without the data being sent again. The transmitted data is encoded in a redundant manner to form a codeword. The codeword is later decoded, allowing errors to be located and corrected. When an error occurs during transmission, the added redundant bits are used to correct the error. FEC is popular when re-transmissions of data is costly or impossible, such as satellite data transmissions. Most EDAC schemes are FEC.

**Block code:** refers to codes that encode, decode and process data in set sized blocks. This approach is used in many practical applications. Block codes are useful because they enable data to be synchronized, balanced and enable error detection/correction. They allow distinct limitations and boundaries to be defined. This, in turn, relates to the adjustment of parameters to increase the capabilities and functions of EDAC codes.

**Binary code:** refers to a two-symbol language that is used to relate text, instructions, images and signals into a form that digital devices can interpret. Binary codes normally represent data by stringing together a combination of 1s and 0s. For example, a binary string of 4 bits can represent 16 possible characters. This means 4 binary bits can represent numbers zero to fifteen. Binary code is classed in terms of the alphabet ( $\Sigma$ ) of size  $q = 2$ .

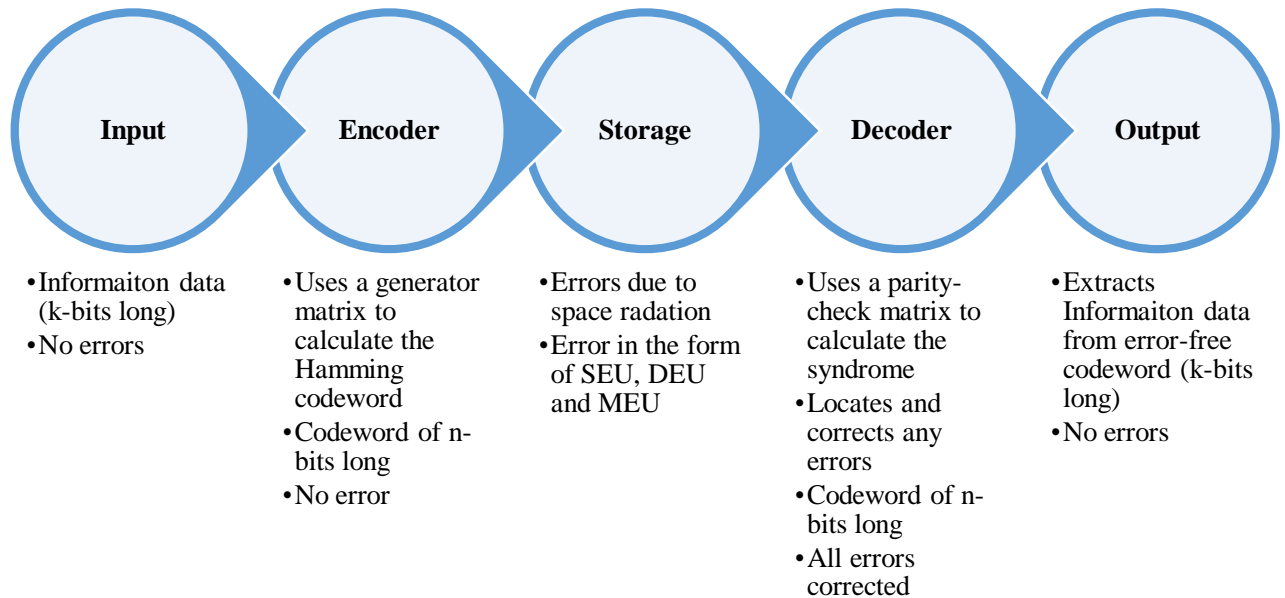
Hamming codes are block codes, which means no memory is needed for the code to function correctly. A Hamming block is basically a codeword containing both the information signal ( $k$ -bit long) and parity bits ( $r$ -bit long). The Hamming scheme can function using a systematic or non-systematic codeword. A codeword is considered systematic when the information data is unaltered in the codeword. However, should the information data be manipulated or some bits rearranged, then the codeword is considered non-systematic. This is shown in Table 4-2.

**Table 4-2: Systematic and non-systematic codewords**

<i>Systematic</i>	= $[k_0 k_1 k_2 k_3 k_4 k_5 k_6 k_7 r_0 r_1 r_2]$
<i>Non-systematic</i>	= $[r_0 r_1 k_0 r_2 k_1 k_2 k_3 k_4 k_5 k_6 k_7]$

## 4.2 Overview of Hamming code

In the sections to follow, a detailed overview of the Hamming code will be given. By touching on the parameters and foundation principles the reader should have a solid understanding of the Hamming code.



**Figure 4-2: General layout of Hamming code**

Hamming code is an EDAC scheme that ensures no information transmitted/stored is corrupted or affected by single bit errors. Hamming codes tend to follow the process illustrated in Figure 4-2. The input is errorless information of k-bits long which is sent to the encoder. The encoder then applies Hamming theorems, calculates the parity bits and attaches them to the received information data, to form a codeword of n-bits. The processed information which contains additional parity bits is now ready for storage.

The type of storage is dependent on the application, which in this thesis is RAM used by the OBC of a nanosatellite. It is during storage that errors are most likely to occur. Errors usually occur when radiated particles penetrate the memory cells contained within the RAM. These errors are defined as bit flips in the memory. Hamming codes are capable of SECSSED but can be extended to SECDDED with an additional parity bit.

The decoder is responsible for checking and correcting any errors contained within the requested data. This is done by applying Hamming theorems, which use a parity-check matrix to calculate the syndrome. The method of decoding used by the Hamming code is sometimes referred to as syndrome decoding. The decoder checks, locates and corrects the errors contained in the codeword before extracting the now error-free information data.

### 4.2.1 General Algorithm

As mentioned previously Hamming code uses parity bits to perform error detection and correction. The placement of these parity bits is dependent on whether or not the code is systematic or non-systematic. In Table 4-3: Bit layout of Hamming code (non-systematic), a typical codeword layout is shown. Please note, this table only includes bit positions 1 to 16 but can continue indefinitely.

From Table 4-3 the following observations can be made:

- Bit position (codeword) is dependent on the amount of data bits protected.
- Parity bit positions are placed according to, 2 to the power of parity bit:
  - $2^0 = 1, 2^1 = 2, 2^2 = 4, 2^3 = 8$  and  $2^4 = 16$ .
- Parity bit's relationship to bit position and data bits (also shown using Figure 4-3)
  - **Parity bit 1 (P1)** represents bit position: 1(P1), 3, 5, 7, etc. (all the odd numbers)
  - **Parity bit 2 (P2)** represents bit position: 2(P2), 3, 6, 7, 10, 11, etc. (sets of 2)
  - **Parity bit 4 (P4)** represents bit position: 4(P4), 5, 6, 7, 12, 13, etc. (sets of 4)
  - **Parity bit 8 (P8)** represents bit position: 8(P8), 9, 10, 11, 12, 13, etc. (sets of 8)
  - **Parity bit 16 (P16)** represents bit position: 16(P16), 17, 18, 19, etc. (sets of 16)
- The position in-between the parity bits are filled with data bits.
- The layout makes each column have a unique parity bit combination, for each bit position.
- This unique parity bit combination is known as the syndrome value.

**Table 4-3: Bit layout of Hamming code (non-systematic)**

Bit position	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Encoded data bits	P1	P2	D1	P4	D2	D3	D4	P8	D5	D6	D7	D8	D9	D10	D11	P16
Encoded data coverage (Non-systematic) Syndrome	P1	x		x		x		x		x		x		x		x
	P2		x	x			x	x			x	x			x	x
	P4				x	x	x	x					x	x	x	x
	P8								x	x	x	x	x	x	x	
	P16															

The syndrome allows errors to be located and corrected. For example, if parity bit P1, P4 and P8 show an error, the location of the error can be found by  $1(P1) + 4(P4) + 8(P8) = 13$ . Therefore, the error affected data bit 9 (D9) in position 13, shown by

The explanation given by Table 4-3 shows the general algorithm used when implementing Hamming code.

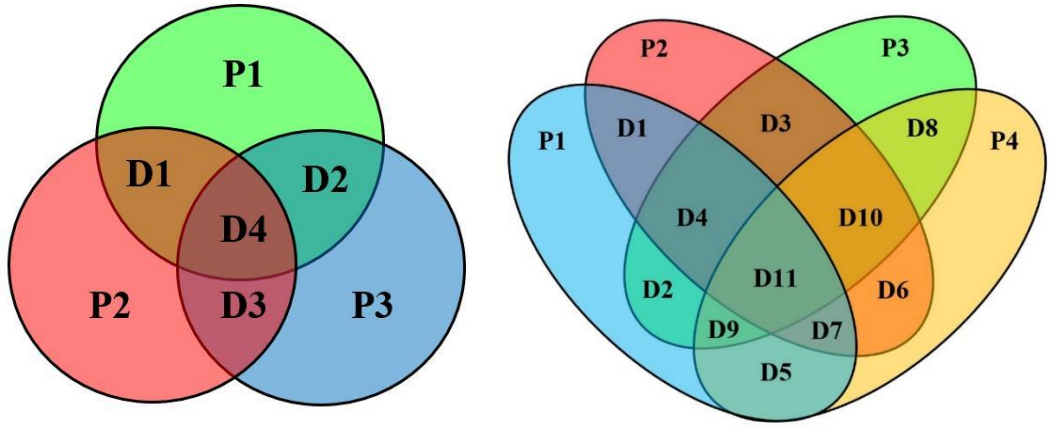


Figure 4-3: Parity VS data, Hamming (7,3) (left) & Hamming (15,11) (right)

#### 4.2.2 Construction of generator matrix (G)

Generator matrix (G) is used when encoding the information data to form the codeword. G forms one of the foundations on which the Hamming code is based. Due to the relationship between the generator matrix and parity-check matrix the Hamming code is capable of SECSED.

G (k x n) is defined as the combination of an identity matrix (I) of size k x k and a submatrix (P) of size k x r. This is shown in equation 4-1 (Chitode 2009).

$$G_{k \times n} = [I_{k \times k} | P_{k \times r}] \quad 4-1$$

#### 4.2.3 Construction of parity-check matrix (H)

Parity-check matrix (H) is used when decoding and correcting the codeword, in order to extract an error-free message. H forms one of the foundations on which the Hamming code is based. Due to the relationship between the parity-check matrix and generator matrix the Hamming code is capable of SECSED.

H (r x n) is defined as the combination of a negative transposed submatrix (P) of size k x r and an identity matrix (I) of size r x r. This is shown in equation 4-2 (Chitode 2009).

$$H_{r \times n} = [-P_{k \times r}^T | I_{r \times r}] \quad 4-2$$

#### 4.2.4 The relationship between generator matrix (G) and parity-check matrix (H)

The construction of G and H can be systematic or non-systematic. However, under both conditions, the rule shown in equation 4-3 should be met. Due to the relationship between G and H, it is possible to obtain G from H. This can be done by taking the

transposed submatrix (left-hand side) contained in H and combining it with an identity matrix of size k. This is a simple but efficient test when uncertain of G and H.

$$G * H^T = 0 \quad 4-3$$

As a final note on G and H, it is possible to manipulate this matrix from systematic to an equivalent non-systematic matrix by using elementary matrix operations, which are:

- Interchange two rows (or columns)
- Multiply each element in a row (or column) by a non-zero number.
- Multiply a row (or column) by a non-zero number and add the result to another row (or column).

#### 4.2.5 Hamming encoder

The Hamming encoder is responsible for generating the codeword (n-bits long) from the message (msg) and generator matrix (G). Once generated the codeword contains both the message and parity bits. The codeword is calculated using the formula expressed in equation 4-4.

$$Codeword_{n-bits} = mod_2(msg_{k-bits} * G_{k \times n}) \quad 4-4$$

The mathematical expression in equation 4-4 is essentially made up of AND and XOR gates. The gate/graphical expression is illustrated using Figure 4-4.

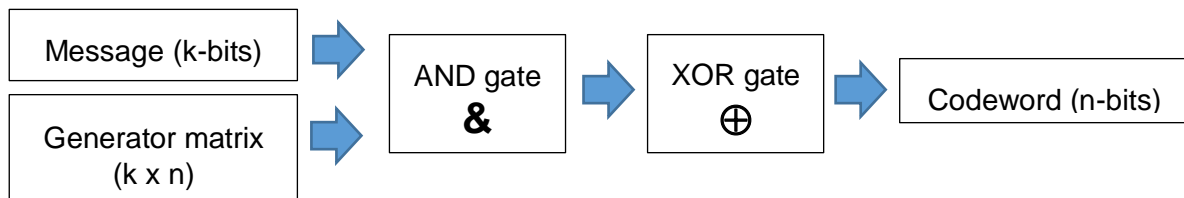


Figure 4-4: Gate/graphical expression of Hamming encoder formula

#### 4.2.6 Hamming decoder

The Hamming decoder is responsible for generating the syndrome (r-bits long) from the codeword (n-bits long) and parity-check matrix (H). Once generated, the syndrome contains the error pattern that allows the error to be located and corrected. How this is done is explained in Chapter 1. 4.2.1 General Algorithm. The syndrome is calculated using the formula expressed in equation 4-5.

$$Syndrome_r = mod_2(Codeword_{n-bits} * H_r^T) \quad 4-5$$

The mathematical expression in equation 4-5 is essentially made up of AND and XOR gates. The gate/graphical expression is illustrated using Figure 4-5.

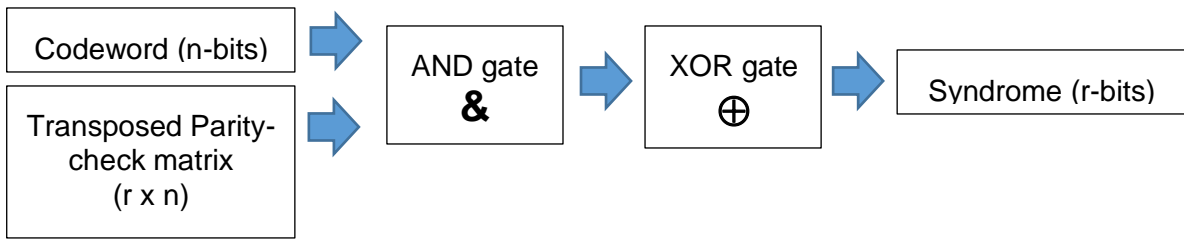


Figure 4-5: Gate/graphical expression of Hamming decoder formula

#### 4.2.7 Extended Hamming code

The extended Hamming code makes use of an additional parity bit. This extra bit is the sum of all the codeword bits (Figure 4-6), which increases the Hamming code capabilities to SECDED. The extended Hamming code can be done in both a systematic and non-systematic form. Table 4-4 shows this in a graphical manner.

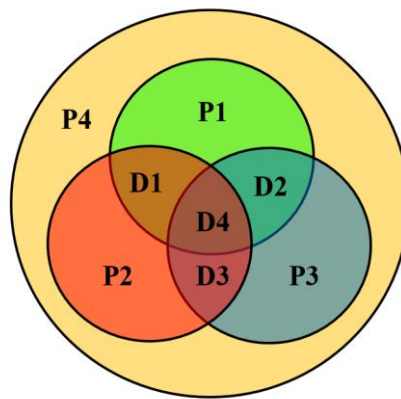


Figure 4-6: Parity relationship to data, extended Hamming (8, 4)

P16, in this case is the added parity bit that allows double error detection. By monitoring this bit and checking whether the bit is odd or even allows the double bit error to be detected, which this is shown in Table 4-4 by .

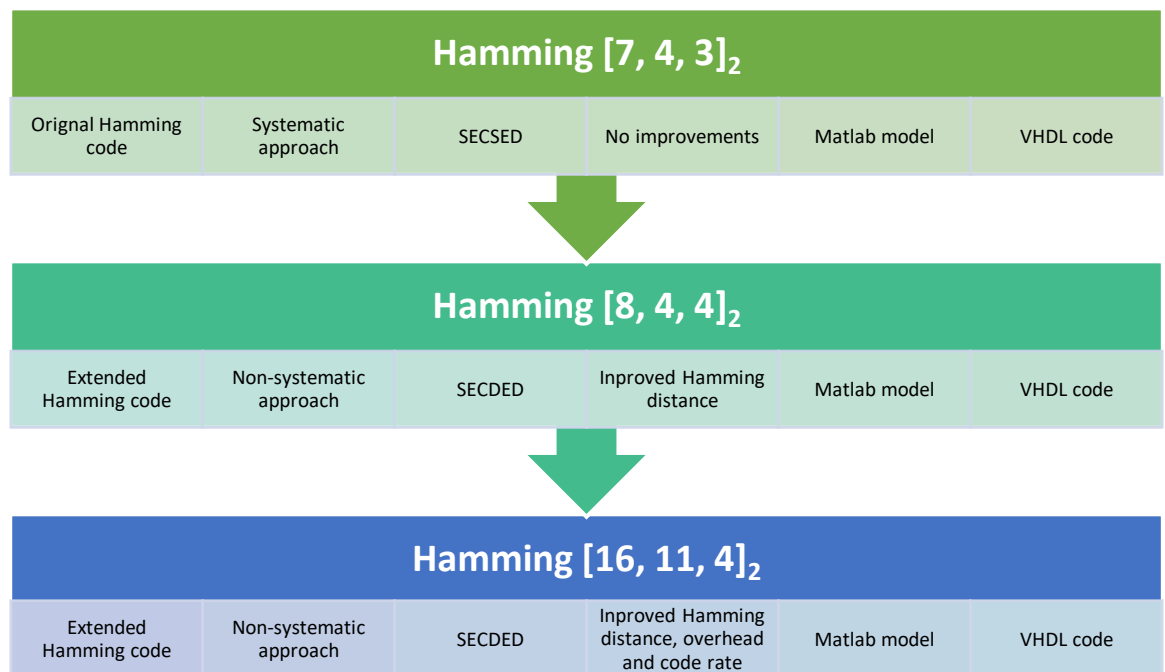
Table 4-4: Bit layout of extended Hamming code (16, 11)

Bit position	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Encoded data bits	P1	P2	D1	P4	D2	D3	D4	P8	D5	D6	D7	D8	D9	D10	D11	P16
Encoded data coverage (Non-systematic) Syndrome	P1	x	x		x		x		x		x		x		x	
	P2		x	x		x	x			x	x			x	x	
	P4				x	x	x	x				x	x	x	x	
	P8								x	x	x	x	x	x	x	
	P16															

### 4.3 Design process

The design process started with understanding the principles on which the Hamming code is based, such as generator matrix (G), parity-check matrix (H) and syndrome. Once a solid understanding of the code was established an online search through open source codes was conducted. This search was not programming language specifically but rather focused on implementation techniques. The search included Java, VHDL, C, C++ and Verilog.

The design process's ultimate goal was to implement an efficient Hamming code in VHDL. The design process followed, which led to the successful completion of this goal, is shown using Figure 4-7.



**Figure 4-7: Overview of the design process**

For each step a working model was produced in Matlab (C programming language). These Matlab models served as proof of concept before the scheme was implemented in Quartus (VHDL). Working models for the improved Hamming codes were produced in Quartus. Each model was tested and their capabilities confirmed. Each step will now be explained in the sections to follow.



### 4.3.1 Hamming [7, 4, 3]<sub>2</sub>

This was the first approach taken to implement Hamming code. Hamming [7, 4, 3]<sub>2</sub> is the original Hamming code proposed by R. Hamming in 1950. This code was implemented in a systematic manner, meaning all parity bits are appended to the end of the information data, forming the codeword. The construction of the generated codeword is shown in Table 4-5. Using the formulas shown in Chapter 1. 3.4: EDAC Selection, some performance aspects of the Hamming [7, 4, 3]<sub>2</sub> can be calculated. This is shown in Table 4-6. From the code's Hamming distance of 3 and its known capabilities, it can be stated that the Hamming [7, 4, 3]<sub>2</sub> is capable of SECSSED.

**Table 4-5: Construction of the Hamming [7, 4, 3] codeword**

Codeword (n = 7)						
Information data (k = 4)				Parity (r = 3)		
D1	D2	D3	D4	P1	P2	P3

**Table 4-6: Calculated performance aspects of Hamming [7, 4, 3]**

Scheme	E <sub>d</sub>	E <sub>c</sub>	Code rate (%)	Bit overhead
Hamming [7, 4, 3] <sub>2</sub>	1	1	57,14%	75,00%

In order to implement the Hamming [7, 4, 3]<sub>2</sub> a generator matrix (*G*) and parity-check matrix (*H*) is needed for the encoding, decoding and the calculation of the syndrome (*S*). The following calculations were done in order to implement the code:

Submatrix (*P*) of *k* × *r* dimensions:

$$P_{k \times r} = P_{4 \times 3} = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 1 \end{bmatrix} \quad 4-6$$

Identity matrix (*I*) of *k* × *k* dimensions:

$$I_{k \times k} = I_{4 \times 4} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad 4-7$$

From equation 4-1 the generator matrix is calculated as follows:

$$\mathbf{G}_{kxn} = \mathbf{G}_{4x7} = [\mathbf{I}_{4x4} | \mathbf{P}_{4x3}]_{4x7} = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 1 & 1 \end{bmatrix} \quad \mathbf{4-8}$$

Identity matrix ( $I$ ) of  $r \times r$  dimensions:

$$\mathbf{I}_{r \times r} = \mathbf{I}_{3 \times 3} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad \mathbf{4-9}$$

From equation 4-2 the parity-check matrix is calculated as follows:

$$\mathbf{H}_{r \times n} = \mathbf{H}_{3 \times 7} = [-\mathbf{P}_{4 \times 3}^T | \mathbf{I}_{3 \times 3}]_{3 \times 7} = \begin{bmatrix} -1 & -1 & 1 & 0 & 1 & 0 & 0 \\ -1 & -1 & 0 & -1 & 0 & 1 & 0 \\ -1 & 0 & -1 & -1 & 0 & 0 & 1 \end{bmatrix} \quad \mathbf{4-10}$$

Once  $G$  and  $H$  have been derived, it is possible to encode and decode the information data in a manner that is capable of SECSSED. The data is encoded using equation 4-4. This was implemented in Matlab using mod-2 additions which are essentially exclusive-OR functions. The same equation was used to encode the information data in VHDL, however, exclusive-OR gates were used to perform the operation.

Hamming  $[7, 4, 3]_2$  made use of syndrome decoding to locate and correct any errors that may occur in the codeword during transmission or storage. The syndrome is calculated according to equation 4-5. A for-loop or case statement can then be used to locate the bit which is flipped within the codeword. Once located the bit is simply inverted, returning it back to its original and correct state. With the codeword now error free, the information bits are separated and extracted, returning an error-free 4-bit message to the receiver.

### 4.3.2 Hamming [8, 4, 4]<sub>2</sub>

Hamming [8, 4, 4]<sub>2</sub> is considered as an extended version of Hamming code. With an additional parity bit, the code is capable of double error detection (DED). This code was implemented in a non-systematic manner, which simplifies the detection of double errors. The construction of the generated codeword is shown in Table 4-7. Using the formulas shown in Chapter 1. 3.4 EDAC Selection, some performance aspects of the Hamming [8, 4, 4]<sub>2</sub> can be calculated. This is shown in Table 4-8. By looking at Table 4-8 the obvious drawback of Hamming [8, 4, 4]<sub>2</sub> is the increase in bit overhead percentage. From the code's Hamming distance of 4 and its known capabilities, it can be stated that the Hamming [8, 4, 4]<sub>2</sub> is capable of SECSSED.

**Table 4-7: Construction of the Hamming [8, 4, 4] codeword**

Codeword (n = 8)							
Information data (k = 4)				Parity (r = 4)			
D1	D2	D3	D4	P1	P2	P3	P4

**Table 4-8: Calculated performance aspects of Hamming [8, 4, 4]**

Scheme	E <sub>d</sub>	E <sub>c</sub>	Code rate (%)	Bit overhead
Hamming [8, 4, 4] <sub>2</sub>	2	1	50,00%	100,00%

In order to implement the Hamming [8, 4, 4]<sub>2</sub> a generator matrix (*G*) and parity-check matrix (*H*) is needed for the encoding, decoding and the calculation of the syndrome (*S*). The following calculations were done in order to implement the code.

Submatrix (*P*) of 4 x 3 dimensions:

$$P_{4 \times 3} = \begin{bmatrix} 1 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix} \quad 4-11$$

Identity matrix (*I*) of size 4 x 4 (equation 4-7) is used when calculating *G*. From equation 4-1 the generator matrix is calculated as follows:

$$G_{4 \times 7} = [I_{4 \times 4} | P_{4 \times 3}]_{4 \times 7} = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 \end{bmatrix} \quad 4-12$$

Hamming rules state  $G$  should be of size  $k \times n$ . Therefore an additional column is needed to satisfy this condition for Hamming  $[8, 4, 4]_2$ . An 8<sup>th</sup> parity column is added to  $G$ . This is done by adding an odd or even parity bit to each row.

$$\mathbf{G}_{k \times n} = \mathbf{G}_{4 \times 8} = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 \end{bmatrix} \quad 4-13$$

By considering the non-systematic Hamming bit layout in Table 4-3,  $G$  calculated in 4-16 can be rearranged to form a non-systematic matrix  $G$ .

$$\mathbf{G}_{4 \times 8} = \begin{bmatrix} 1 & 1 & 1 & 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \\ 1 & 1 & 0 & 1 & 0 & 0 & 1 & 0 \end{bmatrix} \quad 4-14$$

Identity matrix ( $I$ ) of size  $3 \times 3$  (equation 4-9) is used to calculate  $H$ . From equation 4-2 the parity-check matrix can be calculated. With the additional parity bit, the negative submatrix is not used, as the result is the same.  $H$  is calculated in 4-15

$$\mathbf{H}_{3 \times 7} = [\mathbf{P}_{4 \times 3}^T | \mathbf{I}_{3 \times 3}]_{3 \times 7} = \begin{bmatrix} 1 & 1 & 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 & 1 \end{bmatrix} \quad 4-15$$

Hamming rules state  $H$  should be of size  $r \times n$ . Therefore an additional column and row are needed to satisfy this condition for Hamming  $[8, 4, 4]_2$ . Due to the fourth parity bit only being used for detection, the entire row can be filled with ones. With a fourth row added, an 8<sup>th</sup> parity column is also added to  $H$ . This is done by adding an odd or even parity bit to each row.

$$\mathbf{H}_{r \times n} = \mathbf{H}_{4 \times 8} = \begin{bmatrix} 1 & 1 & 0 & 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 & 1 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix} \quad 4-16$$

By considering the non-systematic Hamming bit layout in Table 4-3,  $H$  calculated in 4-16 can be rearranged to form a non-systematic matrix  $H$ :

$$H_{4 \times 8} = \begin{bmatrix} 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix} \quad 4-17$$

Once  $G$  and  $H$  have been derived, it is possible to encode and decode the information data in a manner that is capable of SECCDED. The data is encoded using equation 4-4. This was implemented in Matlab using mod-2 additions which are essentially exclusive-OR functions. The same equation was used to encode the information data in VHDL, however, exclusive-OR gates were used to perform the operation.

Hamming  $[8, 4, 4]_2$  makes use of syndrome decoding to locate and correct any errors that may occurred in the codeword during transmission or storage. The syndrome is calculated according to equation 4-5. A for-loop or case statement can then be used to locate the bit which is flipped within the codeword. Once located the bit is simply inverted, returning it back to its original and correct state. With the codeword now error free, the information bits are separated and extracted, returning an error-free 4-bit message to the receiver.

### 4.3.3 Hamming [16, 11, 4]<sub>2</sub>

Hamming [16, 11, 4]<sub>2</sub> is considered as an extended version of Hamming code. With an additional parity bit, the code is capable of double error detection (DED). This code was implemented in a non-systematic manner, which simplifies the detection of double errors. The construction of the generated codeword is shown in Table 4-9. Using the formulas shown in Chapter 1. 3.4 EDAC Selection some performance aspects of the Hamming [16, 11, 4]<sub>2</sub> can be calculated. This is shown in Table 4-10. Using Table 4-10 it is clear that Hamming [16, 11, 4]<sub>2</sub> has a better code rate and bit overhead than Hamming [8, 4, 4]<sub>2</sub>. From the code's Hamming distance of 4 and its known capabilities, it can be stated that the Hamming [16, 11, 4]<sub>2</sub> is capable of SECSDED.

**Table 4-9: Construction of the Hamming [16, 11, 4] codeword**

Codeword (n = 16)															
Information data (k = 11)											Parity (r = 5)				
D1	D2	D3	D4	D5	D6	D7	D8	D9	D10	D11	P1	P2	P3	P4	P5

**Table 4-10: Calculated performance aspects of Hamming [16, 11, 4]**

Scheme	E <sub>d</sub>	E <sub>c</sub>	Code rate (%)	Bit overhead
Hamming [16, 11, 4] <sub>2</sub>	2	1	68,75%	45,45%

In order to implement the Hamming [16, 11, 4]<sub>2</sub> a generator matrix (G) and parity-check matrix (H) is needed for the encoding, decoding and the calculation of the syndrome (S). The following calculations were done in order to implement the code:

Submatrix (P) of 11 x 4 dimensions:

$$P_{11 \times 4} = \begin{bmatrix} 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 0 & 0 & 1 \end{bmatrix} \quad 4-18$$

Identity matrix ( $I$ ) of size  $11 \times 11$  is used when calculating  $G$ .

$$I_{k \times k} = I_{11 \times 11} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \quad \mathbf{4-19}$$

From equation 4-1 the systematic generator matrix is calculated as follows:

$$G_{11 \times 15} = [I_{11 \times 11} | P_{11 \times 4}] = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 1 \end{bmatrix} \quad \mathbf{4-20}$$

Hamming rules state  $G$  should be of size  $k \times n$ . Therefore, an additional column is needed to satisfy this condition for Hamming  $[16, 11, 4]_2$ . An 8<sup>th</sup> parity column is added to  $G$ . This is done by adding an odd or even parity bit to each row.

$$\mathbf{G}_{k \times n} = \mathbf{G}_{11 \times 16} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 1 \end{bmatrix} \quad 4-21$$

By considering the non-systematic Hamming bit layout in Table 4-3,  $G$  calculated in equation 4-12 can be rearranged to form a non-systematic matrix  $G$ :

$$\mathbf{G}_{11 \times 16} = \begin{bmatrix} 1 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 \\ 1 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 \\ 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \end{bmatrix} \quad 4-22$$

Identity matrix ( $I$ ) of size  $4 \times 4$  (equation 4-7) is used to calculate  $H$ . From equation 4-2 the parity-check matrix can be calculated. With the additional parity bit, the negative submatrix is not used, as the result is the same.  $H$  is calculated in equation 4-23.

$$\mathbf{H}_{4 \times 15} = [\mathbf{P}_{11 \times 4}^T | \mathbf{I}_{4 \times 4}]_{4 \times 15} = \begin{bmatrix} 1 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 & 0 & 1 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \end{bmatrix} \quad 4-23$$



Hamming rules state  $H$  should be of size  $r \times n$ . Therefore an additional column and row are needed to satisfy this condition for Hamming  $[16, 11, 4]_2$ . Due to the fourth parity bit only being used for detection, the entire row can be filled with ones. With a fourth row added, an 8<sup>th</sup> parity column is added to  $H$ . This is done by adding an odd/even parity bit to each row.

$$\mathbf{H}_{r \times n} = \mathbf{H}_{5 \times 16} = \begin{bmatrix} 1 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 1 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix} \quad \mathbf{4-24}$$

By considering the non-systematic Hamming bit layout in Table 4-3,  $H$  calculated in equation 4-24 can be rearranged to form a non-systematic matrix  $H$ :

$$\mathbf{H}_{5 \times 16} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 1 & 1 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 & 0 & 1 & 1 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 1 & 0 & 1 & 1 & 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix} \quad \mathbf{4-25}$$

Once  $G$  and  $H$  have been derived, it is possible to encode and decode the information data in a manner that is capable of SECDED. The data is encoded using equation 4-4. This was implemented in Matlab using mod-2 additions which are essentially exclusive-OR functions. The same equation was used to encode the information data in VHDL, however, exclusive-OR gates were used to perform the operation.

Hamming  $[16, 11, 4]_2$  makes use of syndrome decoding to locate and correct any errors that may occur in the codeword during transmission or storage. The syndrome is calculated according to equation 4-5. A for-loop or case statement can then be used to locate the bit which is flipped within the codeword. Once located the bit is simply inverted, returning it back to its original and correct state. With the codeword now error free, the information bits are separated and extracted, returning an error-free 11-bit message to the receiver.

## 4.4 Implementation

Hamming code was implemented in both Matlab and VHDL. The approach taken to achieve the desired results is explained with the help of detailed flow charts. Should further insight be needed, refer to the attached appendixes.

### 4.4.1 Matlab

For each variation of the Hamming code, a proof of concept model was designed in Matlab. The approach outlined in Figure 4-8.

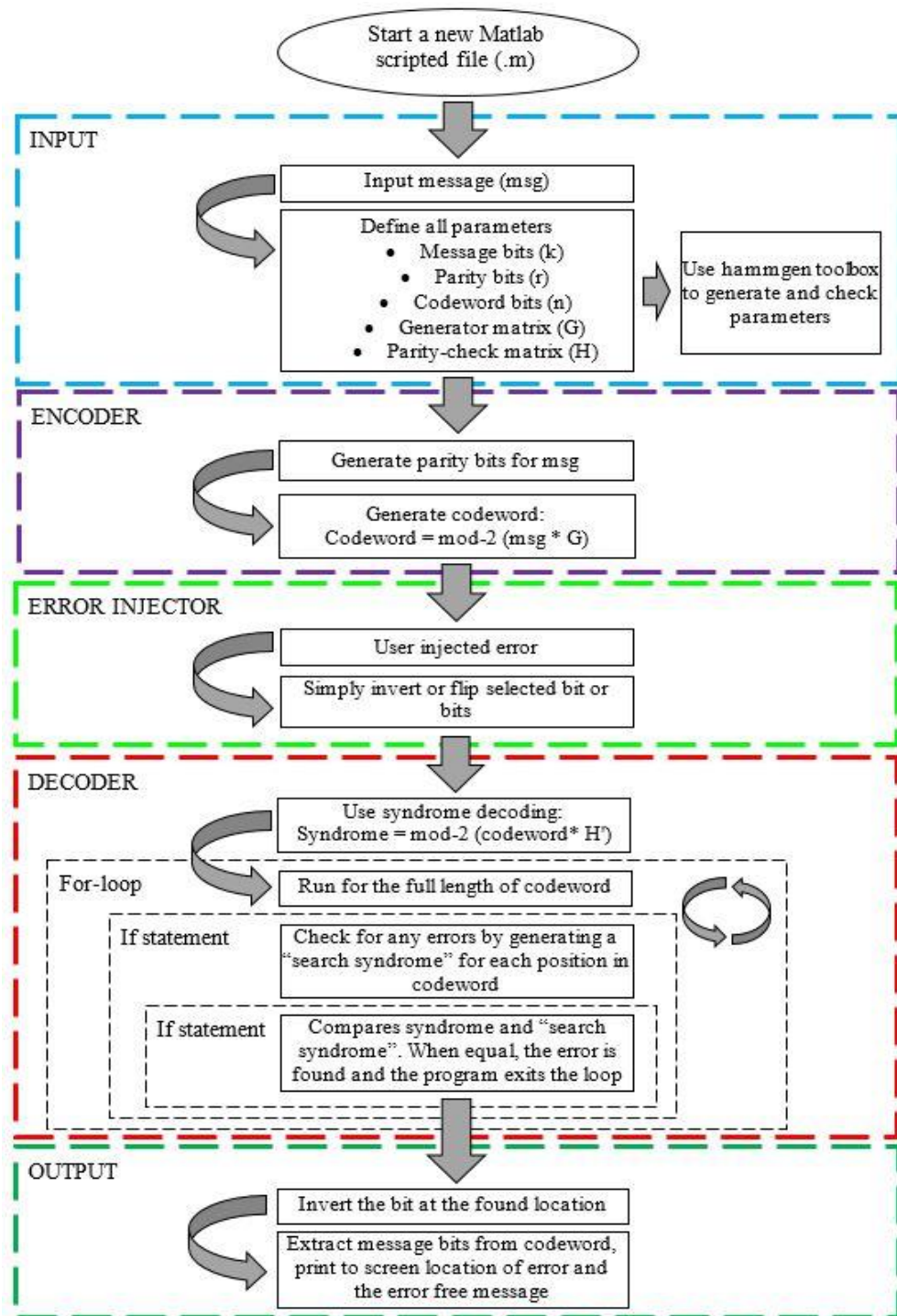


Figure 4-8: Overview of Matlab code (flow chart)

#### 4.4.2 VHDL

Once proof of concept was established using Matlab, Quartus was used to implement the working VHDL model. The Hamming code was programmed using VHDL as it allows the behaviour of the required system to be modelled and simulated. This is a major advantage when optimization is required. A working model of Hamming [8, 4, 4]<sub>2</sub> and Hamming [16, 11, 4]<sub>2</sub> was programmed in VHDL using the approach detailed in Figure 4-9.

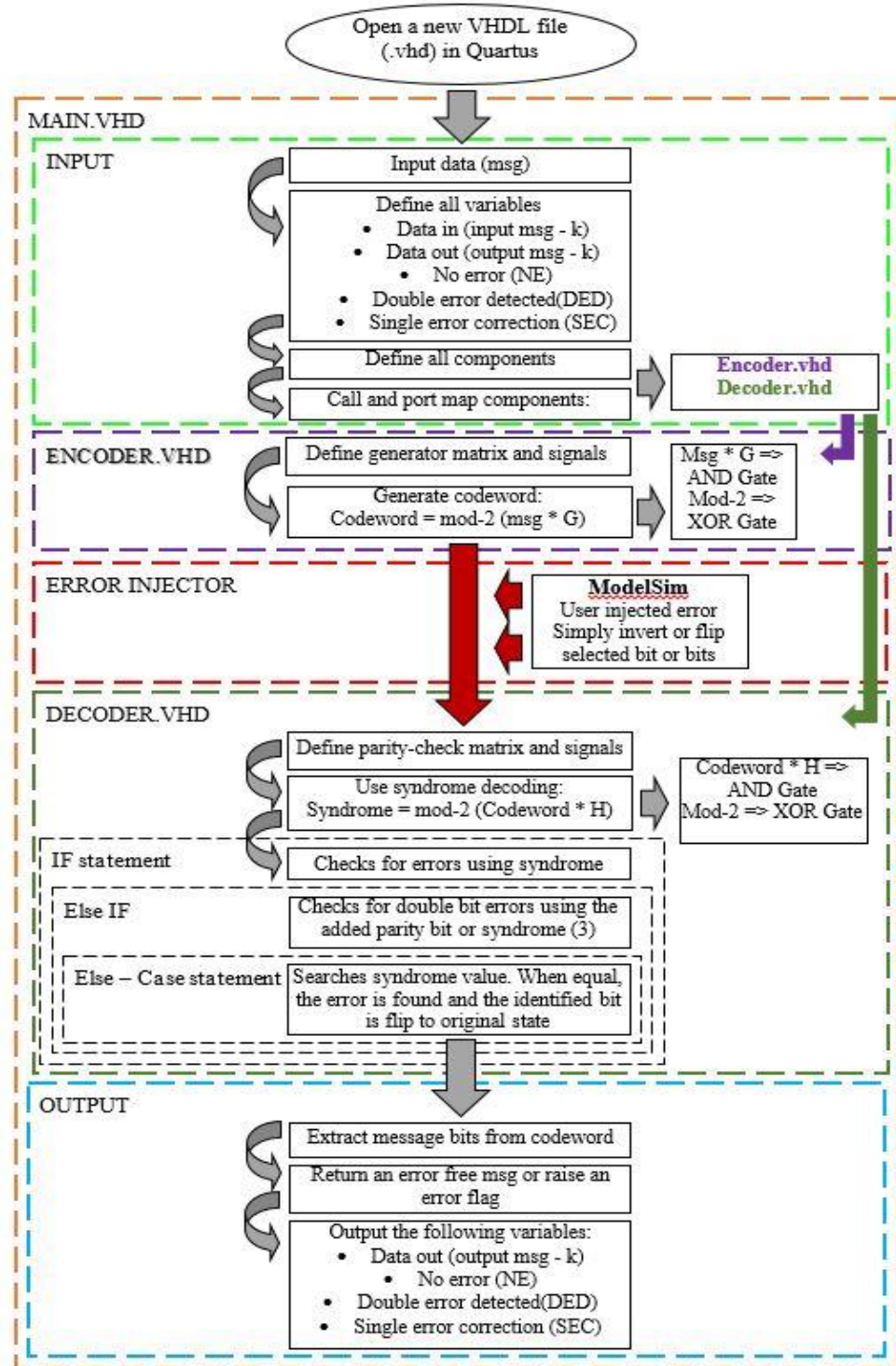


Figure 4-9: Overview of VHDL code (flow chart)

## 4.5 VHDL optimization of Hamming [16, 11, 4]<sub>2</sub>

In this section, the optimization of Hamming [16, 11, 4]<sub>2</sub> is done. The aim of optimization is to reduce resource usages, reduce time delays, improve efficiency, etc.

### 4.5.1 Code optimization

There are many ways of optimizing VHDL code. Some of the main topics when it comes to optimization are (Gschwind & Salapura 2014):

- Efficient adder implementation
- State machines
- Signal selection
- Storage structure
- Placement and Routing

Most VHDL design analysers and compilers contain tools that assist in the optimization of the mentioned aspects. It is important to understand the selected architecture, as it will help with selecting the correct design analyser and compiler. In this thesis, the Quartus Prime package is used to code, analyse, compile and optimize the Hamming code.

#### 4.5.1.1 Register transfer level (RTL) viewer of Hamming [16, 11, 4]<sub>2</sub>

Using the RTL viewer provided under tools in Quartus Prime, an overview of the I/O and VHDL code layout can be seen in Figure 4-10. Note: the overview excludes the registers and clock circuit (see Chapter 5 for full overview). The overview displays the input and outputs, as well as the components defined in `hamming_11_16_main.vhd`.



**Figure 4-10: I/O overview of VHDL code (hamming\_11\_16\_main.vhd)**

Continuing to use the RTL viewer tool it is possible to step into both the encoder and decoder. The RTL viewer optimizes the netlist in order to maximize readability. This allows a unique insight into each VHDL code. To open RTL viewer, go to the tools menu in Quartus, under Netlist Viewer and click on RTL Viewer. The RTL view of the encoder and decoder is shown in Figure 4-11 and Figure 4-12.

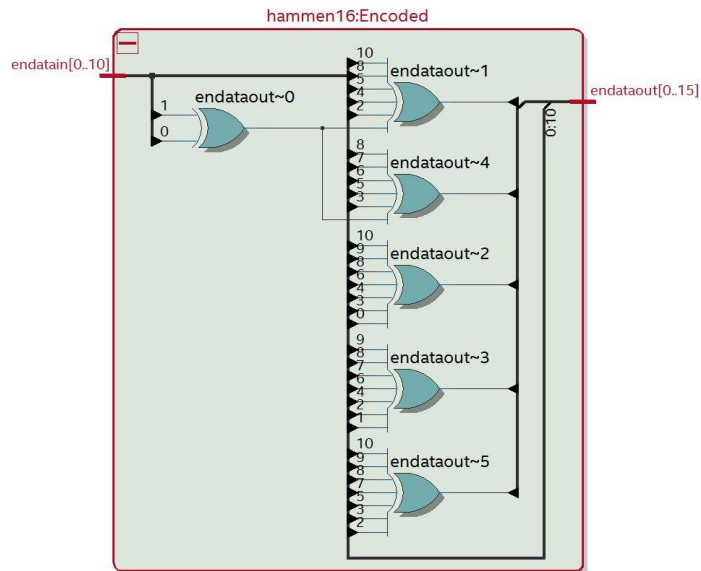


Figure 4-11: RTL overview of the encoder (hammen16.vhd)

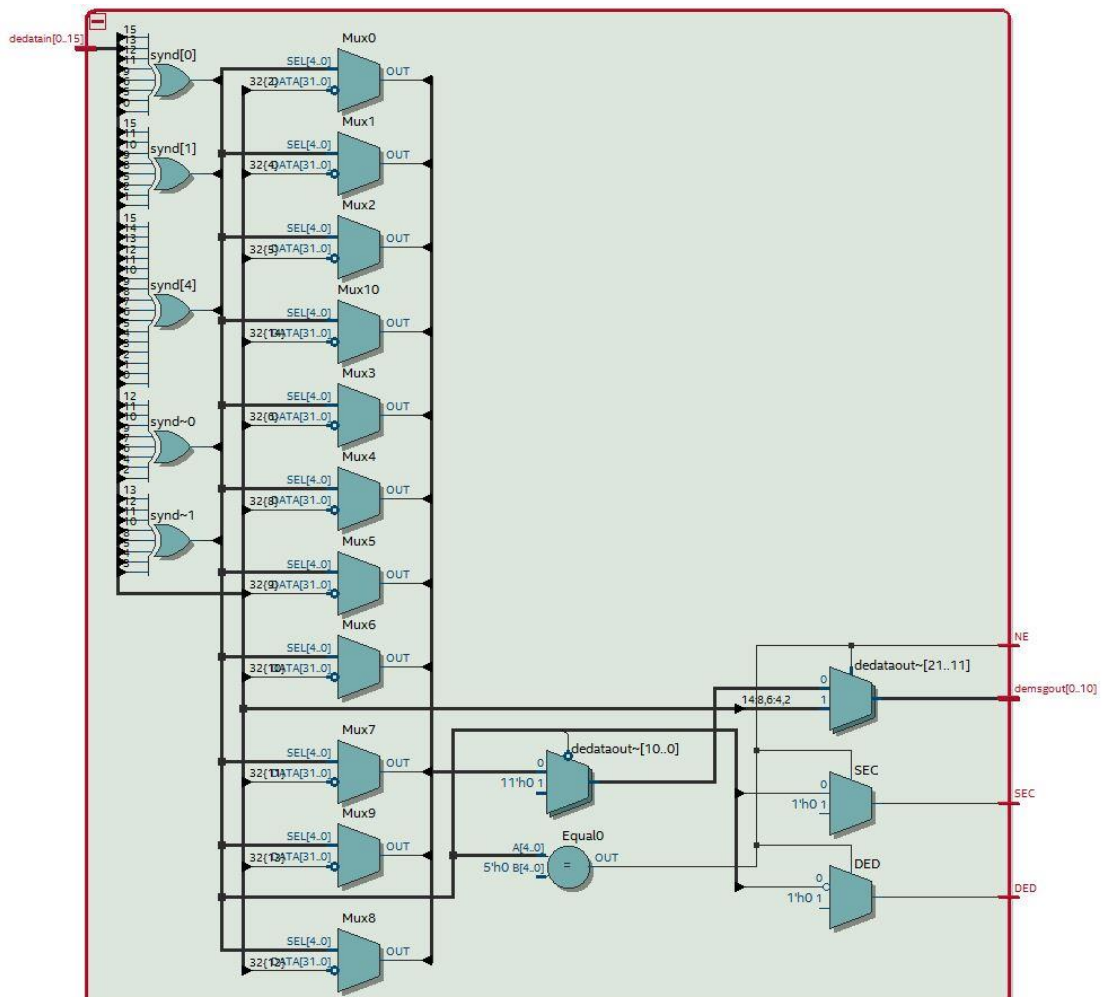


Figure 4-12: RTL overview of the decoder (hammde16.vhd)



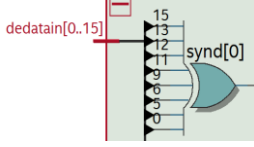
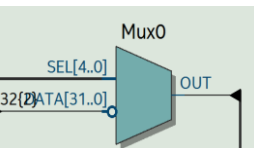
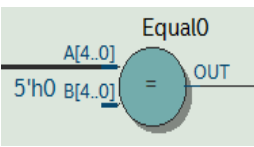
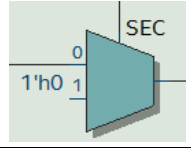
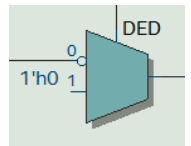
#### 4.5.1.2 Optimization of Hamming [16, 11, 4]<sub>2</sub>

As mentioned before, optimization can be done on a number of different levels. The following steps are taken to optimize the code:

- Remove unnecessary and redundant code.
- Reduce constants and variables where possible.
- Minimize the use of if statements and loops.
- Convert code to structural-level or gate-level.

All of the points are taken into account and with the help of the RTL viewer, the code was optimized. With a good understanding of Hamming [16, 11, 4]<sub>2</sub>, VHDL code, programming, FPGAs and digits, it is possible to derive the equivalent gate-level code. Using the original working VHDL code and RTL viewer, the Hamming [16, 11, 4]<sub>2</sub> code was reduced to structure or gate level. In Table 4-11 are a few examples of how the gate-level code for Hamming [16, 11, 4]<sub>2</sub> was derived.

**Table 4-11: Deriving gate-level code from VHDL code and RTL viewer**

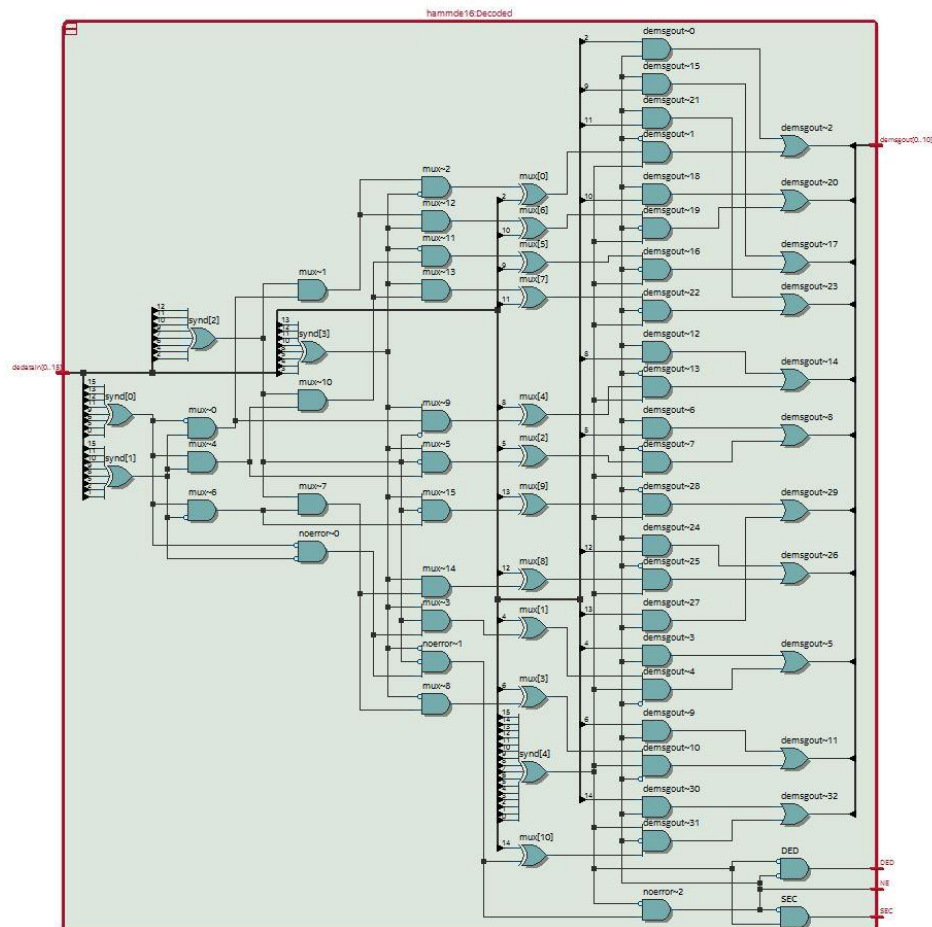
Original VHDL code	RTL viewer	Derived gate-level code
<pre>Constant MaskP1: std_logic_vector(0 to 15) signal din1: std_logic_vector(0 to 15) din1 &lt;= dedatain AND MaskP1; synd(0) &lt;= din1(0) XOR din1(1) XOR din1(2) XOR din1(3) XOR din1(4) XOR din1(5) XOR din1(6) XOR din1(7) XOR din1(8) XOR din1(9) XOR din1(10) XOR din1(11) XOR din1(12) XOR din1(13) XOR din1(14) XOR din1(15);</pre>		<pre>synd(0) &lt;= dedatain(0) XOR dedatain(5) XOR dedatain(6) XOR dedatain(9) XOR dedatain(11) XOR dedatain(12) XOR dedatain(13) XOR dedatain(15);</pre>
<pre>case synd is when "10001" =&gt; dedataout(0) &lt;= NOT dedatain(0); when ... when ... when others =&gt; dedataout &lt;= dedatain; end case;</pre>		<pre>mux(0) &lt;= dedatain(2) XOR ((NOT synd(0)) AND synd(1) AND synd(2) AND (NOT synd(3)));</pre>
<pre>if (synd = "00000") then dedataout &lt;= dedatain; NE &lt;= '1'; Else .... End if</pre>		<pre>noerror &lt;= (NOT synd(0)) AND (NOT synd(1)) AND (NOT synd(2)) AND (NOT synd(3)) AND (NOT synd(4));</pre>
<pre>If ()... else SEC &lt;= '1'; end if;</pre>		<pre>noerror &lt;= (NOT synd(0)) AND (NOT synd(1)) AND (NOT synd(2)) AND (NOT synd(3)) AND (NOT synd(4));</pre>
<pre>if (synd(4) = '0') then DED &lt;= '1'; dedataout &lt;= "0000000000000000"; Else .... End if</pre>		<pre>DED &lt;= ((NOT noerror) AND (NOT synd(4))) OR (noerror AND '0');</pre>

From Table 4-11 it can be observed that redundant code, IF-statements, loops, constants and variables, were either removed or reduced as the code was converted to gate level. By performing this operation, the lines of code that made up the encoder and decoder went from 176 to 127. This observation does not imply that the code has been optimized but is impressive none the less.

By reducing the code to gate level the following changes occurred:

- Encoder contains no constants or variables.
- Encoder went from performing 320 logic (AND and XOR) bit operations to only 30 XOR operations.
- Decoder contains no constants.
- Decoder contains a reduced amount of variables.
- Decoder went from 2 IF statements to none and from 1 case statement to none.

The results of reducing Hamming [16, 11, 4]<sub>2</sub> to gate-level is shown for the decoder using Figure 4-13. The actual results and proof of optimisation of Hamming [16, 11, 4]<sub>2</sub> will be shown in Chapter 5 (Simulation and Test results of Hamming code in VHDL).



**Figure 4-13: RTL overview of the optimised decoder**

#### 4.5.2 Quartus Prime Advisor

Quartus Prime is a design analyser and compiler from Intel. Quartus is extremely powerful in the sense that it comes with optimization advisor tools. These tools are available in all versions of Quartus, including the lite edition which is free. Figure 4-14 shows where to find the optimization advisors: Tools => Advisor => Select wanted optimization advisor.

Most useful of these toolboxes are the Resource and Timing Optimization Advisor (Figure 4-15). Once the code has gone through compilation, the advisor provides a list of recommendations for different improvements that can be made. Each recommendation comes in three types: already met, warning, and information recommendation. For each recommendation, a table showing the recommendation, description, summary and action is provided (shown in Figure 4-16).

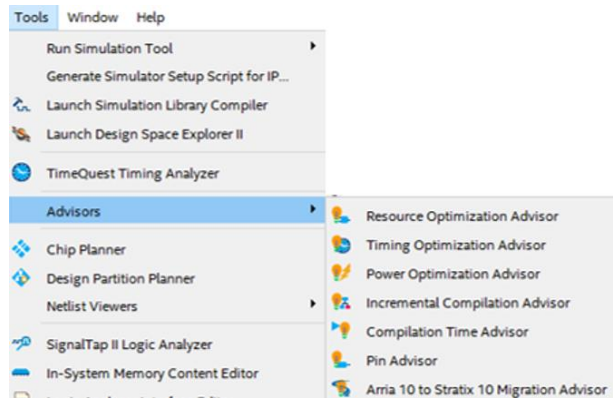


Figure 4-14: Quartus Prime advisor

For the optimisation of Hamming [16, 11, 4]<sub>2</sub> all recommendations made by the advisor were noted and changed where applicable. In most cases the recommendations were considered and taken if the summary stated logic elements usage may decrease. The change of an increase in compilation time was not really an issue due to Hamming [16, 11, 4]<sub>2</sub> code being so small.

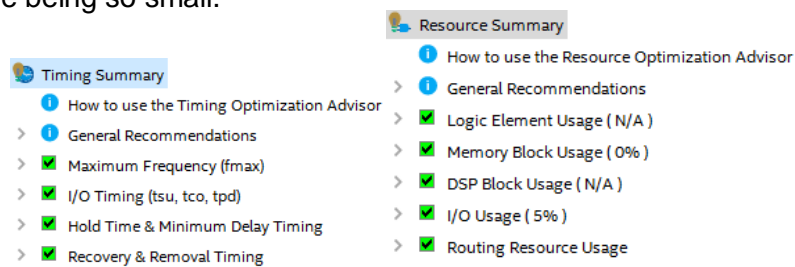


Figure 4-15: Timing (left) and Resource (right) Optimization Advisor

Recommendation	Implement the recommendations in the Resource Optimization Advisor to optimize the resource usage (logic element, memory block, DSP block, I/O, and routing) of your design.
Description	The Resource Optimization Advisor provides a set of recommendations for each design resource category. Intel recommends following the specific order of the recommendations when optimizing design resource usage.
Legend	<ul style="list-style-type: none"> <li>✔ No violations in the specified category.</li> <li>✘ Some violations in the specified category.</li> <li>⚠ Design or project settings do not match Optimization Advisor recommendations.</li> <li>⚠ Some of the design or project settings match Optimization Advisor recommendations, but some don't.</li> <li>✔ Design or project settings match Optimization Advisor recommendations.</li> <li>⬇ Optimization Advisor cannot verify if the recommended changes have been implemented.</li> </ul>
Action	Use the recommendations provided by the Resource Optimization Advisor to make project or individual settings and assignments, or make design changes to optimize the resource usage in your design.

Figure 4-16: Resource Optimization Advisor breakdown



## CHAPTER 5.

### SIMULATION AND TEST RESULTS OF HAMMING CODE IN VHDL

In Chapter 1 Hamming codes were discussed in full, with detailed descriptions of their encoding and decoding procedures. Chapter 1 also touched on different variations of Hamming codes and the optimisation methods. This chapter will show the result of the Hamming code implemented in VHDL and touch on physical protocols and testing procedures.

#### 5.1 Introduction

Hamming codes have many communication and memory applications. They are extremely popular for their effectiveness when it comes to correction of single bit flips and the detection of double bit flips.

Three Hamming codes were implemented and analysed in this thesis, namely Hamming  $[7, 4, 3]_2$ , Hamming  $[8, 4, 4]_2$  and Hamming  $[16, 11, 4]_2$ . With each step the code was improved. Hamming  $[16, 11, 4]_2$  allows SECDED, while providing a better code rate and bit overhead than Hamming  $[8, 4, 4]_2$ . Hamming  $[16, 11, 4]_2$  generates a codeword of double-byte size, which is convenient as most memory blocks work on a byte standard.

In this thesis, Hamming  $[16, 11, 4]_2$  has been implemented in VHDL on both a behaviour/dataflow and gate level (optimised). The simulation results of the Hamming  $[16, 11, 4]_2$  code for both levels of abstraction are shown in the section titled 5.2 Software tests and reports. Hamming  $[16, 11, 4]_2$  will also be optimised using a resource reduction approach and timing analysis approach. This is well documented in the section titled Optimisation.

Although software tests are able to prove functionality and test performance, it is near impossible to generate a test capable of representing a space environment exactly. Hardware tests capable of testing how a device reacts to energised protons and heavy ions is therefore needed. The international standards used during these tests to ensure valid results are documented and facilities capable of conducting these tests are in the section titled Hardware tests.

## 5.2 Software tests and reports

Using software, the developed Hamming code is tested and analysed. This is done on three levels, namely functionality (ModelSim), resource usage (compilation report) and timing analysis (TimeQuest). An overview of the tested VHDL code is shown in Figure 5-1. Figure 5-1: Full RTL overview of VHDL code (hamming\_11\_16\_main.vhd) shows the inputs and outputs (I/O), I/O registers, clocking circuit and the components that make-up the Hamming  $[16, 11, 4]_2$  code.

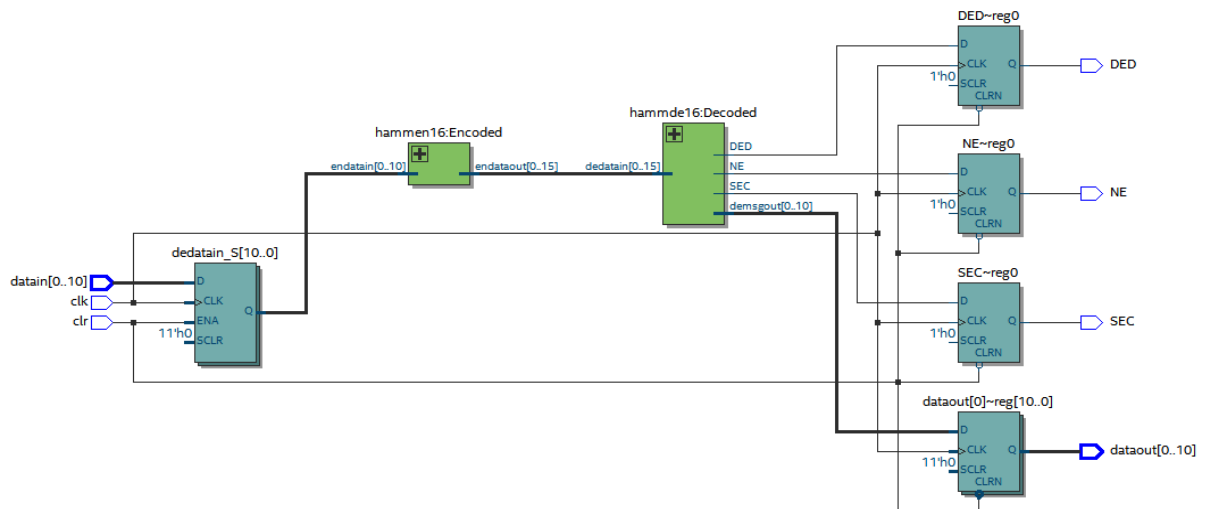


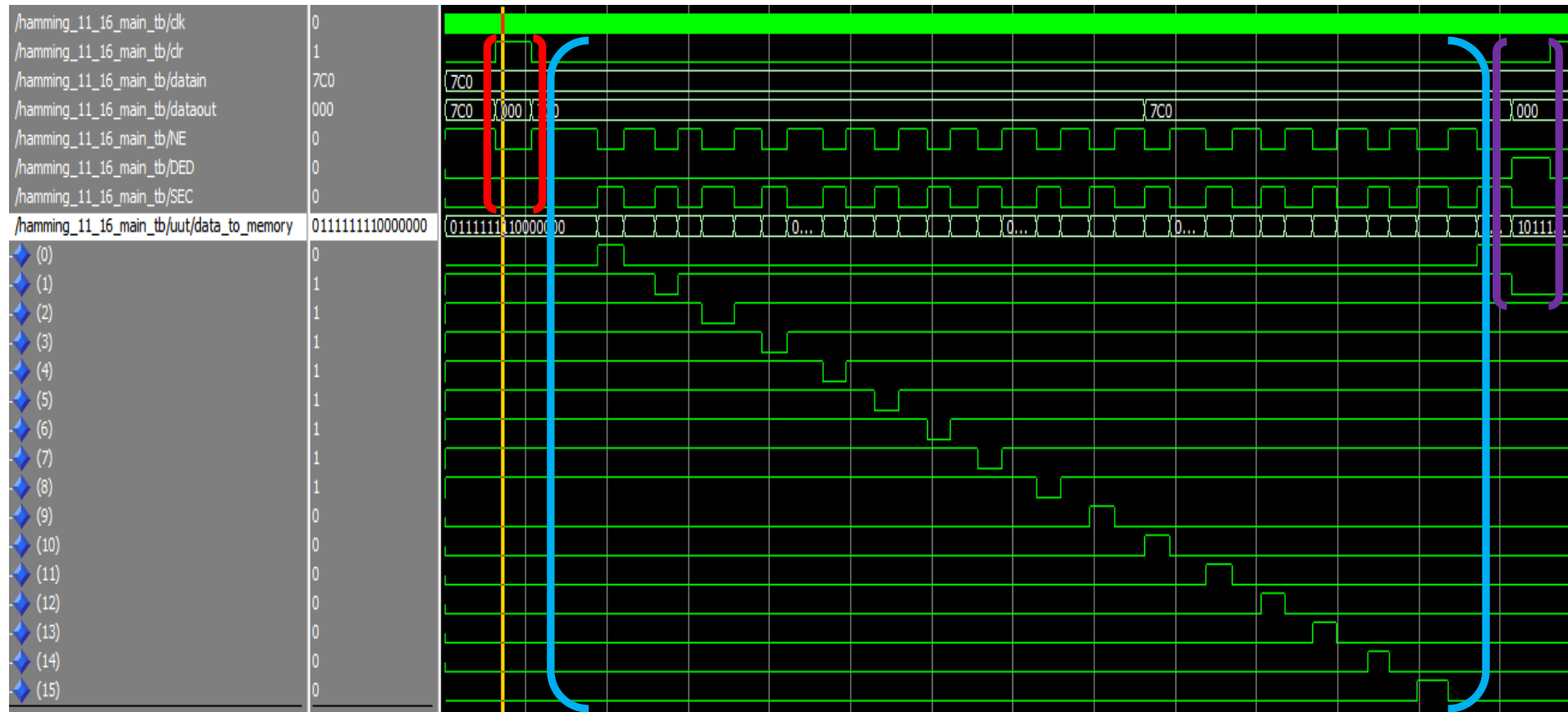
Figure 5-1: Full RTL overview of VHDL code (hamming\_11\_16\_main.vhd)

### 5.2.1 Functionality

Hamming  $[16, 11, 4]_2$  is capable of single error correction and double error detection. With the help of ModelSim, this is clearly shown in Figure 5-2. I/O registers are triggered using the rising edge of clk and can be cleared using clr. This will allow synchronisation and enables the OBC to do a full EDAC reset if necessary. Datain (input), dataout (output) and data\_to\_memory (stored codeword) display the data in the system, while NE (no error), DED (double error detection) and SEC (signal error correction) serve as indication flags.

Using Figure 5-2 the functionality of Hamming [16, 11, 4]<sub>2</sub> is proven. In Figure 5-2 the following should be noted:

- All registers are cleared using the clear signal “clr” (this is shown using ( )).
- Single bit errors are introduced in memory using a bit flip in data\_to\_memory (bits 0 to 15) and flagged by SEC (this is shown using ( )).
- Double bit errors are introduced in memory using bit flips in data\_to\_memory (bits 0 to 15) and flagged by DED (this is shown using ( )).
- Note: thanks to Hamming [16, 11, 4], dataout is unaffected by single bit errors and only gets cleared upon the detection of double bit errors.



**Figure 5-2: ModelSim simulation of Hamming [16, 11, 4] SEDED capabilities**

## 5.2.2 Resource usage summary

The resource usage summary is produced as part of the compilation report, during analysis and synthesis of Hamming [16, 11, 4]<sub>2</sub>. This summary allows the designer to understand the components, registers and gates, which will be implemented when downloading the VHDL code to an FPGA. This knowledge is especially useful when selecting an FPGA that would match the design best, ultimately reducing implementation costs and power requirements when implemented onboard a nano-satellite. The resource usage summary for Hamming [16, 11, 4]<sub>2</sub> is shown in Figure 5-3.

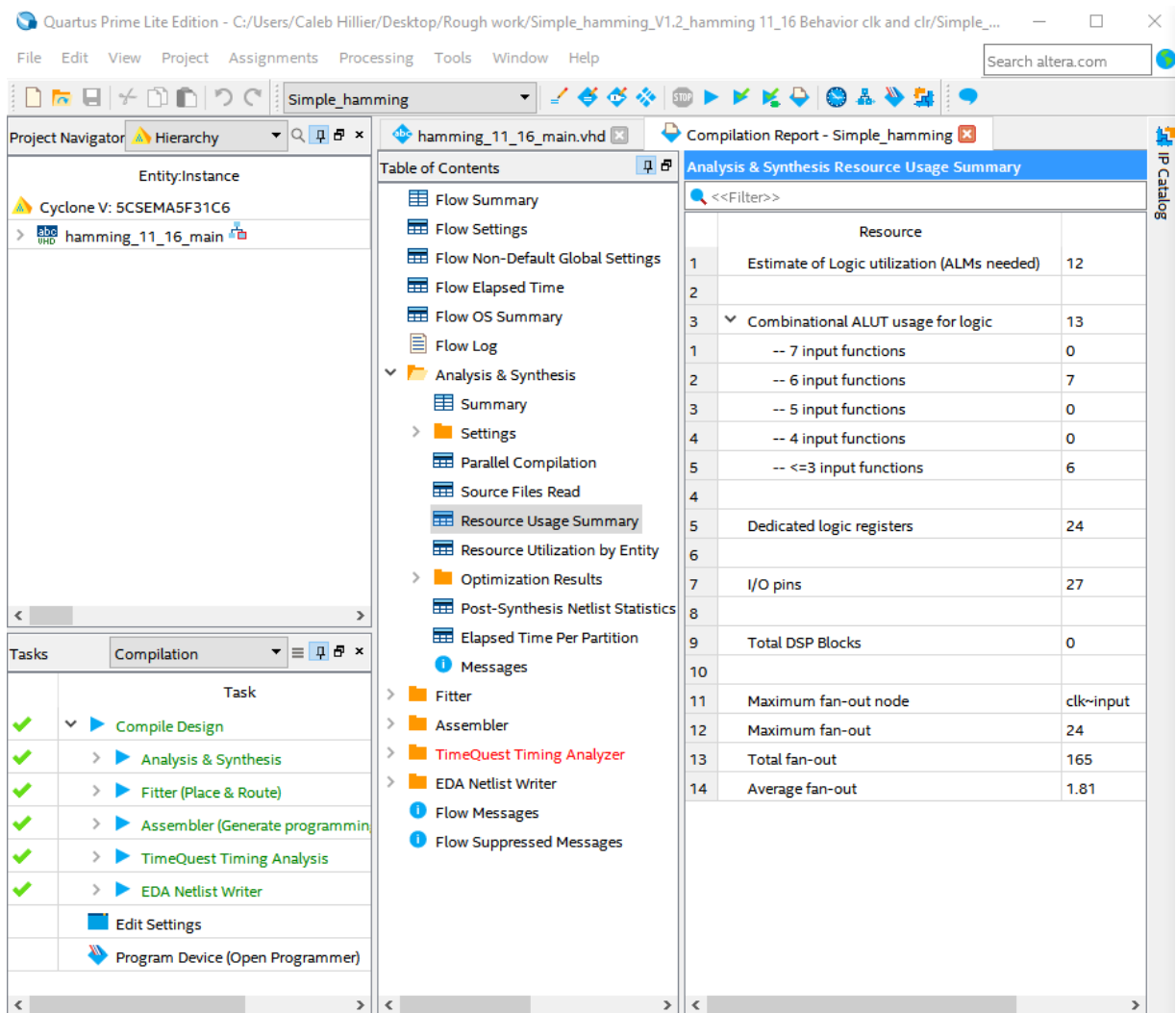
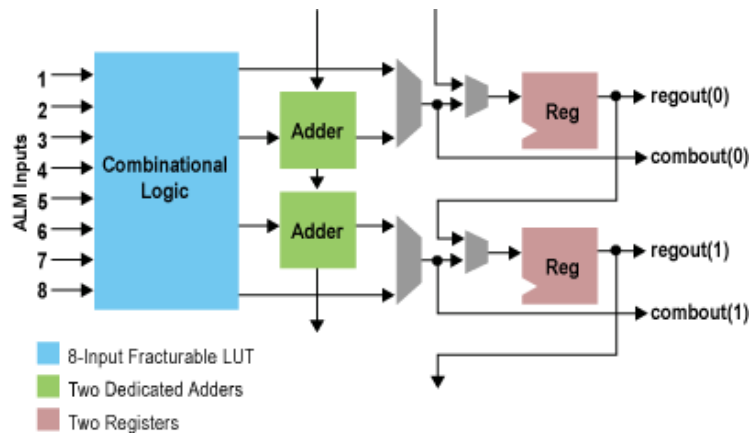


Figure 5-3: Resource usage report on non-optimised Hamming (16, 11, 4)

Important terms to understand from the resource usage summary:

- Adaptive Logic Module (ALM): basic building block of Intel devices to maximize performance and resource usage. The make-up of an ALM for Stratix Series is shown in Figure 5-4.



**Figure 5-4: ALM for Intel Stratix series (Intel 2018)**

- Adaptive Look-Up Tables (ALUT): show the combinations of LUT implemented in the ALM.
- Registers: used to hold small sets of information. This information can be temporary values or instruction sets. Often used to synchronise different sections of code.
- I/O pins: refers to the input and output pins needed on the FPGA. Used to allow the FPGA to communicate with external devices, such as memory, CPU, OBC, etc.
- Maximum fan-out: it is the greatest number of inputs that an output of a single logic gate can safely supply.

From Figure 5-3, it is obvious that Hamming [16, 11, 4] is already a small design, considering no resource optimisation has been done yet (see Chapter 1. 5.3 for optimisation results).

### 5.2.3 Timing Analyser (TimeQuest)

Quartus II packages include a timing analyser program called TimeQuest. TimeQuest can be accessed once the Synopsis Design Constraint (.sdc) file has been included and compilation has been successfully completed. The timing analyser allows the designer to analyse all timing delays that can be expected once the code has been implemented. The main point of reference during timing analysis is maximum frequency ( $f_{max}$ ), which is defined as the longest delay between two registers that run on the same clock. This analyser is especially useful when establishing clocks, delays and understanding the reaction time of paths in the code.

TimeQuest is able to detect the longest delay amongst all paths, which is known as the critical path. The critical path is then analysed and different timing aspects of the path are calculated. This was done for Hamming [16, 11, 4]<sub>2</sub> using a 1 ns clock. The results are shown in Figure 5-5 and Figure 5-6.

Path #1: Setup slack is -0.797 (VIOLATED)

Property	Value
1 From Node	dedatain_S[5]
2 To Node	dataout[10]~reg0
3 Launch Clock	clk
4 Latch Clock	clk
5 Data Arrival Time	5.624
6 Data Required Time	4.827
7 Slack	-0.797 (VIOLATED)

Figure 5-5: Timing report – Path summary for non-optimised Hamming (16, 11, 4)

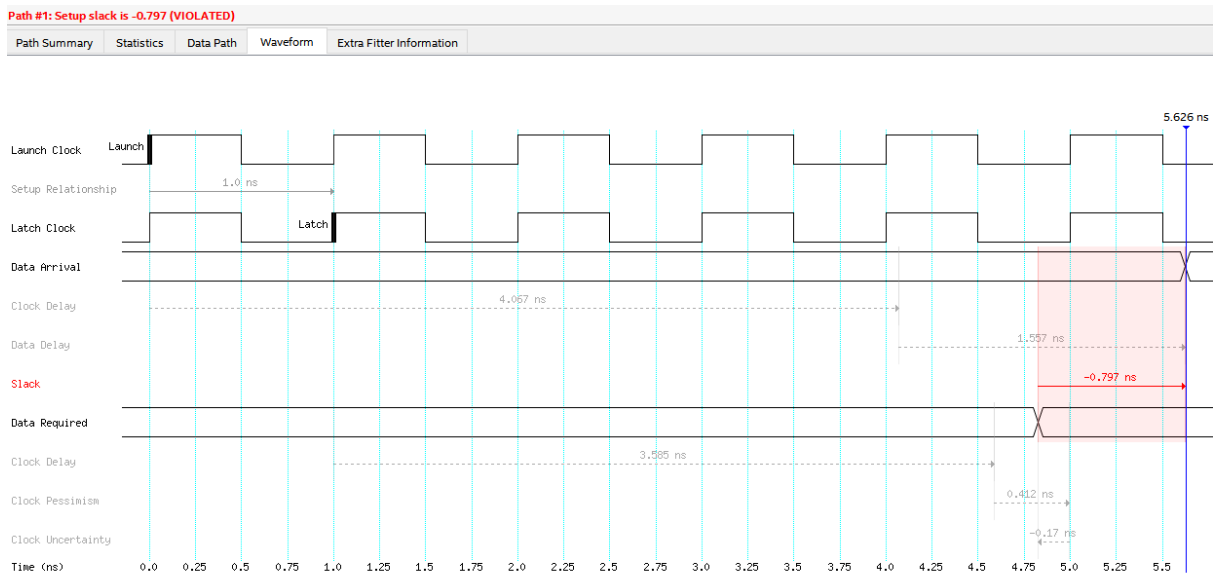


Figure 5-6: Timing report – Waveform for non-optimised Hamming (16, 11, 4)

Important terms to understand from the TimeQuest Timing Analyser:

- Data arrival time: refers to the time it takes the data to arrive at the input of the receiving register.
- Data required time: refers to the time when the data is required to be accessible at the same register.
- Slack: refers to the time difference between the required time and arrival time. A negative slack indicates a violation, while a small positive slack means the time is close to  $f_{max}$  (ideal).

From the Figure 5-5 and Figure 5-6, it is obvious that the time has not been set up or optimised. The negative slack of -0.797 ns, indicates a violation which is corrected during Chapter 1. 5.3 Optimisation.

### 5.3 Optimisation

During optimisation, it is important to understand what happens at chip-level. This will enable code optimisation and reduction. Quartus II comes with optimisation advisors that assist with resource reduction and timing optimisation. There is a definite trade-off between resource reduction and timing optimisation. It is therefore important to identify the application, and FPGA chip used for implementation, before optimising any code. Optimisation for Hamming [16, 11, 4]<sub>2</sub> from a resource reduction approach and timing optimisation is shown in the sections to follow.

#### 5.3.1 Optimisation from a resource usage reduction approach

Due to the Hamming code also already being condensed, resource optimisation is limited. However, better ALUT combination can be implemented, as well as total fan-out reduced. The optimised resources usage is shown in Figure 5-7.

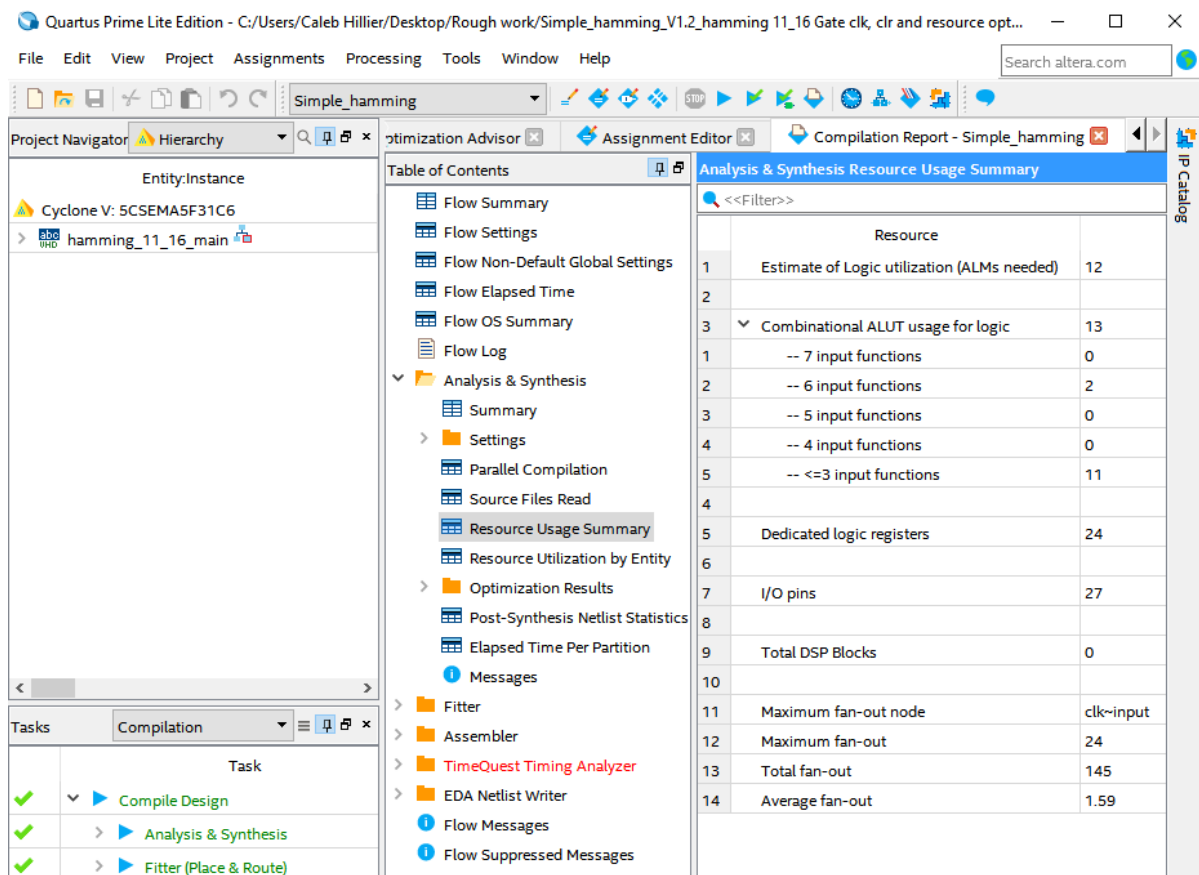


Figure 5-7: Resource usage report on optimised Hamming (16, 11, 4)

Once resource optimisation is complete, timing analysis is done. This is done using TimeQuest which calculates the slack of the critical path. This slack can then be optimised using Synopsys Design Constraint (.sdc) file and selecting an appropriate clock signal. Using TimeQuest a clock of period 2.25 ns is selected, which produces a slack of +0.089 ns. These results are shown in Figure 5-8.

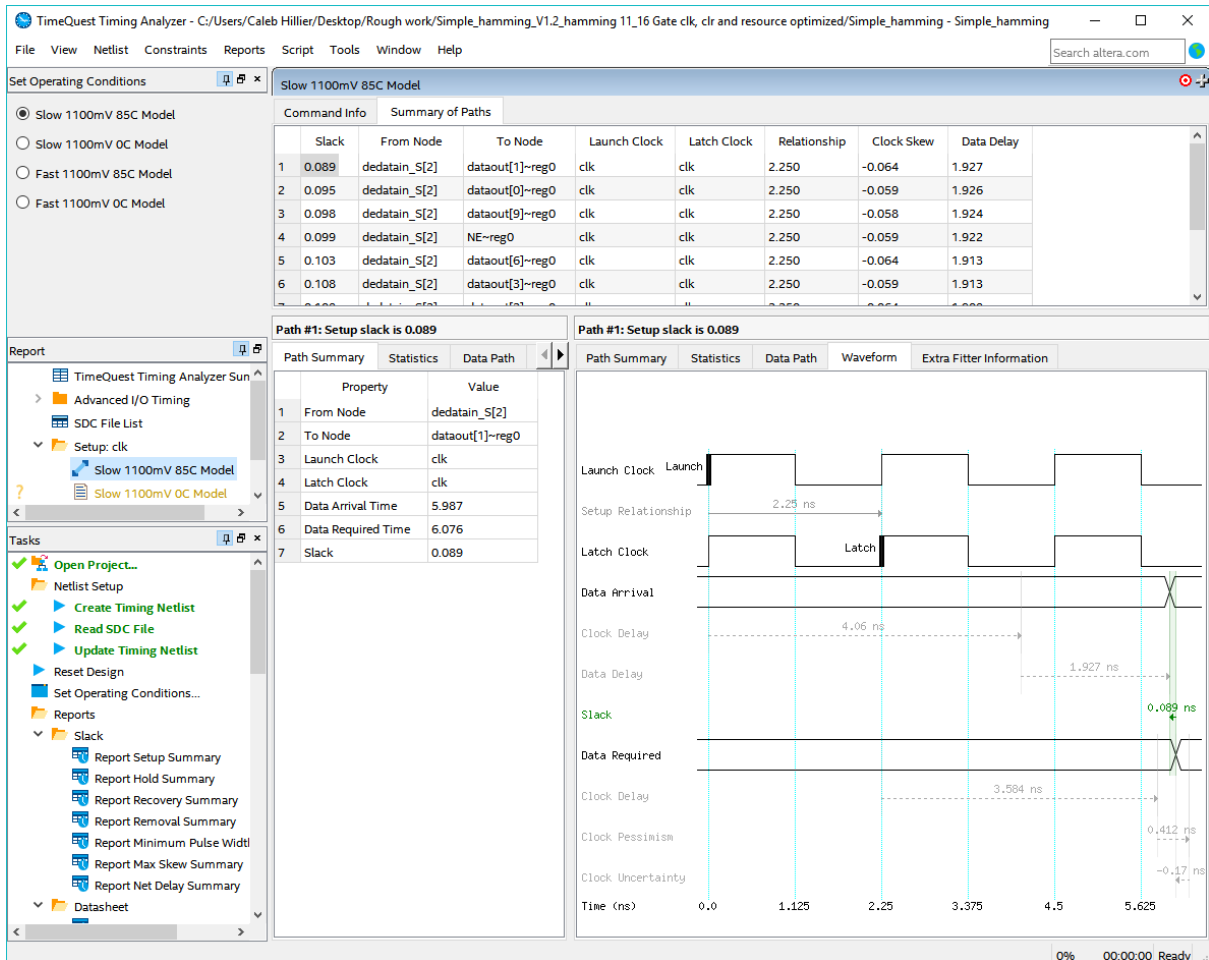


Figure 5-8: Timing report (TimeQuest) for resource optimised Hamming (16, 11, 4)



### 5.3.2 Optimisation from a timing analysis approach

Using the recommendations suggested by Quartus II and by fine-tuning the clock signal, the timing and delays experienced by Hamming [16, 11, 4] were improved. Clocking improvement was done with the help of TimeQuest, which calculated the slack of the critical path. This was then improved and minimized by editing Synopsys Design Constraint (.sdc) file and implementing an appropriate clock signal. Once all time optimisation was completed, TimeQuest was used to show the improvements. Using TimeQuest a clock of period of 1.8 ns is selected, which produces a slack of +0.097 ns. These results are shown in Figure 5-9.

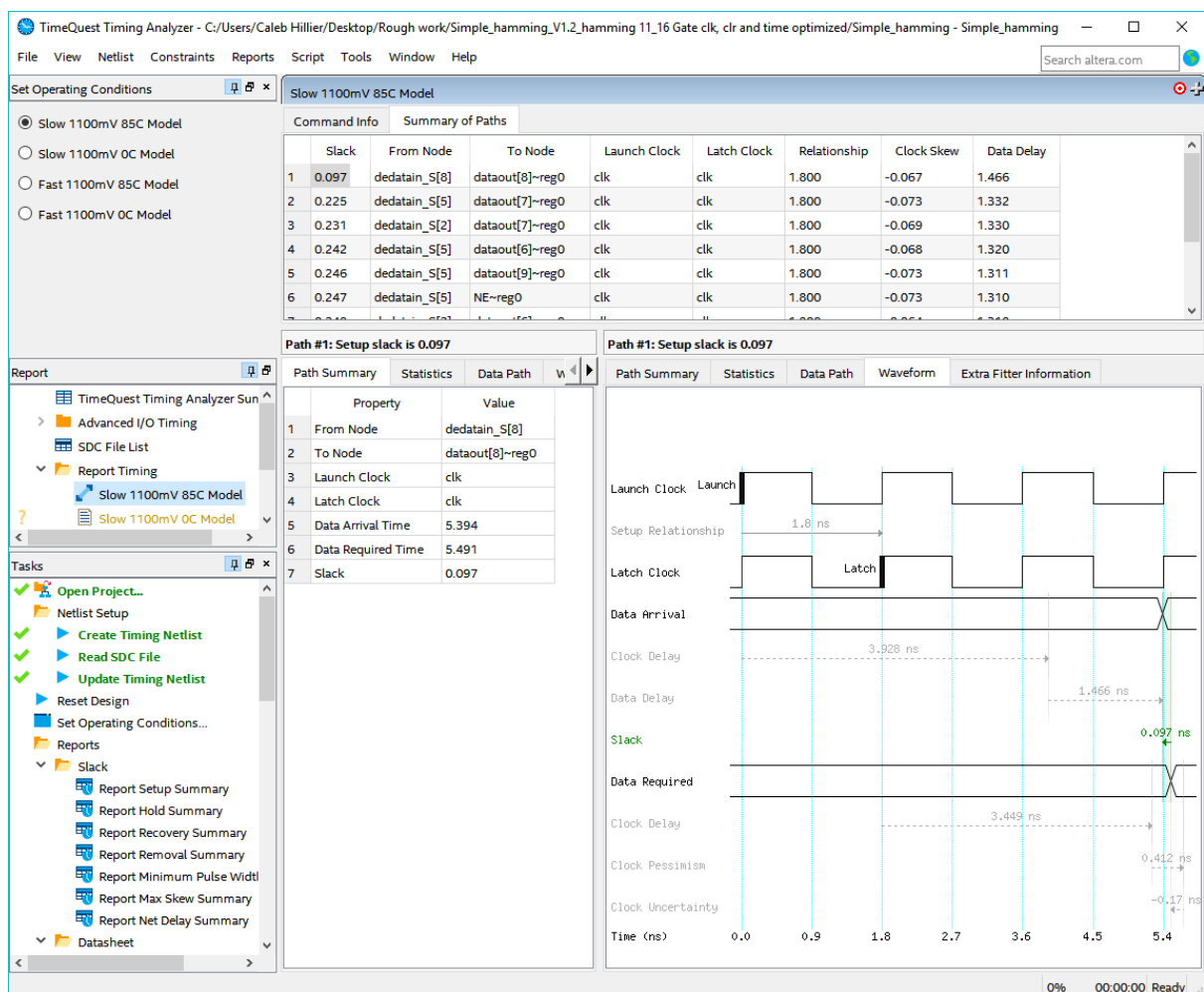


Figure 5-9: Timing report (TimeQuest) for timing optimised Hamming (16, 11, 4)

## 5.4 Hardware tests

Electronic components used onboard satellites need protection against SEE that may occur due to space radiation. Components that have been developed for space environments are referred to as rad-hardened components. These rad-hardened components however normally come with a large price tag and are extremely hard to obtain. This makes commercial off-the-shelf (COTS) electronic components highly attractive in nanosatellite projects, where budgets are low and experimental devices are welcomed (in some case encouraged).

COTS components come with a number of advantages and disadvantages. Advantages being better performance, lower prices, accessibility, etc. Disadvantages of COTS components include unreliability and weakness against transient errors. These COTS components require fault-tolerant techniques and architectures to increase reliability. One of these fault-tolerant techniques are EDAC schemes which help prevent SEE within the memory of nanosatellites.

These schemes, however, require significant testing and validation before being implemented by the space project team. There is no better manner to conduct these tests than using real energised particles to simulate the space environments. For obvious reasons, this will best represent how an EDAC would perform in space. This manner of testing is considered hardware testing. In the sections to follow, international standards and protocols will be mentioned as well as testing facilities located in Cape Town, South Africa.

### 5.4.1 Standards and protocols

In this thesis, JEDEC standards will be referred to and referenced. JEDEC has been the global leader in developing open standards and publications for the microelectronics industry for over 50 years. The JEDEC committee has many focus areas, one being solid state technology which include space products. There are two main documents which hold relevance when it comes to testing of EDAC schemes, namely JESD234 - SEE caused by proton radiation (JEDEC 2013) and JESD57A - SEE caused by Heavy Ion Irradiation (JEDEC 2017).

#### 5.4.1.1 JESD234

The JESD234 document is titled: "Test standards for the measurement of proton radiation single event effects in electronic devices". This document defines the requirements and procedures for 40 to 500 MeV proton irradiation of electronic devices for Single Event Effects (SEE), and reporting the results.

The proton (MeV) range should be determined by the space environment the satellite in question will be exposed too. This study could be done using software that predicts the space environment and radiation effects on electronic devices, like OMERE. This was done in Chapter 2 Space Radiation. Proton energy levels of below +/-50 MeV are significantly harder to test and are therefore only tested in unique cases. It is also important to note that these tests can reduce the reliability of the device and maybe render the device radioactive. This implies the device-under-test (DUT) should be a test model. Testing of above 200 MeV is considered unnecessary when testing for non-destruction SEU. With the focus on nanosatellites (LEO) the range is usually between 40 – 300 MeV.

A test facility should be selected which meets all requirements and should be able to provide documentation on:

- Proton range
- Beam characteristics
- Operating conditions
- Experimental set-up: physical arrangement of test facilities and test equipment.
- SEE detection: the closest to real-time detection as possible.
- Flux range: usually  $10^5$  to  $10^9$  protons/cm<sup>2</sup>s
- Fluence levels: nominal fluence of  $1 \times 10^{10}$  protons/cm<sup>2</sup> for soft devices. This is done with the hope that at least 100 events occur during testing.

The testing facility is also normally responsible for dosimetry, which refers to the equipment and techniques used to measure all aspects of the test. This includes proton accelerator accuracy (usually +/- 10%) and beam degraders. The researcher or engineer is responsible for putting together a detailed test plan. This is laid out in detail in the JESD234 document. Special attention should be given to test matrix, list of proton energies to be utilised and data collection. Usually lab technicians will assist with providing all relevant information and also advice on what is possible and yields the best results.

#### **5.4.1.2 JESD57A**

The JESD57A document is titled: “Procedures for the measurement of single-event effects in semiconductor devices from heavy ion irradiation”. The standards laid out in the JESD57A are specifically for heavy ion (ions with an atomic number of  $Z > 1$ ). The test results for SEE caused by heavy ion irradiation is a plot or table of the SEE cross section vs. linear energy transfer (LET). This can lead to the prediction of expected SEE rate for the DUT.

When conducting heavy ion tests, beam dosimetry is important. This should be done before the test and consists of two main aspects: measuring the energy and purity of the beam, and measuring the ion flux and the spatial uniformity. These tests normally consider beam energy variations up to 10% and a 1% beam impurity acceptable. During the test, it is important to document any additional materials in the beam's path, such as composed casing, as it affects test energies, LET and penetration range. These assessments and measurements are usually done by the test facility staff but it is ultimately the user's responsibility to ensure valid test results.

A detail test plan is required before testing is conducted. The requirements of a good test plan and testing procedures are detailed in JESD57A document (JEDEC 2017), however, certain aspects should be highlighted:

- Sample size: should be the largest possible size for the DUT
- Test conditions: for application-specific condition or worst-case conditions
- SEE detection methods: SEE must be done by comparison of the DUT response to some reference state(s) or post-irradiation bit patterns or current levels with the pre-irradiation pattern or current levels.
- Beam characteristics: refers to species and characteristics of the heavy-ion source to be used, as well as angles of incidence on the DUT.
- Flux range: typically ion flux range is  $1 \times 10^3 \text{ cm}^{-2} \cdot \text{s}^{-1}$  to  $1 \times 10^5 \text{ cm}^{-2} \cdot \text{s}^{-1}$

In the case of operating procedures, the DUT operation conditions are fixed and the beam conditions are varied. A test duration usually lasts until the desired number of errors has been measured or the desired maximum fluence has been reached. Usually, lab technicians will assist with providing all relevant information and also advice on what is possible and yields the best results.

#### **5.4.2 Test facilities**

iThemba labs provides an accelerator and ancillary facilities for research and training in nuclear and accelerator physics, radiation biophysics, radiochemical and material sciences and radionuclide productions. Situated close to Blue Downs in Cape Town, iThemba labs is the first choice when it comes to radiation testing of electronic devices. This is not only due to the lab location and test facility, but also because it is an NRF facility. FSATI who produced the ZA cube 1 and ZA cube 2 nanosatellites is also partially NRF funded. This allows certain advantages when it comes to research related to space advances.

The iThemba labs testing facilities are equipped with a Separated Sector Cyclotron (SSC) and two injectors namely Solid Pole Injector Cyclotron 1 (SPC1) and Solid Pole Injector Cyclotron 2 (SPC2). SSC is designed to allow proton acceleration of up to 200 MeV, and also to accelerate light and heavy ions using SPC1 and SPC2 (Cornell et al. 1992). The SPC1 provides pre-acceleration of light-ion beams from its internal hooded-arc PIG ion source. The SPC2 provides pre-acceleration of heavy-ion and polarised-ion beams, produced by two external ion sources. The cyclotron parameter is shown using Table 5-1.

**Table 5-1: Cyclotron parameters at iThemba labs (iThemba n.d.)**

<b>Cyclotron Parameters</b>	
<b>Separated Sector Cyclotron (SSC)</b>	
K-value	200 MeV
Injection radius (hill)	1.01 m
Largest extraction radius (hill)	4.43 m
Magnetic flux density	1.26 T (max)
Magnetic sectors	4 × 34° sectors
Resonators	2 × 49° deltas, $\lambda/2$
Dee voltage	250 kV (max)
Power amplifier	130 kW
RF power dissipation	80 kW/resonator
Frequency range	5 – 27.5 MHz
Harmonic numbers	4 & 12
Frequency variation	Short-circuit plates and variable capacitors
<b>Solid Pole Injector Cyclotron 1 (SPC1)</b>	
K-value	8 MeV
Extraction radius	0.476 m
Magnetic flux density	0.86 T (max)
Number of sectors	4
Resonators	2 × 90° degrees, $\lambda/4$
Frequency range	5 – 27.5 MHz
Harmonic numbers	2&6
Ion source	PIG source
<b>Solid Pole Injector Cyclotron 2 (SPC2)</b>	
K-value	10 MeV
Extraction radius	0.476 m
Magnetic flux density	0.86 T (max)
Number of sectors	4
Resonators	2 × 90° degrees $\lambda/4$
Frequency range	5 – 27.5 MHz
Harmonic numbers	2&6
Ion source	ECR source & polarised hydrogen source

## CHAPTER 6.

### CONCLUSION

In this thesis, EDAC schemes are extensively discussed. ZA cube 2 nanosatellite was used as a case study when conducting research on space radiation, error correction codes and Hamming code. All findings, results and recommendations are concluded in the sections to follow.

#### 6.1 Findings

Space radiation has caused numerous mission failures. Through research it became apparent that failures caused by SEU and MEU are extremely common and SEE are more frequent while the nanosatellite is in the SAA. It was found that there are a number of EDAC schemes and techniques currently used, most commonly Hamming, RS codes and TMR. The EDAC schemes are usually implemented using an FPGA. From the literature survey, it is clear that there is a need for research in the area of EDACs. By conducting an in-depth literature review, it was established that Hamming code was capable of performing the functionality desired.

In order to understand space radiation, a study was conducted using the orbital parameters of nanosatellite ZA Cube 2, which is a nanosatellite being developed by FSATI in collaboration with CPUT. Using ZA cube 2's orbital parameters a radiation study was conducted using OMERE and TRIM software which looked at earth radiation belts (ERB), galactic cosmic radiation (GCR), solar particle events (SPE) and shielding. In the case of ERB, trapped protons of a maximum integral flux of  $1.65 \times 10^3 \text{ cm}^{-2} \cdot \text{s}^{-1}$  flux at energy +/- 100 KeV, which decays to a minimum integral flux of  $1.55 \text{ cm}^{-2} \cdot \text{s}^{-1}$  flux at energy +/- 300 MeV. It was found that the common nanosatellite casing of 2mm Al was only effective as a shield against protons below 20 MeV and heavy ions below 30 MeV. In order to ensure that SEE does not occur, additional mitigation techniques are needed to protect sensitive/vulnerable devices. These techniques could be Triple modular redundancy (TMR), software EDAC schemes, and others.

There are two types of ECC, namely, error detection codes and error correction codes. For protection against radiation, nanosatellites use error correction codes like Hamming, Hadamard, Repetition, Four Dimensional Parity, Golay, BCH and Reed Solomon codes. Using detection capabilities, correction capabilities, code rate and bit overhead, each EDAC scheme is evaluated. The evaluation of Hamming codes is shown in Table 6-1. Nanosatellites in SSO LEO of around 550 km, experience on

average +/- 97 SEU bit flips per day, with an average of 98.6% of all errors being SEU. Around 80% of all errors occur within the SAA region.

**Table 6-1: Evaluation of Hamming code**

Schemes (n, k, D <sub>min</sub> )			E <sub>d</sub>	E <sub>c</sub>	Code rate (%)	Bit overhead
Hamming						
7	4	3	1	1	57,14%	75,00%
15	11	3	1	1	73,33%	36,36%
31	26	3	1	1	83,87%	19,23%
63	57	3	1	1	90,48%	10,53%
Extended Hamming						
8	4	4	2	1	50,00%	100,00%
16	11	4	2	1	68,75%	45,45%
64	58	4	2	1	90,63%	10,34%
1024	1017	4	2	1	99,32%	0,69%

Hamming codes are classified as error detection and correction codes that are forward error correction block binary codes. These codes are based on the use of parity bits which allows EDAC using a generator matrix and parity check matrix. Normal Hamming codes make use of a syndrome decoder which ultimately allows the error to be located. Once located the error is corrected to its original state. Three variations of Hamming was designed and tested during the completion of this thesis. A short summary of these codes is shown in Table 6-2.

**Table 6-2: Summary of developed codes**

Hamming scheme	Variation	Approach	Capabilities	Improvements	Matlab model	VHDL model
Hamming [7,4,3] <sub>2</sub>	Original	Systematic	SECSED	None	Yes	Yes
Hamming [8,4,4] <sub>2</sub>	Extended	Non - Systematic	SECDED	Hamming distance	Yes	No
Hamming [16,11,4] <sub>2</sub>	Extended	Non - Systematic	SECDED	Hamming distance, overhead and code rate	Yes	Yes

Hamming [16, 11, 4]<sub>2</sub> was then converted to gate level and optimised from a resource approach and timing approach. The results of the optimised code are shown in Table 6-3. With the Hamming code design complete, hardware tests were considered. The international JEDEC standards are recommended when conducting a proton (JESD234) and heavy ion tests (JESD57A). These standards ensure reliable and accurate test results. iTemba lab in Cape Town house an accelerator capable of proton (up to 200 MeV) and heavy ion testing. In order to conduct SEE tests at a facility like iTemba lab, detailed and extensive planning is necessary. This planning and actual

testing are done by both the ion beam operators and satellite engineers. This ensures cost and time is kept minimal.

**Table 6-3: Original VS optimised Hamming compression**

Hamming [16,11,4] <sub>2</sub>				
Description		Original	Resource reduction	Timing reduction
Resource summary	ALM	12	12	12
	ALUT	13	13	13
	7- input	0	0	0
	6- input	7	2	7
	5- input	0	0	0
	4- input	0	0	0
	≤ 3- input	6	22	6
	Dedicated logic registers	24	24	24
	I/O pins	27	27	27
	Max fan-out	24	24	24
	Total fan-out	165	145	165
	Average fan-out	1.81	1.59	1.81
	Timing analysis	Clock period	1 ns	2.25 ns
From node		Datain_s[5]	Datain_s[2]	Datain_s[8]
To node		Dataout_s[10]~reg0	Dataout[1]~reg0	Dataout[8]~reg0
Data arrival time		5.624 ns	5.987 ns	5.394 ns
Data required time		4.827 ns	6.076 ns	5.491 ns
Slack		-0.797 ns (violation)	0.089 ns	0.097 ns

## 6.2 Outcomes

During the course of this thesis, an in-depth radiation study of ZA cube 2's space environment is conducted. This provides insight into the environment to which the satellite will be exposed to during orbit. It also provides insight which will allow accurate testing should accelerator tests with protons and heavy ion be conducted.

A detailed look at different EDAC schemes, together with a code comparison study was conducted. This study provides the reader with a good understanding of all common EDAC schemes, which is extremely useful should different EDAC capabilities be needed.

Hamming code was extensively studied and implemented using different approaches, languages and software. The final version of the Hamming code was Hamming [16, 11, 4]<sub>2</sub>. This code allows SECDED. Using only 12 ALM the code is extremely small,



meaning the selected FPGA will consume a minimal amount of power. By optimising the resource usage the average fan-out can be reduced from 1.81 to 1.59 and runs on a period of 1.8 ns with no violation and an arrival time of 5.987 ns. When optimised from a timing perspective the code can be optimised to run off a clock period of 1.8, with no violations and an arrival time of 5.394 ns.

Due to Hamming [16, 11, 4]<sub>2</sub> resource usage of the original code already being so small, the timing optimisation approach is recommended, as it is 0.593 ns faster. This implies that a Hamming code was developed capable of protecting 11 bits of information against SEU and capable of DED. The code is capable of reading and writing to memory within 5.394 ns using only 12 ALMs and 24 registers.

The field of nanosatellites is constantly evolving and growing at an astonishing pace! This is due to the fact that it provides a platform from which the boundaries of space and technology are constantly being pushed. As technology advances memory chip cell architecture is becoming more and more dense, especially with the development of nanotechnology. This creates a growing demand for more advanced and reliable EDAC systems that are capable of protecting all memory aspects of satellites. The code developed in this thesis may not have the best capabilities but it can guarantee SECDED at fast speeds.

### **6.3 Recommendations**

During the completion of this thesis, a few aspects were identified that could be improved or further developed. These aspects are highlighted in the sections to follow.

#### **6.3.1 Investigate hybrid designs – Hamming plus TMR**

It has been shown and proven in this thesis that Hamming code is an effective EDAC scheme for single error correction and double error detection. The code itself can be implemented from a software approach (in the actual code of the satellites OS) or from a hardware approach (by adding an external FPGA). In both cases, implementation is relatively easy and additional delays and power demands are low when compared to other EDAC schemes. Under normal conditions in LEO, Hamming code is sufficient for dealing with SEE, however when traversing through the SAA, multiple bit flips (+/- 0.08% chance of occurrence) and double byte errors (+/- 1.21% chance of occurrence) are possible. In these cases, a hybrid of Hamming code could prove more effective. For example, a Hamming (SECDED) code implemented in a TMR (MECMED) manner could be a viable option for satellites in LEO. Further research into these hybrid Hamming systems and their implementation is recommended and needed.

### **6.3.2 Real data of SEE recorded on board the ZA-cube 2**

Using ZA-cube 2 as a case study during this thesis, gave the work a real problem to solve and relate to. It is recommended that if possible, RAM on board the ZA-cube 2 be monitored and all SEE be recorded. This will provide a huge amount of information with regards to SEE probability, the rate of occurrence, amount of bits affected and generating an accurate test matrix. This will allow the correct EDAC scheme to be selected for the specific orbit. Should an EDAC already be implemented onboard, its performance can be monitored and even improved depending on the type of FPGA used.

### **6.3.3 Conduct hardware test according to JEDEC standards**

Before an EDAC is implemented on board a nanosatellite, the engineers and invested parties need to be convinced of its performance and capabilities. Using the JEDEC standards while conducting protons and heavy ion tests, ensure the data obtained is both accurate and reliable. Results of such tests are essential should the developed device be commercialised after gaining flight history.

### **6.3.4 Implement Hamming code on an anti-fused based FPGA**

The Hamming code designed in this paper has been tested and compiled for a Cyclone V: DE1-SoC FPGA. This thesis establishes the size and resources needed to implement an effective Hamming code. Using this information, a suitable FPGA should be selected that meets all design specification of the nanosatellite and the code. Due to the properties of an anti-fused based FPGA, it is recommended that the tested code be implemented on a suitable anti-fused based FPGA. In an article published by The Journal of Military Electronics and Computing, the following was stated: "Further, radiation-tolerant anti-fuse FPGAs offer the following additional benefits: reduced weight and board space due to decrease in devices required; ease of implementation with no configuration components; the lowest FPGA power consumption; high reliability; and availability of medium-to-high-density solutions" (O'Neill 2003).

Once this is done and all JEDEC standards are met, the EDAC system can be implemented, launched and tested on board a nanosatellite and tested while in orbit.

## REFERENCES

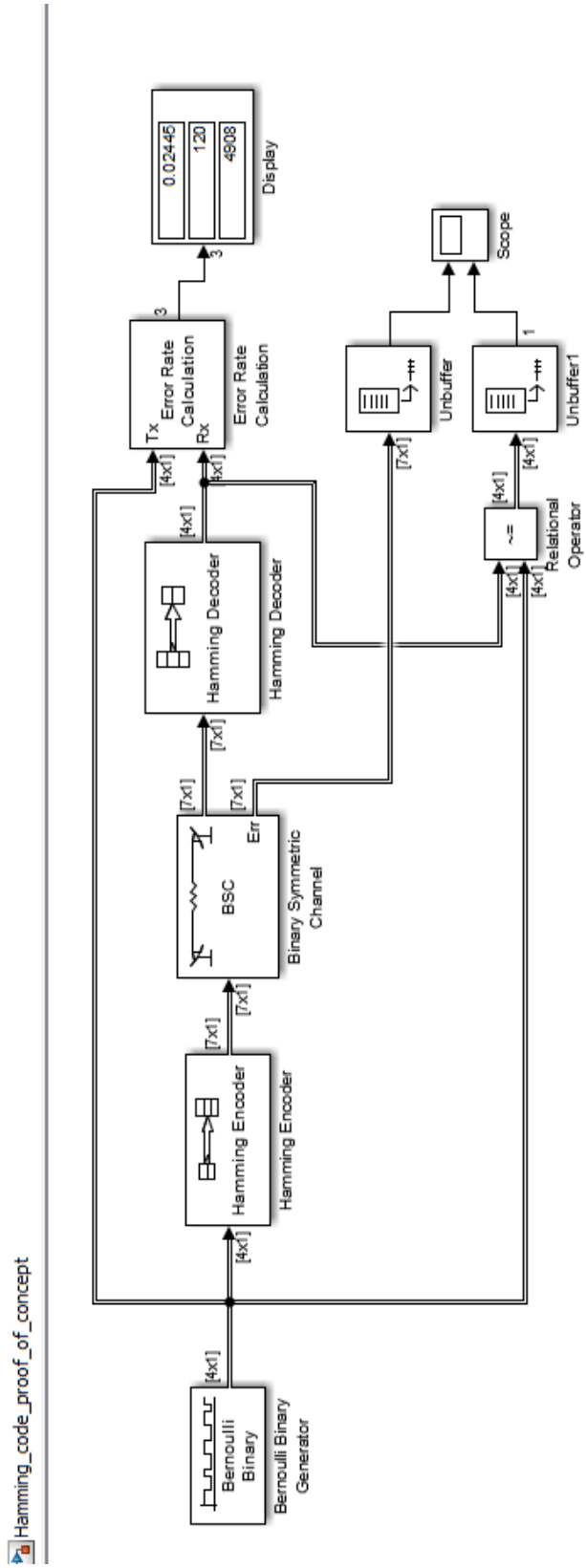
- Ahmad, S. et al., 2013. Comparison of EDAC Schemes for DDR Memory in Space Applications. *International Conference on Aerospace Science & Engineering (ICASE)*, pp.1–5.
- Bensusen, S. et al., 2013. Hyperwall Heliophysics and Space Weather.
- Bentoutou, Y., 2012. A Real Time EDAC System for Applications Onboard Earth Observation Small Satellites. *IEEE transaction on aerospace and electronic systems*, 48(1), pp.648-657.
- Bilal, Y., Khan, S.A. & Khan, Z.A., 2013. A Refined Four-Dimensional Parity Based EDAC and Performance Analysis Using FPGA. *International Conference on Open Source Systems and Technologies (ICOSST)*, pp.81–86.
- Chitode, J. S., 2009. *Digital Communication* Second, Technical Publication Pune.
- Choi, C.Q., 2011. Software Glitch Blamed for Turning Satellite Into Space Zombie.
- Cornell, J.C., Conradie, J.L. & Fourie, D.T., 1992. Beamlines for a Second Injector Cyclotron at NAC. *13th International Conference on Cyclotrons and their Applications*, pp.621–624.
- CPUT, F., 2017. ZACUBE-1 - French South African Institute of Technology.
- CPUT, F., 2016. ZACUBE-2 - French South African Institute of Technology.
- D. Heynderickx, 2002. Review on modelling of the radiation belts. *Matter, anti-matter and dark matter*, pp.87–96.
- ECSS, 2008. Space engineering - ECSS-E-ST-10-12C, (November).
- Gil, P. et al., 2014. Modified Hamming Codes to Enhance Short Burst Error Detection in Semiconductor Memories. *2014 Tenth European Dependable Computing Conference*, pp.62–65.
- Gschwind, M. & Salapura, V., 2014. Optimizing VHDL code for FPGA targets, pp.1–13.
- Halbert, 2006. Single Event Effects.
- Hamming, R.W., 1950. Error Detecting and Error Correcting codes. *The Bell System Technical Journal*, XXIX(2).
- Holbert, K.E., 2007. Space Radiation Environmental Effects.
- Hosamani, R. & Karne, A.S., 2014. Design and Implementation of Hamming Code on FPGA using Verilog. *International Journal of Engineering and Advanced Technology (IJEAT)*, 4(2), pp.180–184.
- Intel, 2018. Stratix Series FPGA Fracturable Look-Up Table Logic Structure.
- IThemba, Cyclotron Parameters.
- James Ziegler, 2013. James Ziegler - SRIM & TRIM.
- JEDEC, 2017. *Test Procedures for the Measurement of Single-Event Effects in Semiconductor Devices from Heavy Ion Irradiation*, JEDEC Solid State Technology Association 2017.

- JEDEC, 2013. *Test standard for the measurement of proton radiation single event effects in electronic devices*, JEDEC Solid State Technology Association 2013.
- Jindal, V., 2006. Design of Hamming code using Verilog HDL. *Electronics for you*, (February), pp.94–96.
- Kanemasu, M., 1999. Golay Codes. *MIT Undergraduate Journal of Mathematics*, pp.95–100.
- Keith Campbell, 2012. Engineering News - Sumbandila provided lessons for next SA satellite. *Creamer Media's*.
- Langford, M., 2014. Space Radiation Analysis Group - NASA, JSC. *NASA, JSC*.
- Maki, A., 2009. 2-1-5 Space Radiation Effect on Satellites. *Journal of the National Institute of Information and Communications Technology*, 56(1-4), pp.49–55.
- Malek, M., Coding Theory Hadamard Codes. *California State University, East Bay*, pp.1–8.
- Metra, C., Riccb, B. & Bologna, U., 1995. Novel Berger Code Checker, pp.287–295.
- Miller, I., 2012. Geomagnetism 2012 - The Sedona effect. *Sedonanomalies*.
- Morgan, S.P., 1998. Richard Wesley Hamming (1915 - 1998). *Notices of the AMS*, 45(8), pp.972 – 977.
- NASA/SP, 2012. NASA Thesaurus Volume 1 - Hierarchical Listing With Definitions. *National Aeronautics and Space Administration*, 1, p.879.
- O'Neill, K., 2003. Antifuse FPGA Technology Best Option for Satellite Applications. *The Journal of Military Electronics and Computing*, December.
- Odenwald, S.F., 2001. *The 23rd Cycle: Learning to Live with a Stormy Star*, New York: Columbia University Press.
- Parvathi, P., 2015. FPGA based design and implementation of Reed-Solomon encoder & decoder for Error Detection and Correction. *Conference on Power, Control, Communication and Computational Technologies for Sustainable Growth (PCCCTSG)*, pp.261–266.
- Poolakkaparambil, M. et al., 2011. BCH Code Based Multiple Bit Error Correction in Finite Field Multiplier Circuits. *IEEE - International Symposium on Quality Electronic Design (ISQED)*, (12th), pp.615–621.
- Ramabadran, T. V., 1990. A Coding Scheme for m-out-of-n Codes. *IEEE Transactions on communications*, 38(8), pp.1156–1163.
- Reshmi, R., Joseph, S. & Praveen, U.K., 2015. EDAC by Using Orthogonal Codes. *International Journal of Advanced Research in Electronics and Communication Engineering (IJARECE)*, 4(3), pp.632–635.
- Road, H., 2015. FPGA implementation of 4d-parity based data coding technique. *IJRET: International Journal of Research in Engineering and Technology*, 4(3), pp.593–598.
- Saxena, N.R. & McCluskey, E.J., 1990. Analysis of checksums, extended-precision checksums, and cyclic redundancy checks. *IEEE Computer Society*, 39(7), pp.969–975.

- Singh, N.P. et al., 2013. RAM error detection & correction using HVD Implementation. *European Scientific Journal*, 9(33), pp.424–435.
- System Tool Kit (STK), 2017. STK - Ionizing Radiation from the South Atlantic Anomaly (SAA) Using STK SEET.
- Tawar, V. & Gupta, R., 2015. A 4-Dimensional Parity based Data Decoding Scheme for EDAC in Communication Systems. *International Journal for Research in Applied Science & Engineering Technology (IJRASET)*, 3(Iv), pp.183–191.
- Villiers, D. De & Zyl, R. Van, ZACube-2: The successor to Africa's first nanosatellite.
- Wall, J. & Macdonald, A., 1993. NASA ASIC Guide Title Page. *Jet Propulsion Laboratory California Institute of Technology and National Aeronautics and Space Administration*.
- Wallmark, J.T. & Marcus, S.M., 1962. Minimum Size and Maximum Packing Density Nonredundant Semiconductor Devices \*. *Proceedings of the IRE*, 50(3), pp.286–298.
- Zell, H., 2013. Radiation Belts with Satellites - NASA. *National Aeronautics and Space Administration*.
- Zell, H., 2015. Van Allen Probes Mission Overview - NASA. *National Aeronautics and Space Administration*.

# APPENDICES

## 8.1 Simulink model of Hamming [7, 4, 3] (proof of concept)



## 8.2 Matlab code for Hamming [16, 11, 4]<sub>2</sub>

```
% Caleb Hillier
% 213183552
% Thesis matlab simulation
% Simple hamming code [16,11] for SECDED

clear all; close all; clc

% Generate the parity check matrix H, the generator matrix G, the codeword
% length n, and the message length k for the Hamming code with m = 4
[H,G,n,k] = hamngen(4)

% Add a parity column to generator matrix G
G = [mod(sum(G,2),2) G] % sums up each row, finds the remainder and
% concatenates it to the front of G
% Converts generator matrix G to non-systematic
G = [G(:,1) G(:,2) G(:,6) G(:,4) G(:,7) G(:,8) G(:,9) G(:,3) G(:,10)
G(:,11) G(:,12) G(:,13) G(:,14) G(:,15) G(:,16) G(:,5)]

% Add a parity column and a row of ones to parity check matrix H
H = [H mod(sum(H,2),2); ones(1,16)]
% Converts parity check matrix H to non-systematic
H = [H(:,1) H(:,2) H(:,6) H(:,4) H(:,7) H(:,8) H(:,9) H(:,3) H(:,10)
H(:,11) H(:,12) H(:,13) H(:,14) H(:,15) H(:,16) H(:,5)]
% Transposes parity check matrix H
H = H'

% INPUT - users choice%
msg = [ 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0] %Message block vector-change to any 4 bit
sequence

% Set-up parameters already generated by hamngen(4)
%k = 4; % # of message bits per block
%r = 4; % # of parity bits per block
%n = k + r; % # of codeword bits per block
n = 16

% ENCODER %
code = mod(msg*G,2) %Encode message - generates codeword

% ERROR INJECTOR - INTRODUCING AN ERROR (add one error to code)%
%code(1)= ~code(1);
%code(2)= ~code(2);
%code(3)= ~code(3);
%code(4)= ~code(4); %Pick one, comment out others
%code(5)= ~code(5);
%code(6)= ~code(6);
%code(7)= ~code(7);
%code(8)= ~code(8);
%code(9)= ~code(9);
%code(10)= ~code(10);
%code(11)= ~code(11);
%code(12)= ~code(12); %Pick one, comment out others
%code(13)= ~code(13);
%code(14)= ~code(14);
%code(15)= ~code(15);
%code(16)= ~code(16);
recd = code %Received codeword with error

% DECODER%
```

```

syndrome = mod(recd * H,2)           %Syndrome = remainder of ((message with
                                     %error * Parity-check matrix)/2)
                                     %Returns the remainder after division
                                     %Result always positive as divisor is
                                     %positive

% CHECKS FOR NO ERRORS %
if syndrome == [0 0 0 0 0]
    disp('There are no errors in the received codeword');
else

% CHECKS FOR DOUBLE ERRORS %
syndrome_dec = bin2dec(num2str(syndrome))
if mod(syndrome_dec,2) == 0           %A Double error has occurred if syndrome
                                     %is even
    disp('Two errors have been detected in the received code word');
else

% DETECTION AND CORRECTION OF SINGLE BIT ERRORS %
%Find position of the error in codeword (index)
%index = 100;
find = 0;                             %Initialise variable
for ii = 1:n                           %Run if statement for full length of code
                                     %word
    if ~find                             %Run loop till find = 1
        errvect = zeros(1,n);           %Create vector containing only zeros same
                                     %size as code word
        errvect(ii) = 1;                %Please 1 in position being tested
        search = mod(errvect * H,2);     %Search = remainder of ((test
                                     %vector * Parity-check matrix)/2)
                                     %Returns the remainder after
                                     %division
        if search == syndrome           %Run if loop if search and syndrome vector
                                     %are the same
            find = 1;                   %Set find to 1 to exit previous if statement
            index = ii;                 %Stores position of error in variable index
        end                             %Exit if loop
    end                                 %Exit if loop
end                                     %Exit for loop

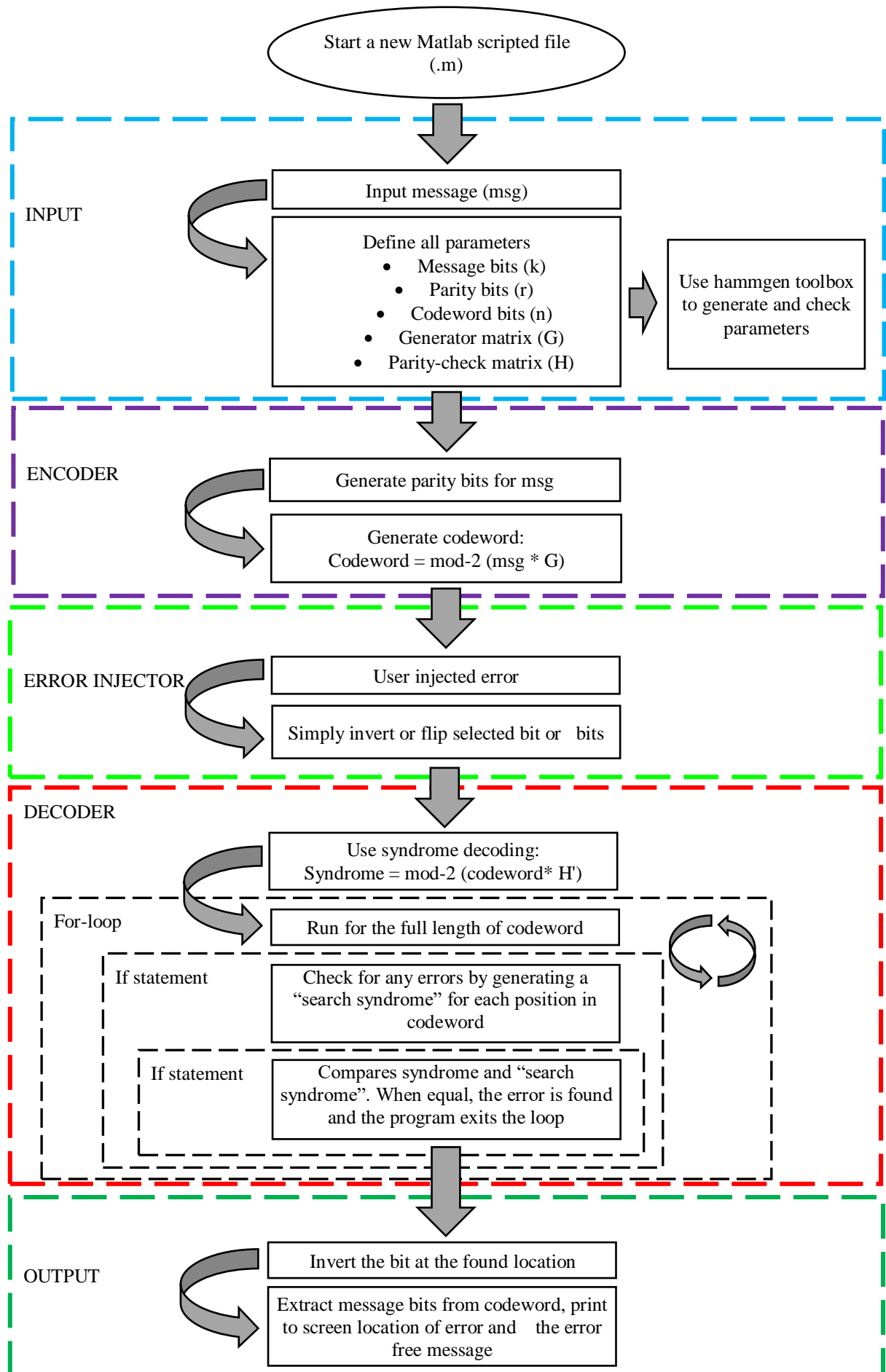
%Display position of error
disp(['Position of error in codeword = ',num2str(index)]);
correctedcode = recd;
correctedcode(index) = mod(recd(index)+1,2); %Corrected codeword

%Strip off parity bits
msg_decoded = correctedcode;
a = msg_decoded(3);
b = msg_decoded(5:7);
c = msg_decoded(9:15);
msg_decoded = cat(2,a,b,c) end end

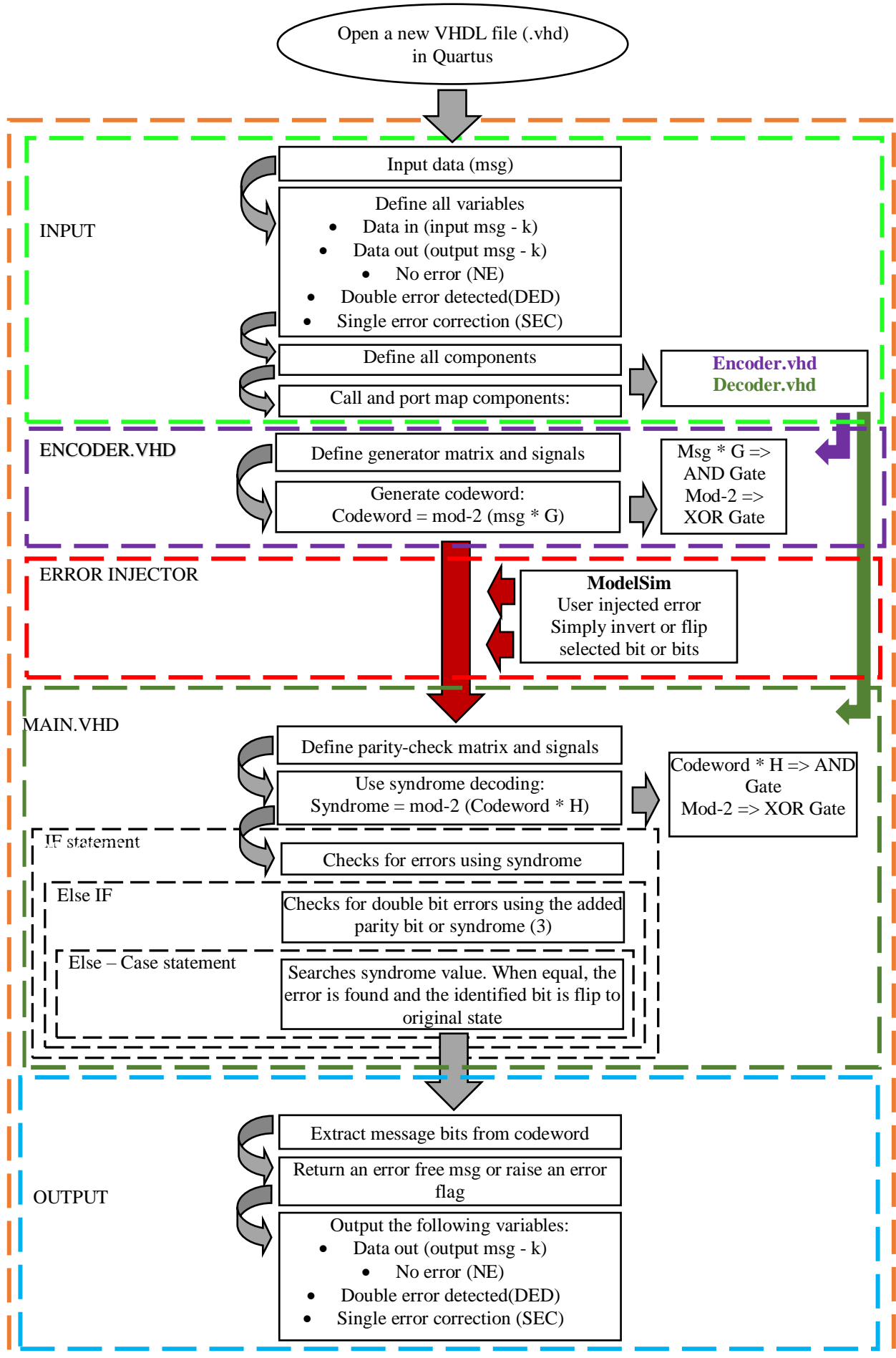
```



### 8.3 Explanation and code flow of Hamming [16, 11, 4]<sub>2</sub> in Matlab



## 8.4 Explanation and code flow of Hamming [16, 11, 4]<sub>2</sub> in VHDL



## 8.5 VHDL code for Hamming [16, 11, 4]<sub>2</sub> – main

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

entity hamming_11_16_main is
port (
    clk : in std_logic;
    clr : in std_logic;
    datain: in std_logic_vector (0 to 10);
    dataout : out std_logic_vector(0 to 10);
    NE : out std_logic;
    DED : out std_logic;
    SEC : out std_logic);
end hamming_11_16_main;

ARCHITECTURE beh OF hamming_11_16_main IS

component hammen16 is
port (
    endatain: in std_logic_vector (0 to 10);
    endataout : out std_logic_vector(0 to 15));
end component;

component hammde16 is
port (
    dedatain: in std_logic_vector (0 to 15);
    demsgout : out std_logic_vector(0 to 10);
    NE : out std_logic;
    DED : out std_logic;
    SEC : out std_logic);
end component;

signal data_to_memory: STD_LOGIC_VECTOR(0 to 15):= "0000000000000000";
signal dedatain_S: std_logic_vector (0 to 10):= "00000000000";
signal demsgout_S : std_logic_vector(0 to 10):= "00000000000";
signal NE_S : std_logic:= '0';
signal DED_S : std_logic:= '0';
signal SEC_S : std_logic:= '0';

BEGIN

    Encoded: hammen16 port map(dedatain_S,data_to_memory);
    Decoded: hammde16 port map( data_to_memory, demsgout_S, NE_S,
DED_S, SEC_S);

    process( clk, clr, data_to_memory)
    begin
```

```

        if (clr = '0') then
            dataout <= "000000000000";
            NE <= '0';
            DED <= '0';
            SEC <= '0';
        else if (rising_edge(clk)) then

            dedatain_S <= datain;
            dataout <= demsgout_S;
            NE <= NE_S;
            DED <= DED_S;
            SEC <= SEC_S;

        end if;
    end if;
end process;
END beh;

```

### 8.6 VHDL code for Hamming [16, 11, 4]<sub>2</sub> - encoder

```

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

entity hammen16 is
    port (
        endatain: in std_logic_vector (0 to 10);
        endataout : out std_logic_vector(0 to 15));
end hammen16;

ARCHITECTURE beh OF hammen16 IS

    Constant MaskP1 : std_logic_vector(0 to 10) := "11101100101";
    Constant MaskP2 : std_logic_vector(0 to 10) := "10011010111";
    Constant MaskP3 : std_logic_vector(0 to 10) := "10000000000";
    Constant MaskP4 : std_logic_vector(0 to 10) := "01101011110";
    Constant MaskP5 : std_logic_vector(0 to 10) := "01000000000";
    Constant MaskP6 : std_logic_vector(0 to 10) := "00100000000";
    Constant MaskP7 : std_logic_vector(0 to 10) := "00010000000";
    Constant MaskP8 : std_logic_vector(0 to 10) := "11010111100";
    Constant MaskP9 : std_logic_vector(0 to 10) := "00001000000";
    Constant MaskP10 : std_logic_vector(0 to 10) := "00000100000";
    Constant MaskP11 : std_logic_vector(0 to 10) := "00000010000";
    Constant MaskP12 : std_logic_vector(0 to 10) := "00000001000";
    Constant MaskP13 : std_logic_vector(0 to 10) := "00000000100";
    Constant MaskP14 : std_logic_vector(0 to 10) := "00000000010";
    Constant MaskP15 : std_logic_vector(0 to 10) := "00000000001";
    Constant MaskP16 : std_logic_vector(0 to 10) := "00110101111";

```

```

signal din1, din2, din3, din4, din5, din6, din7, din8 : std_logic_vector(0 to 10):=
"00000000000"; --to hold datain AND maskp
signal din9, din10, din11, din12, din13, din14, din15, din16 : std_logic_vector(0 to
10):= "00000000000"; --to hold datain AND maskp

```

```

BEGIN

```

```

-- din = datain * MaskP1
din1 <= endatain AND MaskP1;
din2 <= endatain AND MaskP2;
din3 <= endatain AND MaskP3;
din4 <= endatain AND MaskP4;
din5 <= endatain AND MaskP5;
din6 <= endatain AND MaskP6;
din7 <= endatain AND MaskP7;
din8 <= endatain AND MaskP8;
din9 <= endatain AND MaskP9;
din10 <= endatain AND MaskP10;
din11 <= endatain AND MaskP11;
din12 <= endatain AND MaskP12;
din13 <= endatain AND MaskP13;
din14 <= endatain AND MaskP14;
din15 <= endatain AND MaskP15;
din16 <= endatain AND MaskP16;

-- Generate encoded message: [p1 p2 d1 p4 d2 d3 d4 p8]
endataout(0) <= din1(0) XOR din1(1) XOR din1(2) XOR din1(3) XOR din1(4) XOR
din1(5) XOR din1(6) XOR din1(7) XOR din1(8) XOR din1(9) XOR din1(10);
endataout(1) <= din2(0) XOR din2(1) XOR din2(2) XOR din2(3) XOR din2(4) XOR
din2(5) XOR din2(6) XOR din2(7) XOR din2(8) XOR din2(9) XOR din2(10);
endataout(2) <= din3(0) XOR din3(1) XOR din3(2) XOR din3(3) XOR din3(4) XOR
din3(5) XOR din3(6) XOR din3(7) XOR din3(8) XOR din3(9) XOR din3(10);
endataout(3) <= din4(0) XOR din4(1) XOR din4(2) XOR din4(3) XOR din4(4) XOR
din4(5) XOR din4(6) XOR din4(7) XOR din4(8) XOR din4(9) XOR din4(10);
endataout(4) <= din5(0) XOR din5(1) XOR din5(2) XOR din5(3) XOR din5(4) XOR
din5(5) XOR din5(6) XOR din5(7) XOR din5(8) XOR din5(9) XOR din5(10);
endataout(5) <= din6(0) XOR din6(1) XOR din6(2) XOR din6(3) XOR din6(4) XOR
din6(5) XOR din6(6) XOR din6(7) XOR din6(8) XOR din6(9) XOR din6(10);
endataout(6) <= din7(0) XOR din7(1) XOR din7(2) XOR din7(3) XOR din7(4) XOR
din7(5) XOR din7(6) XOR din7(7) XOR din7(8) XOR din7(9) XOR din7(10);
endataout(7) <= din8(0) XOR din8(1) XOR din8(2) XOR din8(3) XOR din8(4) XOR
din8(5) XOR din8(6) XOR din8(7) XOR din8(8) XOR din8(9) XOR din8(10);
endataout(8) <= din9(0) XOR din9(1) XOR din9(2) XOR din9(3) XOR din9(4) XOR
din9(5) XOR din9(6) XOR din9(7) XOR din9(8) XOR din9(9) XOR din9(10);
endataout(9) <= din10(0) XOR din10(1) XOR din10(2) XOR din10(3) XOR din10(4)
XOR din10(5) XOR din10(6) XOR din10(7) XOR din10(8) XOR din10(9) XOR
din10(10);
endataout(10) <= din11(0) XOR din11(1) XOR din11(2) XOR din11(3) XOR din11(4)
XOR din11(5) XOR din11(6) XOR din11(7) XOR din11(8) XOR din11(9) XOR

```

```

    din11(10);
enddataout(11) <= din12(0) XOR din12(1) XOR din12(2) XOR din12(3) XOR din12(4)
    XOR din12(5) XOR din12(6) XOR din12(7) XOR din12(8) XOR din12(9) XOR
    din12(10);
enddataout(12) <= din13(0) XOR din13(1) XOR din13(2) XOR din13(3) XOR din13(4)
    XOR din13(5) XOR din13(6) XOR din13(7) XOR din13(8) XOR din13(9) XOR
    din13(10);
enddataout(13) <= din14(0) XOR din14(1) XOR din14(2) XOR din14(3) XOR din14(4)
    XOR din14(5) XOR din14(6) XOR din14(7) XOR din14(8) XOR din14(9) XOR
    din14(10);
enddataout(14) <= din15(0) XOR din15(1) XOR din15(2) XOR din15(3) XOR din15(4)
    XOR din15(5) XOR din15(6) XOR din15(7) XOR din15(8) XOR din15(9) XOR
    din15(10);
enddataout(15) <= din16(0) XOR din16(1) XOR din16(2) XOR din16(3) XOR din16(4)
    XOR din16(5) XOR din16(6) XOR din16(7) XOR din16(8) XOR din16(9) XOR
    din16(10);

```

END beh;

## 8.7 VHDL code for Hamming [16, 11, 4]<sub>2</sub> – decoder

```

library ieee;
use ieee.std_logic_1164.all;

entity hammde16 is
port (
    dedatain: in std_logic_vector (0 to 15);
    demsgout : out std_logic_vector(0 to 10);
    NE : out std_logic;
    DED : out std_logic;
    SEC : out std_logic);
end hammde16;

architecture beh of hammde16 is

    Constant MaskP1 : std_logic_vector(0 to 15) := "1000011001011101";
    Constant MaskP2 : std_logic_vector(0 to 15) := "0110010011110001";
    Constant MaskP3 : std_logic_vector(0 to 15) := "0010101101111000";
    Constant MaskP4 : std_logic_vector(0 to 15) := "0001110010111100";
    Constant MaskP5 : std_logic_vector(0 to 15) := "1111111111111111";

    signal dedataout : std_logic_vector(0 to 15):= "0000000000000000";
    signal synd : std_logic_vector(0 to 4):= "00000";
    signal din1, din2, din3, din4, din5: std_logic_vector(0 to 15):= "0000000000000000";
    --to hold dedatain AND maskp

begin

    din1 <= dedatain AND MaskP1;
    din2 <= dedatain AND MaskP2;

```

```

din3 <= dedatain AND MaskP3;
din4 <= dedatain AND MaskP4;
din5 <= dedatain AND MaskP5;

synd(0) <= din1(0) XOR din1(1) XOR din1(2) XOR din1(3) XOR
din1(4) XOR din1(5) XOR din1(6) XOR din1(7) XOR
din1(8) XOR din1(9) XOR din1(10) XOR din1(11) XOR
din1(12) XOR din1(13) XOR din1(14) XOR din1(15);

synd(1) <= din2(0) XOR din2(1) XOR din2(2) XOR din2(3) XOR
din2(4) XOR din2(5) XOR din2(6) XOR din2(7) XOR
din2(8) XOR din2(9) XOR din2(10) XOR din2(11) XOR
din2(12) XOR din2(13) XOR din2(14) XOR din2(15);

synd(2) <= din3(0) XOR din3(1) XOR din3(2) XOR din3(3) XOR
din3(4) XOR din3(5) XOR din3(6) XOR din3(7) XOR
din3(8) XOR din3(9) XOR din3(10) XOR din3(11) XOR
din3(12) XOR din3(13) XOR din3(14) XOR din3(15);

synd(3) <= din4(0) XOR din4(1) XOR din4(2) XOR din4(3) XOR
din4(4) XOR din4(5) XOR din4(6) XOR din4(7) XOR
din4(8) XOR din4(9) XOR din4(10) XOR din4(11) XOR
din4(12) XOR din4(13) XOR din4(14) XOR din4(15);

synd(4) <= din5(0) XOR din5(1) XOR din5(2) XOR din5(3) XOR
din5(4) XOR din5(5) XOR din5(6) XOR din5(7) XOR
din5(8) XOR din5(9) XOR din5(10) XOR din5(11) XOR
din5(12) XOR din5(13) XOR din5(14) XOR din5(15);

process(dedatain, synd, dedataout)
begin
  NE <= '0';
  DED <= '0';
  SEC <= '0';
  dedataout <= dedatain;

  if (synd = "00000") then
    dedataout <= dedatain;
    NE <= '1';
    elsif (synd(4) = '0') then
      DED <= '1';
      dedataout <= "0000000000000000";
    else

      SEC <= '1';
      case synd is

        when "10001" => dedataout(0) <= NOT dedatain(0);           --position 0
        when "01001" => dedataout(1) <= NOT dedatain(1);           --position 1

```

```

        when "01101" => dedataout(2) <= NOT dedatain(2);           --position 2
        when "00011" => dedataout(3) <= NOT dedatain(3);           --position 3
        when "00111" => dedataout(4) <= NOT dedatain(4);           --position 4
        when "11011" => dedataout(5) <= NOT dedatain(5);           --position 5
        when "10101" => dedataout(6) <= NOT dedatain(6);           --position 6
        when "00101" => dedataout(7) <= NOT dedatain(7);           --position 8
        when "01011" => dedataout(8) <= NOT dedatain(8);           --position 8
        when "11101" => dedataout(9) <= NOT dedatain(9);           --position 9
        when "01111" => dedataout(10) <= NOT dedatain(10);        --position 10
        when "11111" => dedataout(11) <= NOT dedatain(11);        --position 11
        when "10111" => dedataout(12) <= NOT dedatain(12);        --position 12
        when "10011" => dedataout(13) <= NOT dedatain(13);        --position 13
        when "00001" => dedataout(14) <= NOT dedatain(14);        --position 14
        when "11001" => dedataout(15) <= NOT dedatain(15);        --position 15
        when others => dedataout <= dedatain;

    end case;
end if;

demsgout <= dedataout(2) & dedataout(4 to 6) & dedataout(8 to 14);

end process;
end beh;

```

## 8.8 Gate level VHDL code for Hamming [16, 11, 4]<sub>2</sub> – encoder

```

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

entity hammen16 is
    port (
        endatain: in std_logic_vector (0 to 10);
        endataout : out std_logic_vector(0 to 15));
end hammen16;

ARCHITECTURE beh OF hammen16 IS

BEGIN

    endataout(0)<=      endatain(0) XOR endatain(1) XOR endatain(2) XOR endatain(4)
                      XOR endatain(5) XOR endatain(8) XOR endatain(10);
    endataout(1) <=    endatain(0) XOR endatain(3) XOR endatain(4) XOR endatain(6)
                      XOR endatain(8) XOR endatain(9) XOR endatain(10);
    endataout(2) <=    endatain(0);
    endataout(3) <=    endatain(1) XOR endatain(2) XOR endatain(4) XOR endatain(6)
                      XOR endatain(7) XOR endatain(8) XOR endatain(9);
    endataout(4) <=    endatain(1);
    endataout(5) <=                                         endatain(2);

```



```

endataout(6) <=   endatain(3);

endataout(7) <=   endatain(0) XOR endatain(1) XOR endatain(3) XOR endatain(5)
                  XOR endatain(6) XOR endatain(7) XOR endatain(8);

endataout(8) <=   endatain(4);
endataout(9) <=   endatain(5);
endataout(10) <=  endatain(6);
endataout(11) <=  endatain(7);
endataout(12) <=  endatain(8);
endataout(13) <=  endatain(9);
endataout(14) <=  endatain(10);
endataout(15) <=  endatain(2) XOR endatain(3) XOR endatain(5) XOR endatain(7)
                  XOR endatain(8) XOR endatain(9) XOR endatain(10);

END beh;

```

### 8.9 Gate level VHDL code for Hamming [16, 11, 4]<sub>2</sub> – decoder

```

library ieee;
use    ieee.std_logic_1164.all;

entity hammde16 is
  port (
    dedatain: in std_logic_vector (0 to 15);
    demsgout : out std_logic_vector(0 to 10);
    NE       : out std_logic;
    DED      : out std_logic;
    SEC      : out std_logic);
end hammde16;

architecture beh of hammde16 is

  signal dedataout : std_logic_vector(0 to 15):= "0000000000000000";
  signal mux       : std_logic_vector(0 to 10):= "00000000000";
  signal synd      : std_logic_vector(0 to 4):= "00000";
  signal noerror   : std_logic := '0';

begin

  synd(0) <= dedatain(0) XOR dedatain(5) XOR dedatain(6) XOR dedatain(9) XOR
             dedatain(11) XOR dedatain(12) XOR dedatain(13) XOR dedatain(15);

  synd(1) <= dedatain(1) XOR dedatain(2) XOR dedatain(5) XOR dedatain(8) XOR
             dedatain(9) XOR dedatain(10) XOR dedatain(11) XOR dedatain(15);

  synd(2) <= dedatain(2) XOR dedatain(4) XOR dedatain(6) XOR dedatain(7) XOR
             dedatain(9) XOR dedatain(10) XOR dedatain(11) XOR dedatain(12);

  synd(3) <= dedatain(3) XOR dedatain(4) XOR dedatain(5) XOR dedatain(8) XOR
             dedatain(10) XOR dedatain(11) XOR dedatain(12) XOR dedatain(13);

```

```

synd(4) <= dedatain(0) XOR dedatain(1) XOR dedatain(2) XOR dedatain(3) XOR
dedatain(4) XOR dedatain(5) XOR dedatain(6) XOR dedatain(7) XOR
dedatain(8) XOR dedatain(9) XOR dedatain(10) XOR dedatain(11) XOR
dedatain(12) XOR dedatain(13) XOR dedatain(14) XOR dedatain(15);

```

```

process(dedatain, synd, dedataout, mux, noerror)
begin

```

```

noerror <= (NOT synd(0)) AND (NOT synd(1)) AND (NOT synd(2)) AND (NOT
synd(3)) AND (NOT synd(4));

```

```

SEC <= ((NOT noerror) AND synd(4)) OR (noerror AND '0');

```

```

DED <= ((NOT noerror) AND (NOT synd(4))) OR (noerror AND '0');

```

```

mux(0) <= dedatain(2) XOR ((NOT synd(0)) AND synd(1) AND synd(2) AND (NOT
synd(3)));

```

```

mux(1) <= dedatain(4) XOR ((NOT synd(0)) AND (NOT synd(1)) AND synd(2)
AND synd(3));

```

```

mux(2) <= dedatain(5) XOR (synd(0) AND synd(1) AND (NOT synd(2)) AND
synd(3));

```

```

mux(3) <= dedatain(6) XOR (synd(0) AND (NOT synd(1)) AND synd(2) AND (NOT
synd(3)) );

```

```

mux(4) <= dedatain(8) XOR ((NOT synd(0)) AND synd(1) AND (NOT synd(2))
AND synd(3));

```

```

mux(5) <= dedatain(9) XOR (synd(0) AND synd(1) AND synd(2) AND (NOT
synd(3)) );

```

```

mux(6) <= dedatain(10) XOR ((NOT synd(0)) AND synd(1) AND synd(2) AND
synd(3));

```

```

mux(7) <= dedatain(11) XOR (synd(0) AND synd(1) AND synd(2) AND synd(3));

```

```

mux(8) <= dedatain(12) XOR (synd(0) AND (NOT synd(1)) AND synd(2)
AND synd(3));

```

```

mux(9) <= dedatain(13) XOR (synd(0) AND (NOT synd(1)) AND (NOT synd(2))
AND synd(3));

```

```

mux(10) <= dedatain(14) XOR ((NOT synd(0)) AND (NOT synd(1)) AND (NOT
synd(2)) AND (NOT synd(3)));

```

```

demsgout(0) <= (dedatain(2) AND noerror) OR ( ( (synd(4)) AND mux(0)) OR (NOT synd(4)
AND '0') ) AND (NOT noerror) );

```

```

demsgout(1) <= (dedatain(4) AND noerror) OR ( ( (synd(4)) AND mux(1)) OR (NOT synd(4)
AND '0') ) AND (NOT noerror) );

```

```

demsgout(2) <= (dedatain(5) AND noerror) OR ( ( (synd(4)) AND mux(2)) OR (NOT synd(4)
AND '0') ) AND (NOT noerror) );

```

```

demsgout(3) <= (dedatain(6) AND noerror) OR ( ( (synd(4)) AND mux(3)) OR (NOT synd(4)
AND '0') ) AND (NOT noerror) );

```

```

demsgout(4) <= (dedatain(8) AND noerror) OR ( ( (synd(4)) AND mux(4)) OR (NOT synd(4)
AND '0') ) AND (NOT noerror) );

```

```

demsgout(5) <= (dedatain(9) AND noerror) OR ( ( (synd(4)) AND mux(5)) OR (NOT synd(4)
AND '0') ) AND (NOT noerror) );

```

```

demsgout(6) <= (dedatain(10) AND noerror) OR ( ( (synd(4)) AND mux(6)) OR (NOT synd(4)
AND '0') ) AND (NOT noerror) );

```

```

AND '0') ) AND (NOT noerror) );

demsgout(7) <= (dedatain(11) AND noerror) OR ( ( ((synd(4)) AND mux(7)) OR (NOT synd(4)
AND '0') ) AND (NOT noerror) );
demsgout(8) <= (dedatain(12) AND noerror) OR ( ( ((synd(4)) AND mux(8)) OR (NOT synd(4)
AND '0') ) AND (NOT noerror) );
demsgout(9) <= (dedatain(13) AND noerror) OR ( ( ((synd(4)) AND mux(9)) OR (NOT synd(4)
AND '0') ) AND (NOT noerror) );
demsgout(10) <= (dedatain(14) AND noerror) OR ( ( ((synd(4)) AND mux(10)) OR (NOT
synd(4) AND '0') ) AND (NOT noerror) );

NE <= noerror;

end process;
end beh;

```

### 8.10 VHDL code for Hamming [16, 11, 4]<sub>2</sub> - testbench

```

-----
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.std_logic_unsigned.all;           -- ADD
USE ieee.numeric_std.ALL;

entity hamming_11_16_main_tb is
end hamming_11_16_main_tb;

architecture bhv of hamming_11_16_main_tb is

component hamming_11_16_main is
port (
  clk : in std_logic;
  clr : in std_logic;
  datain: in std_logic_vector (0 to 10);
  dataout : out std_logic_vector(0 to 10);
  NE : out std_logic;
  DED : out std_logic;
  SEC : out std_logic);
end component;

--Inputs
signal clk : std_logic:= '0';
signal clr : std_logic:= '0';
signal datain : std_logic_vector(0 to 10):= "00000000000";

--Outputs
signal dataout : std_logic_vector(0 to 10);
signal NE : std_logic;
signal DED : std_logic;

```

```

signal SEC : std_logic;

constant clk_period : time := 2 ns;
BEGIN

    uut: hamming_11_16_main PORT MAP (
        clk => clk,
        clr => clr,
        datain => datain,
        dataout => dataout,
        NE => NE,
        DED => DED,
        SEC => SEC);

    -- Clock process definitions (clock with 50% duty cycle is generated here)
    clk_process :process
    begin
        clk <= '0';
        wait for clk_period/2; --for 1 ns signal is '0'.
        clk <= '1';
        wait for clk_period/2; --for next 1 ns signal is '1'.
    end process;

    stim_proc: process
    begin

        datain <= "11111000000";
        wait for 100ns;

    end process;
END

```