Cape Peninsula
University of Technology

CHANNEL CODING ON A NANO-SATELLITE PLATFORM

by

ANGELA – TAFADZWA SHUMBA

Thesis submitted in partial fulfilment of the requirements for the degree

Master of Engineering: Electrical Engineering

in the Faculty of Engineering

at the Cape Peninsula University of Technology

Supervisor: Dr Y. Blanchard
Co-supervisor: Prof. R. van Zyl

Bellville
November 2017

DECLARATION

I, Angela – Tafadzwa Shumba, declare that the contents of this dissertation/thesis represent 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.

Signed **Date**

ABSTRACT

The concept of forward error correction (FEC) coding introduced the capability of achieving near Shannon limit digital transmission with bit error rates (BER) approaching 10^{-9} for signal to noise power (E_b/N_o) values as low as 0.7. This brought about the ability to transmit large amounts of data at fast rates on bad/noisy communication channels. In nano-satellites, however, the constraints on power that limit the energy that can be allocated for data transmission result in significantly reduced communication system performance. One of the effects of these constraints is the limitation on the type of channel coding technique that can be implemented in these communication systems. Another limiting factor on nano-satellite communication systems is the limited space available due to the compact nature of these satellites, where numerous complex systems are tightly packed into a space as small as 10x10x10cm. With the miniaturisation of Integrated-Circuit (IC) technology and the affordability of Field-Programmable-Gate-Arrays (FPGAs) with reduced power consumption, complex circuits can now be implemented within small form factors and at low cost. This thesis describes the design, implementation and cost evaluation of a ½-rate convolutional encoder and the corresponding Viterbi decoder on an FPGA for nano-satellites applications. The code for the FPGA implementation is described in VHDL and implemented on devices from the Artix7 (Xilinx), Cyclone V (Intel-fpga), and Igloo2 (Microsemi) families. The implemented channel code has a coding gain of ~3dB at a BER of 10^{-3}. It can be noted that the implementation of the encoder is quite straightforward and that the main challenge is in the implementation of the decoder.

ACKNOWLEDGEMENTS

I wish to thank:

- God for being my rock and anchor always
- Prof Yves Blanchard for the patience, support, and guidance
- Prof Robert Van Zyl for the funding and support during the course of this research
- My family for their unwavering support and love
- F'SATI colleagues for providing assistance every time I needed it especially Dr. Ifriky Tadadjeu Sokeng
- The CPUT faculty of engineering for the 2015 academic year funding

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.

GLOSSARY

Terms/Acronyms/Abbreviations	Definition/Explanation
ACS	Add compare Select
AIS	Automatic Identification System
APP	A posteriori Probability
AR4JA	Accumulate, Repeat-by-4, and Jagged Accumulate
AWGN	Additive white Gaussian Noise
BCJR	An algorithm which provides posteriori probability estimates for each bit in a codeword based on a received signal according to the constraints imposed by the code structure named after its inventors, Bahl, Cocke, Jelinek & Raviv.
BER	Bit Error Rate
Block Encoding	A one-to-one transformation of sequences of length k of elements of a source alphabet to sequences of length n of elements of a code alphabet with n>k.
BMU	Branch metric
BPSK	Binary Phase Shift Keying
CCSDS	Consultative Committee for Space Data Systems
Channel	The medium used to transmit signals between the transmitter and the receiver.
Circulant	A square matrix where each row is a one element cyclic shift to the right of the preceding row.
Code rate	The average ratio of the number of binary digits at the input of an encoder to the number binary digits at its output.
Codeblock	A sequence of n symbols obtained as a result of block encoding. This is the result of encoding a sequence of k information symbols.
Coding gain	The difference between the SNR of an uncoded system and a coded system required to reach the same BER level when error correction is implemented.
Concatenation	The use of two or more encoders to process data sequentially with the output of one encoder used as the input of the next.
Constraint length	A term used in convolutional coding referring to the

	number of consecutive input bits needed to determine the value of the output symbols at a particular time.
COTS	Commercial off the shelf
CPUT	Cape Peninsula University of Technology
CubeSat	A 10cm cube satellite with a mass of up to 1.33kg
Decoding lag/latency	The time it takes the decoder to give a valid output after receiving its first input
Dynamic power	In FPGA design power analysis the dynamic power is the fluctuating power as a design is run. It represents the amount of power generated by the switching user logic and routing.
E_b/N_o	A communication measure of efficiency defined as the ratio of received energy per bit to noise required to achieve a specified bit error rate, also known as the signal to noise ratio per bit (SNR)
F'SATI	French – South Africa Institute of Technology
FEC	Forward error correction
FER	Frame Error Rate
FPGA	Field programmable gate array
Interleaving	The process of arranging sequential data in a non-contiguous manner to make it more resilient to burst errors.
JPL	Jet Propulsion Lab
LDPC	Low-Density Parity Check
MDA	Maritime Domain Awareness
Nano-satellite	A satellite with a mass between 1 and 10kg
NASA	National Aeronautics and Space Administration
Octet	A binary word consisting of eight contiguous bits.
Packet	A unit of data used to transmit information in a communication network.
PAR	Place and route
Payload application information	Information obtained from a satellite payload
P_E	Error Probability
Protocol	A set of procedures and their enabling format conventions that define the orderly exchange of information between entities.
QPSK	Quadrature Phase Shift Keying

Quasi – cyclic	A type of cyclic codes where a cyclic shift of a codeword by 1 position results in another codeword.
Quiescent power	In FPGA design power analysis, the quiescent power is the power drawn by the device upon powering up and there is no activity in the loaded circuit.
RAM	Random Access Memory
Satellite pass	The amount of time in which the satellite is in view of the receiving station and communication is possible.
Shannon Limit	The theoretical maximum rate at which data can be transferred through a channel of a particular bandwidth and noise characteristics without error. Also known as channel capacity.
Space Link	The communication link between the satellite and the ground station or between two satellites in space.
Subsystem	In this document refers to the major self-contained systems within a satellite such as the Communication, Power, On-Board Computer (OBC), Thermal, Propulsion, Attitude Determination and Control (ADCS), and Structure subsystems
TB	Trace-back referring to the Viterbi decoder truncation length
Telemetry system	The end-to-end system of layered data handling services which exist to enable a spacecraft to send measurement information, in an error-controlled environment, to receiving elements (application processes) in space or on Earth
Transfer Frame	A data unit that has been encoded for transmission. It contains all the information (synchronisation, header, channel frame counts, end of frame, data field) required for the receiver to receive and decode it.
Trellis	The state diagram of a convolutional code structure
User	A human or machine-intelligent process which directs and analyses the progress of a space mission.
VHDL	VHSIC Hardware Description Language
VLSI	Very Large Scale Integration

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER ONE
INTRODUCTION

1.1 Background

Satellite technology has been vastly used for a number of applications, such as Earth observation, remote sensing, communication and technology demonstration, among others. However, the development and launch are costly and, therefore, a limited number of nations and entities had the ability to get involved in the space race since its conception in 1957 (Mitra, 2005).

In 1999, the CubeSat standard was developed by Prof. Jordi Puig-Suari at California Polytechnic State University and Prof. Bob Twiggs at Stanford University. This innovation revolutionised the demographic of entities involved in space technology development. The CubeSat standard introduced a satellite development method that significantly reduced mission cost and development time, allowing for the increased space access using small satellites as shown in APPENDIX A: CUBESAT LAUNCH STATISTICS (Cillibot et al., 2005), (Woellert et al., 2011).

1.1.1 CubeSat Programme at CPUT

Owing to the development of the CubeSat standard, the Cape Peninsula University of Technology (CPUT) started a satellite systems engineering postgraduate programme. The aim of this programme is human capital development in South Africa in the field of satellite technology. The postgraduate programme is offered by the French-South African Institute of Technology (F'SATI) hosted by CPUT. Out of this programme, a group of postgraduate students and staff developed ZACube-1, a 1U CubeSat dubbed TshepisoSat (Figure 1.1.1), which was launched on November 21, 2013 (CPUT: F'SATI, n.d.). As a result of the success of TshepisoSat, which is still operational to date, the second CubeSat in the ZACube-i series, ZACube-2, is under development. The proposed mission for this 3U CubeSat is focused on Maritime Domain Awareness (MDA), which will involve the tracking of ships using the Automatic Identification System (AIS) protocol. Due to the technology demonstration heritage of CubeSats, ZACube-2 will also include imagers to demonstrate fire tracking using CubeSats (de Villiers & van Zyl, 2015).

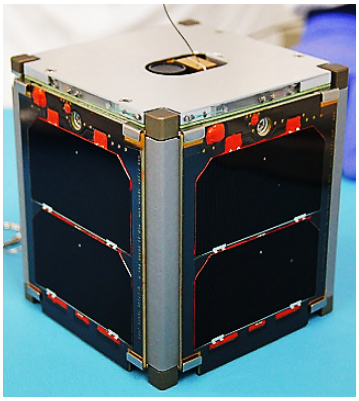


Figure 1.1.1: F'SATI CPUT CubeSat, TshepisoSat
Adapted from (Van Zyl et al., 2013)

1.1.2 Rationale

A nano-satellite, such as TshepisoSat, is made up of compactly stacked interconnected subsystems responsible for the various operations of the satellite as illustrated in Figure 1.1.2. Achieving satisfactory efficiency in any one of the subsystems would improve the efficiency of the entire satellite system (Wertz & Larson, 1999). Each of these subsystems is made up of numerous components interacting to achieve the desired functionality. As such, improving and reducing the resource requirements and consumption of the communication system adds value to the improvement of the entire satellite.

Improving the performance of a digital communication system and increasing the reliability of communication are possible at a cost, which is considered when selecting the method used to improve the system. An increase in the signal-to-noise ratio (SNR) indicates an improvement in the performance of a communication system. The SNR can be improved by increasing the transmission power or the antenna gain (Sklar & Harris, 2004). However, given the general low-budget nature of nano-satellites, the cost of increasing these parameters is very high and, therefore, this approach is generally not cost effective. The physical constraints (size) of nano-satellites also make the increase of transmit power or antenna gain unfavourable (de Milliano & Verhoeven, 2010). The use of forward error correction to improve the reliability of a communication system is, therefore, less costly than these techniques. Introducing channel coding in a communication system introduces the possibility of approaching the maximum transmission rate theoretically possible (Sklar & Harris, 2004).

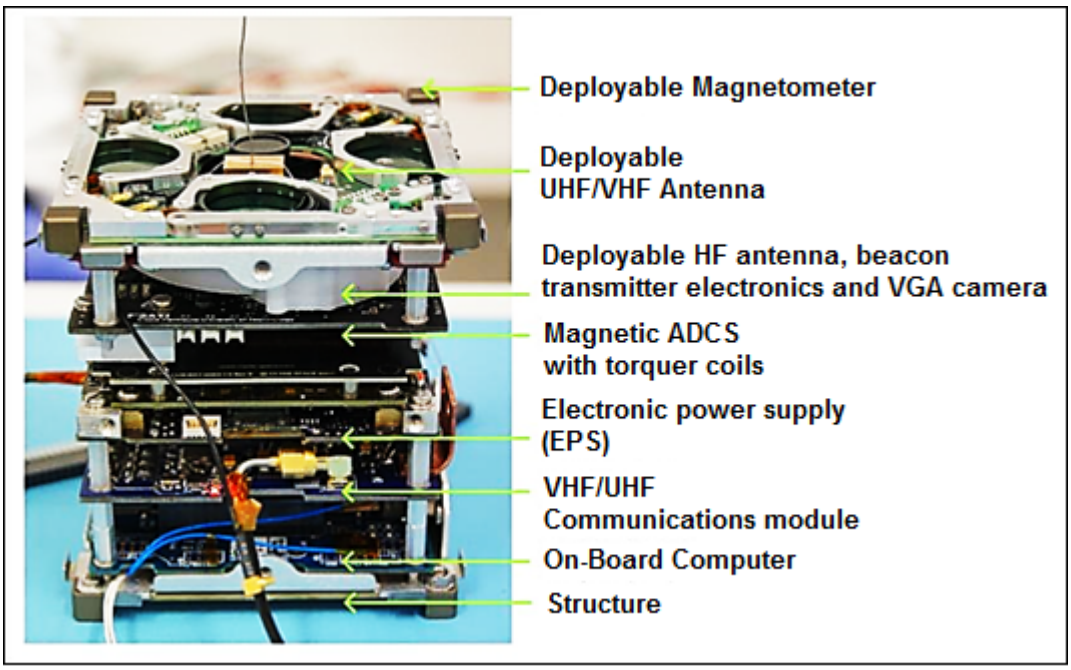


Figure 1.1.2: TshepisoSat internal layout

Adapted from (Kramer, n.d.)

1.2 Problem

Implementing nano-satellites for missions that involve the handling and transfer of large data volumes, such as the proposed ZACube-2 AIS, requires fast and reliable communication systems. In general, lower data rates result in more reliable communication than faster data rates; however, the short overpass times of satellites in low Earth orbit limit the amount of information that can be transmitted or received during a single pass. As a result, an increase in transmission rates is required in order to keep up with the need for high volume applications. On the other hand, most nano-satellite communications systems operate at relatively low data rates as a result of the size and power limitations. For this reason, high data volume applications require up- or downloading of the data over multiple passes. High bit rate communications systems on-board nano-satellites that reliably interpret the received data would increase the amount of information exchanged within one satellite pass.

1.3 Objectives

The main objective of this project is to implement a forward error correction (FEC) algorithm, which conforms to existing standards on an adaptable and reliable platform. This algorithm should also be capable of operating using the limited resources available on a nano-satellite. This means that the cost of the coding gain

achieved from the error correction should be acceptable considering available resources.

The objectives can be categorised as follows:

- implement a forward error correction encoding and decoding algorithm to improve reliability and transmission rates of payload application information;
- implement an error correction coding scheme that conforms to the international (CCSDS) standards; and
- implement an error correction technique that can be integrated with the nano-satellite communication protocol in its scarce resource environment.

1.4 Research Questions

The primary question to be addressed by this research is:

- How can a reliable and well-performing CCSDS[1] compliant forward error correction encoder and decoder be selected and implemented on a nano-satellite?

There are also a number of sub-questions that are answered in order to address the primary question:

- Are there CCSDS channel coding techniques defined for implementation on nano-satellites?
- How is FEC coding evaluated and how does that affect the criteria used to select one for implementation in a satellite communication system?
- What resources are required for FEC implementation and how can their use be optimised or minimised?
- What is the trade-off between performance and resource utilisation appropriate for a nano-satellite?

1.5 Assumptions & Delineation

In order to successfully choose a suitable coding algorithm as well as the applicable hardware, the following assumptions are made:

[1] The Consultative Committee for Space Data Systems (CCSDS) is an organisation formed by the world's major space agencies in order to discuss the common problems in the development and operation of space data systems. It develops standards and recommendation to enable interoperability and cross support between space agencies.

- the upper communication layer data shall provide a packet format for the encoder and a frame suitable for the physical communication layer shall be produced as the encoder output;
- on the decoder side, compatible data frames coming from the physical layer are received to be decoded and provided as a packet or part of a packet for the higher communication layer;
- the implementation platform (hardware) to be used is already available on the communication subsystem (e.g. FPGA, microcontrollers, etc.); and
- the hardware to be used has already undergone the required radiation testing and qualification for space application.

The extent of research is determined by the following demarcations:

- communication must be compatible with the standards used by existing ground stations; therefore, only existing channel coding techniques will be used for the realisation of the project aims and objectives;
- source coding and modulation that are part of the encoder are not part of this research and as such demodulation and source decoding are also not a part of this research; and
- data synchronisation and related procedures are not part of this study.

1.6 Document structure

This document describes the selection, design, development and testing of a forward error correction encoding and decoding algorithm for nano-satellite implementation.

The document is structured as follows:
Chapter 2 presents a background to digital communication systems, outlining the fundamentals of the error control coding concept. This chapter also contains an introduction to CubeSats.
Chapter 3 contains the comparisons of the CCSDS recommended standard error correction algorithms. The CCSDS standard highlights Convolutional, Reed-Solomon, Low-Density Parity Check (LDPC) and Turbo codes. The chapter also includes an introduction to the requirements for these techniques as well as the performance comparison according to predefined metrics.
Chapters 4 and 5 outline the design, testing and verification details of the encoder and decoder selected for implementation, respectively.
Chapter 6 contains the conclusions and recommendations for future improvement.

CHAPTER TWO
BACKGROUND

2.1 Introduction

The subject of error correction is rooted in digital communication and information theory; as such, a basic understanding of these concepts as related to forward error correction is required. The basic fundamentals of digital communication are outlined in this chapter to establish the role played by error correction in the context of a complete communication system. Finally, the chapter ends with a basic introduction to CubeSats, a special class of nano-satellites.

2.2 Digital Communication

Digital communication generally refers to a method of communication where information between a source and receiver are manipulated such that they can be represented by a sequence of discrete messages (Garg & Wang, 2005). In most cases, the digital communication system is made up of the elements shown in Figure 2.2.1 (Sweeney, 2002; Patankar, 2009).

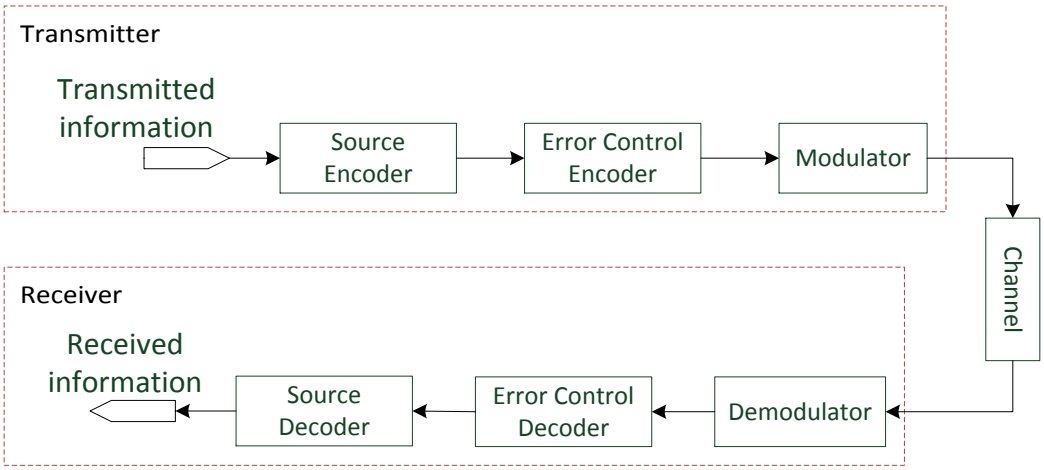


Figure 2.2.1: Digital communication system block
Adapted from (Sweeney, 2002)

The ***source encoder*** is responsible for mapping the data to be transmitted into a binary information sequence and representing it with the least possible number of bits.

The ***error control encoder*** adds redundancy to the information sequence to give the receiver the means to overcome the effects of any noise encountered by the signal during propagation.

The ***modulator*** converts the encoded data stream into an analog signal waveform, which can be propagated over a physical channel.

The ***channel*** is the medium through which the analog signal travels between the transmitter and the receiver. Noise in the channel may change the value of the encoded data being transmitted.

By using the parameters of the modulator in the system, the ***demodulator*** converts the received analog signals into a digital format, which best estimates the transmitted encoded data stream.

The ***error control decoder*** uses the data stream coming from the demodulator and estimates the original information sequence using the error control encoder characteristics.

The ***source decoder***, with knowledge of the source coding technique used in the encoder and within the limits of its code capabilities, recreates, if possible, the information sequence into the original data as given by the source.

2.2.1 Performance metrics

One of the main concerns when designing any digital communication system is to use transmit power and available bandwidth as efficiently as possible. The bandwidth efficiency can be quantified by using the ratio of data rate to signal bandwidth, whereas the power efficiency can be characterised by the probability of errors as a function of signal-to-noise ratio (SNR) (Garg & Wang, 2005).

There are several figures of merit linked to digital communication, including the SNR, which is the ratio between the average signal power and the average noise power (Sklar & Harris, 2004). E_b/N_0, which is the ratio between the bit energy and the noise spectral density as shown in equation 2.2.1 (Sklar, 2001; Sklar & Harris, 2004), is the most used digital communication system figure of merit:

$$\frac{E_b}{N_0} = \frac{S}{N} = \frac{W}{R} \qquad \textbf{2.2.1}$$

where;

S = signal power, W = bandwidth, N = noise power, and R = bit rate.

The other criterion used to evaluate digital communication system performance is the bit error probability (P_B). In many digital communication performance analyses, plots of P_B versus E_b/N_0, such as the one illustrated in Figure 2.2.2, are used for evaluating systems where a smaller E_b/N_0 signifies a more efficient process for the specified error probability (Sklar, 2001).

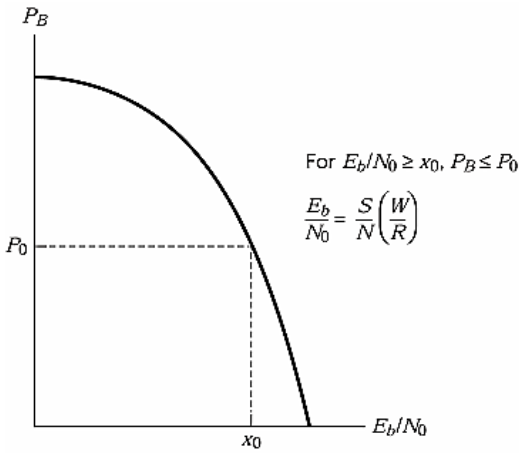


Figure 2.2.2: Typical plot of P_B vs E_b/N_0
Adapted from (Sklar, 2001)

2.2.2 Effects of noise on system performance

Errors during transmission can occur because of two classes of signal degradation:

- degradation due to reduced received signal power or increased noise or interference power; and
- degradation due to signal distortion, such as is caused by inter-symbol interference (ISI).

Figure 2.2.3 illustrates the difference between the effects of the aforementioned degradation classes. It can be deduced that the degradation due to signal distortion would require a significantly large (essentially impractical) E_b/N_0 value to achieve the desired P_B. On the other hand, the degradation due to noise power increase or

reduced signal power would require a finite E_b/N_0 value to achieve the required P_B, which is, although challenging in power limited systems, achievable (Sklar, 2001).

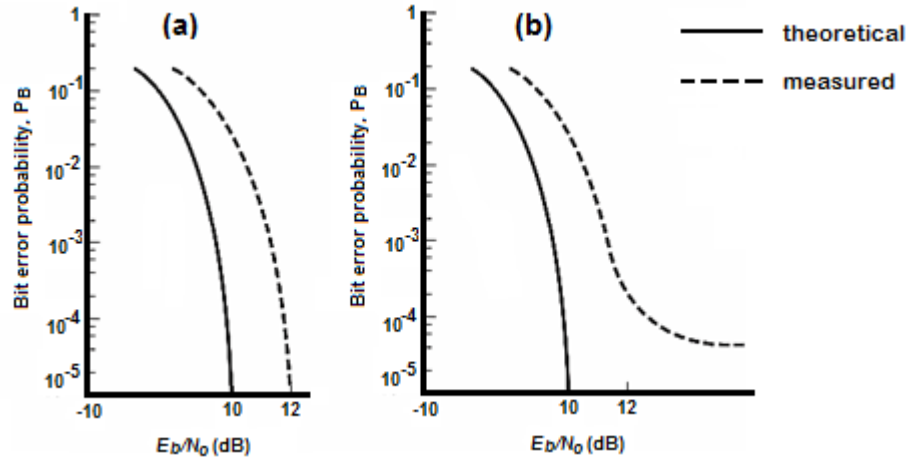


Figure 2.2.3: P_B vs E_b/N_0 for (a) degradation due to signal or noise power variation, and (b) signal distortion

Adapted from (Sklar, 2001)

2.2.3 Information theory

In response to channel noise and the inability to reproduce information at the receiving end exactly as was transmitted at the sending end, Claude Shannon in 1948 (Shannon, 1948) developed what is called *information theory*. Shannon theorised the existence of a maximum rate at which a signal could be propagated over a channel and received without the existence of errors in the received signal. This theoretical maximum rate, referred to as channel capacity/Shannon limit of an additive white Gaussian noise (AWGN) channel is given by equation 2.2.2 (Costello & Forney, 2007; Tse & Viswanath, 2005):

$$C = W\ log_2\left(1 + \frac{S}{N}\right)\ [\text{bits/sec}] \qquad \textbf{2.2.2}$$

where;
W is the bandwidth in Hz, S is the transmitting signal power in Watt, and N the added Gaussian noise power in Watt.

As a result of this notion, efforts to practically achieve transmission at channel capacity are continuing (Costello & Forney, 2007). This can be achieved by transmitting correlated information bits, which can be inferred by the receiver using

probability theorems. The correlation of the input bits is generated by a '*code'* that produces vectors belonging to a predefined alphabet, which is used at the receiving end to infer the original input bits (Shannon, 1948).

2.2.4 Forward error correction

The use of these *'codes'* as mentioned in the previous paragraph introduces the concept of forward error correction (FEC). FEC refers to the addition of redundant bits to information bits to facilitate the inference of the original information bits at the receiver end of the communication system. The FEC system consists of an encoder/decoder pair, where the encoder is responsible for creating the correlation between information bits. The decoder is, therefore, responsible for identifying any errors in received vectors and correcting them to the best of its capability (Sklar & Harris, 2004).

There are two main types of FEC codes, namely block codes and convolutional codes (Atlanta RF, 2013; Calhan et al., 2007). A block code encoder gives an output with block length n that is made up of a message block of length k and parity bits, which are used by the decoder to infer the original message. The amount of redundancy in relation to the information is usually defined using the *rate* of the code, which is defined as the ratio of the encoder input to the encoder output. The rate of block codes is, therefore, given as $R = \frac{k}{n}$ (Calhan, Ceken, & Erturk, 2007).

The block codes memory requirement is limited as the output codeword is made up of the current message block and a set of generated parity bits (Sklar & Harris, 2004; Rong et al., 2011). A convolutional encoder has m memory elements where the output code is determined by processing the input bit at the same time as the preceding m information bits.

The efficiency of these codes is measured by comparing the number of added redundant bits to the number of errors that can be corrected after the redundancy.

Coding gain can also be used as a figure of merit as it gives information on the extent of E_b/N_0 reduction due to coding implementation and is calculated as in equation 2.2.3 (Sklar & Harris, 2004).

$$G(dB) = \left(\frac{E_b}{N_0}\right)_u (dB) - \left(\frac{E_b}{N_0}\right)_c (dB)$$

where $\left(\frac{E_b}{N_0}\right)_u = \text{uncoded } E_bN_0$ and $\left(\frac{E_b}{N_0}\right)_c = \text{coded } E_bN_0$. **2.2.3**

Convolutional codes generally outperform their block code counterparts of the same complexity as a result of their continuous processing, unlike block codes that divide their input information sequence into separate blocks (Calhan et al., 2007; Viterbi, 1971; MIT, 2005).

At low values of E_b/N_0, the implementation of error correction is not beneficial to the improvement of system error performance as the decoder can only correct a finite number of errors. As the number of transmission errors exceeds the code capacity, the decoder may even introduce more errors. In such cases, implementing error correction worsens the error performance instead of improving it (Sklar, 2001).

2.3 CubeSats

A nano-satellite is any satellite with a mass less than 10kg as seen in Table 2.3.1, which shows the classes of satellites according to mass (Sweeting & Underwood, 2003).

Table 2.3.1: Satellite Classification

Class	Size
Large satellites	>1000kg
Small satellites	500 – 1000kg
Mini satellites	100 – 500kg
Microsatellites	10 – 100kg
Nanosatellites	1-10kg
Pico satellites	<1kg

A CubeSat is defined as a cuboid-shaped satellite with dimensions of 10x10x10cm and a mass of up to 1.33kg in its smallest 1-unit (1U) form factor. The CubeSat specifications were developed with the main objective of increasing space accessibility, sustaining frequent launches and reducing cost and development time (California Polytechnic State University, 2009). Due to the affordability and short development lifecycle (Sweeting & Underwood, 2003) attributed to these small satellites, some educational institutions, such as CPUT, have adopted the CubeSat as a capacity development tool.

The extent of usage of these satellites is continually increasing as a result of advancements in technology miniaturisation, which enables the implementation of sophisticated functions while using very little real estate (Sweeting & Underwood, 2003). Although the advancements in miniaturised technologies are rapid, CubeSats operate on extremely constrained power and volume budgets to perform missions of increasing complexity. As a result, any of the subsystem designs need to be efficient and conservative; yet still adequately functional. The availability of CubeSat specific space grade components is limited and when available they are costly; therefore, COTS are used in most projects (Rogers & Summers, 2010; Polaschegg, 2005).

2.4 Conclusion

This chapter has provided a brief background to the concepts related to forward error correction theory, and introduced CubeSats as a specific class of nano-satellites. This sets the foundation for the following chapter which continues the literature review, focusing on the subject of error correction techniques as recommended by the CCSDS in their *TM Synchronization and Channel Coding* recommended standard.

CHAPTER THREE
CHANNEL CODES

3.1 Introduction

One of the objectives of this project is the implementation of an error correction algorithm that conforms to the CCSDS standard. This chapter describes some of the codes, which are recommended by the CCSDS standard. It recommends four basic types of channel coding methods for satellite communication, namely Reed-Solomon, Convolutional, Turbo and LDPC codes, as well as concatenated versions of the aforementioned codes. Nano-satellite applications are not the focus area for the recommended standards; therefore, in order to choose a method for nano-satellite implementation, the recommended codes are compared through studies referencing literature and simulations to determine a suitable technique. The preferred code is selected such that the resultant transmission exhibits reasonably high data throughput with low energy per information bit (E_b) at the same BER in relation to the un-coded system. The selection of the code is also such that the resultant BER of the coded system is lower than the BER of an un-coded system with the same energy per information bit.

3.2 Code Evaluation

The code performance metrics introduced in Chapter 2, including some implementation capability parameters, are used to evaluate the performance of the coding algorithms. These comparison parameters are used to make an informed decision in the selection of a suitable algorithm for the purposes of this project. The performance and implementation capability parameters listed below are used:

- Performance parameters
    - code rate
    - coding gain
    - BER performance
- Implementation complexity parameters
    - Encoding complexity
    - Decoding complexity
    - Memory requirements
    - Latency/decoding lag
    - Implementation requirements

When simulation is used for the code evaluation, the results are obtained and analysed in the MATLAB numerical computing environment by creating virtual channels and models of complete communication scenarios. The BER analysis tool from MATLAB is also used to view and compare theoretical performance characteristics of the simulated codes.

3.2.1 Convolutional codes

Convolutional coding is identified mainly using code rate (*r*) and constraint length (*K*) with continuous data bit encoding. The CCSDS recommends a convolutional code with a constraint length of 7 and a rate of ½ denoted as a (7, ½) convolutional code, which is suitable for channels where the noise is predominantly Gaussian. The rate of ½ means that 2 encoded output symbols are produced for every input data bit. This property, therefore, implies that the bandwidth requirement is twice the amount required when no convolutional coding is implemented (CCSDS, 2012; CCSDS, 2011a).

The number of memory elements m can be obtained from the constraint length K as $m = K - 1$; therefore, the recommended code has 6 memory elements. This in turn implies that the 6 bits preceding each input must always be available when determining an output; consequently, registers large enough to store the 6 bits are required for convolutional encoding (CCSDS, 2011a; CCSDS, 2012)

3.2.1.1 Performance

Viterbi decoding of the convolutional code can be implemented using either hard decision or soft decision decoding. Hard decision decoding refers to the use of 1-bit resolution quantisation where each bit of information has two possible levels, such as binary data (Calhan et al., 2007). Soft decision decoding makes use of multiple bit quantisation, which implies that the data has an increased level of reliability as one information bit has multiple possible values in-between the 0 and 1 (CCSDS, 2012; Kumar & Gupta, 2011). The MATLAB implementation of the recommended convolutional code with continuous Viterbi decoding in comparison to the un-coded BPSK and QPSK transmissions is illustrated in Figure 3.2.1. It can be concluded that introducing the code introduces a coding gain of approximately 5 dB using soft decision decoding and 2.5 dB using hard decision decoding.

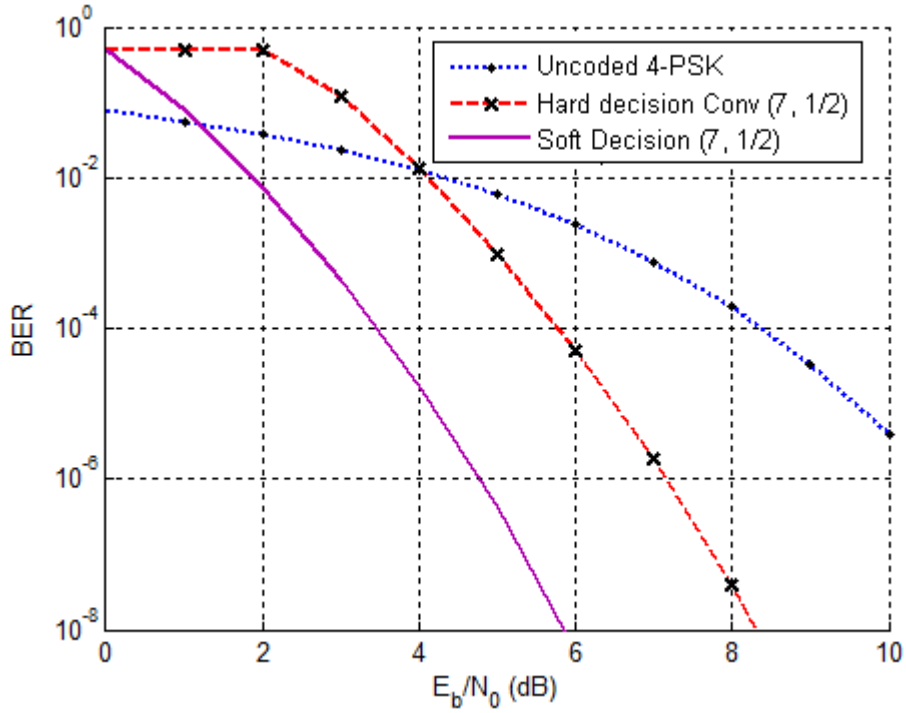


Figure 3.2.1: Simulated BER performance of convolutional code vs un-coded QPSK using MATLAB BER tool

Implementing convolutional codes has its advantages and disadvantages as listed below:

- convolutional codes are capable of handling random errors, such as the ones prevalent in an AWGN channel;
- convolutional codes are vulnerable to burst errors;
- the encoding algorithm is simple and, therefore, requires a simple circuit for implementation; and
- the decoding algorithm is, however, is slightly more complex and requires substantially more resources for implementation than the encoder.

3.2.2 Reed-Solomon (RS) codes

Reed-Solomon (RS) codes are non-binary block codes, which use multi-bit symbols to define codewords and are described using their input and output block lengths (Mitchell, 2009). The information blocks are constructed using symbols that are made up of multiple bits; when a symbol error is detected, the code corrects the entire symbol as if all the bits in the symbol were erroneous. This property makes RS codes good in correcting burst errors where binary codes falter (Mitchell, 2009; Sklar & Harris, 2004). The CCSDS recommends the (255,223) and the (255,239) RS codes, which implies that (CCSDS, 2011a):

- the input block length for the codes can be either 223 or 239;
- both recommended codes have an output block length of 255;
- the maximum number of detectable errors per block is 32 for the (255, 233) code, which allows for the correction of a maximum of 16 errors; and
- the (255, 239) code can detect a maximum of 16 errors per block and correct a maximum of 8 errors per block.

3.2.2.1 Performance

The performance characteristics of the recommended RS codes with hard decision decoding in terms of BER versus E_b/N_o as obtained from the MATLAB BER analysis tool are illustrated in Figure 3.2.2. Comparing the RS code and the un-coded BPSK BER shows that implementing RS codes introduces a coding gain of approximately 3dB at a BER of 10^{-6}.

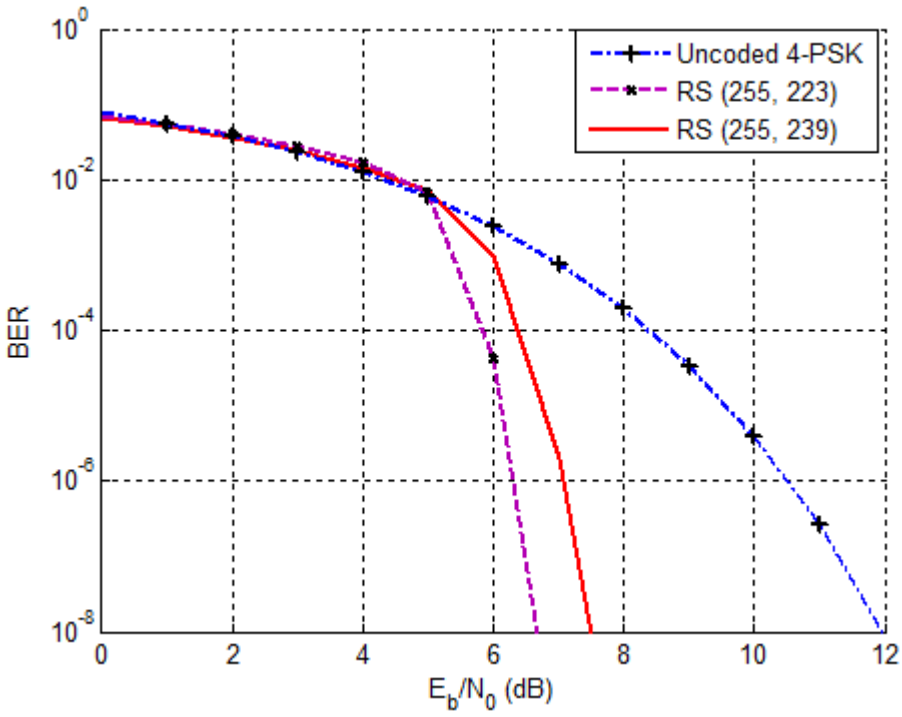


Figure 3.2.2: Simulated BER performance of RS (255,239) and RS (255,223) vs un-coded QPSK using MATLAB BER tool

Soft decision decoding algorithms for RS codes have also been explored in (Koetter & Vardy, 2003; Lu et al., 2014; Chen et al., 2013) among others; however, their complexity does not make them suitable at this time for CubeSat implementation.

Implementing RS codes has its advantages and disadvantages as listed below (Mitchell, 2009; Kumar & Gupta, 2011; Costello & Forney, 2007):

- as a result of their non-binary nature, RS codes can correct burst errors unlike binary codes such as convolutional codes;
- they are unable to handle random errors if the number of errors in one code word exceeds the code correcting capability;
- in RS error correction, the number of unknowns are twice that of binary codes; binary codes decoders only require error location, whereas RS decoders require location and error value;
- the non-binary nature of RS codes enables their decoders to be used for long block lengths with a shorter decoding time requirement than that of other codes;
- the symbol based arithmetic and numerous calculations required for error correction make the implementation of RS code more complex than binary codes; however, the RS codes provide better throughput than binary codes; and
- they are preferable for channels where the un-coded error rate is not too high as they provide significant error correction using minimal redundancy at relatively high data rates.

3.2.3 Turbo codes

Turbo codes were proposed by Berrou, Glavieux and Thitimajshima as high-performance error correcting codes with a required E_b/N_o of 0.7 for a BER of 10^{-5} with ½ code rate (Sklar, 1997; Divsalar & Pollara, 1995). These codes are a result of combined systematic terminated convolutional codes that are connected together using an interleaver to form a high-performance block code (CCSDS, 2012). The two convolutional codes are referred to as constituent codes and each of the codes contributes to the parity bit generation during encoding (Sklar, 1997). The CCSDS recommended encoder is made up of two recursive encoders with a constraint length of 5 and selectable rates of ½, ⅓, ¼ and ⅙ (CCSDS, 2011a).

With reference to the convolutional encoder characteristics as described earlier in the chapter, the constraint length of 5 means that each constituent encoder contains 4 memory elements, which store the preceding input bits that directly contribute to the encoder output.

3.2.3.1 Performance

As illustrated in Figure 3.2.3, Turbo code error performance is superior to that of convolutional and RS codes with BER values below 10^{-4} at very low energy per

information bit values. The simulated results in Table 3.2.1 contain Turbo code error performance of the recommended code rates for various block lengths with 10 decoder iterations in each case. It can be seen that the codes with higher code rates have better BER performance. It can also be concluded that the code error performance increases with block length; therefore, Turbo codes can be used when transmitting large amounts of data (CCSDS, 2012).

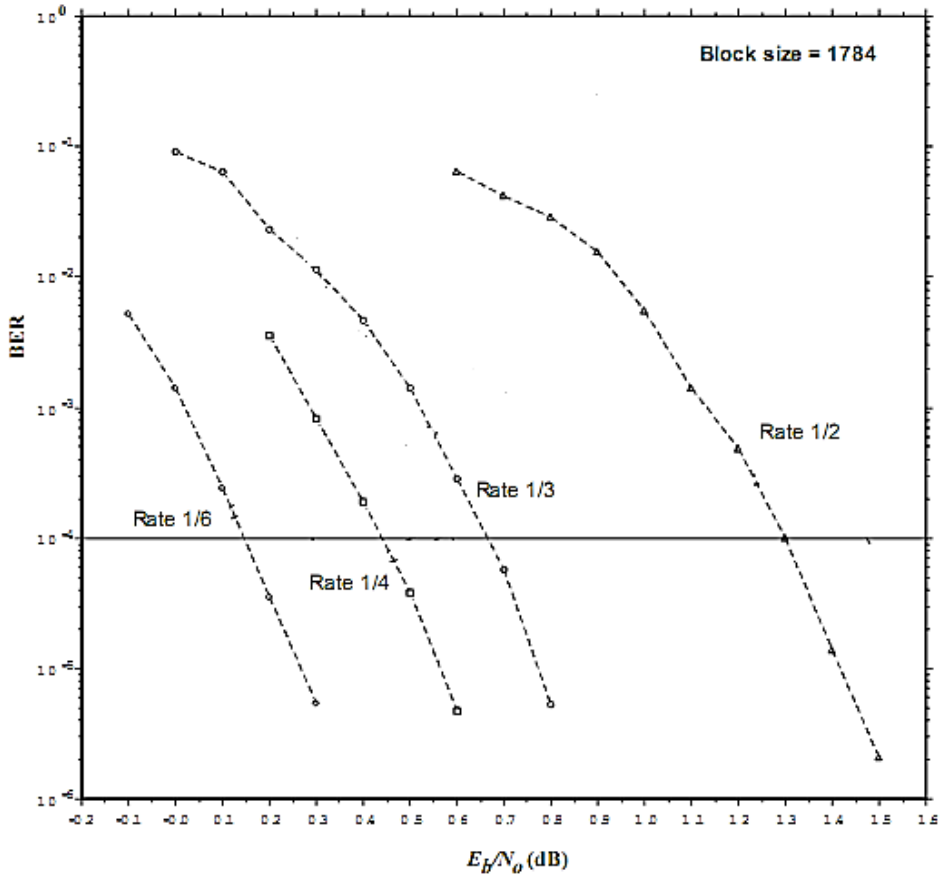


Figure 3.2.3: Turbo code BER performance, Block Size 1784 Bits, Measured from JPL DSN Turbo Decoder, 10 iterations

Adapted from (CCSDS, 2012)

Turbo codes are generally characterised by complex encoding and decoding operations. However, the coding gain within 0.8dB of the Shannon limit at a BER of 10^{-6} (O'Dea, 2013) is high enough to render the complexity acceptable for high performance applications. The implementation of Turbo codes introduces advantages and disadvantages as listed below (O'Dea, 2013; CCSDS, 2012; Madhow, 2008; Atlanta RF, 2013):

- Turbo codes are appropriate for low power communications over long distances because they exhibit low BER at low SNR, which means that transmission can be close to error-free even with very low energy signals;

- the code requires a complex decoder as a result of the multiple encoder components;
- the decoder calculations required for error correction require knowledge of the channel characteristics;
- the decoding process is iterative and, therefore, the memory requirement is very large;
- Turbo encoding introduces latency, since an entire length of information must be read before encoding begins; and
- the decoder also processes an entire block before giving an output, which introduces decoding latency.
-

Table 3.2.1: Turbo decoder BER performance approximation results for various block lengths, 10 iterations, compiled from (CCSDS, 2012)

Block length	Rate	E_b/N_o (dB) at BER 10^{-4}
1784	½	1.3
	⅓	0.66
	¼	0.43
	⅙	0.14
3568	½	1.11
	⅓	0.47
	¼	0.25
	⅙	-0.17
7136	½	0.97
	⅓	0.34
	¼	0.135
	⅙	-0.25
8920	½	0.9
	⅓	0.3
	¼	0.1
	⅙	-0.8
16384	½	0.875
	⅓	0.25
	¼	0.02
	⅙	-0.036

3.2.4 LDPC Codes

LDPC codes are a class of binary codes that can be used to obtain coding gains and good performance at low E_b/N_o values (Li et al., 2006). The CCSDS standard recommends a code known as C2 with (n, k) = (8176, 7154) and a rate of 7/8 as well as a set of 9 AR4JA (Accumulate, Repeat-by-4, and Jagged Accumulate) LDPC codes with parameters defined in Table 3.2.2 (CCSDS, 2011a; CCSDS, 2012). C2 is

optimised for near Earth satellite applications, while the other 9 codes are optimised for deep space communication (CCSDS, 2007; CCSDS, 2011a).

Table 3.2.2: Recommended AR4JA LDPC code specifications

Information block length *k*	Code block length *n*		
	Rate ½	Rate ⅔	Rate ⅘
1024	2048	1536	1280
4096	8192	6144	5120
16384	32768	24576	20480

3.2.4.1 Performance

The performance of the LDCP codes is dependent upon both the code used and the decoder used for that code. The error performance of the recommended codes is illustrated in Figure 3.2.4, which shows very low BER values at low energy per information bit. The results were obtained from a hardware simulation experiment performed at JPL as described by (Andrews et al., 2007).

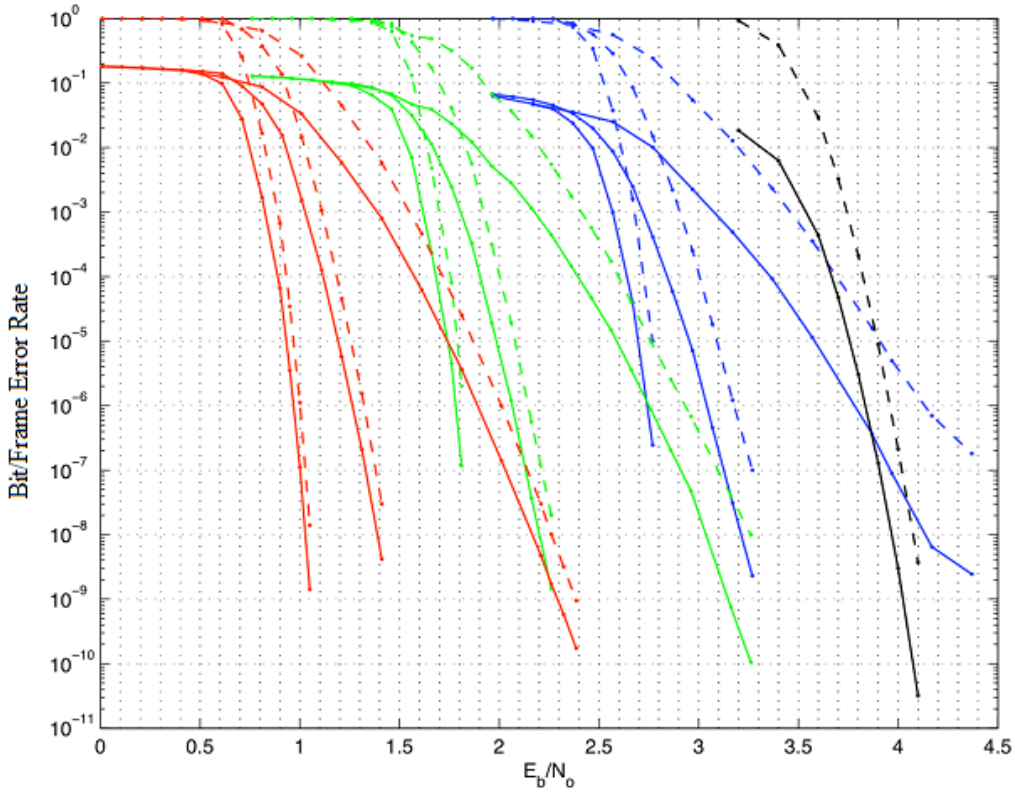


Figure 3.2.4: BER (solid) and FER (dashed) for Nine AR4JA Codes and C2, with Code Rates 1/2 (Red), 2/3 (Green), 4/5 (Blue), and 7/8 (Black); and Block Lengths *k*=16384, 4096, 1024 (Left to Right in Each Group), and 7156 (Code C2)

Adapted from (Andrews et al., 2007)

There are a number of advantages and disadvantages linked to the implementation of LDPC codes, such as the ones listed below:

- at high bit rates, LDPC codes offer smaller bandwidth expansion;
- the codes offer near Shannon limit BER performance; and
- the codes can be used for applications that require long codewords.

However, the decoding complexity of these codes makes their practical implementation challenging. The ideal mathematical functions are also not practically realisable, which results in the loss of coding gain due to approximations (Andrews et al., 2007; CCSDS, 2012).

3.3 Code Selection

3.3.1 BER performance

The performance of a channel code can be determined by using the error rate in relation to the channel resources that are required in order to achieve the desired error rate. The recommended codes as described above are compared based on AWGN channel implementation in terms of the resultant BER and the E_b/N_o required to achieve that BER. Generally, a good channel code will either reduce the BER at a fixed E_b/N_o or achieve the desired BER with a lower E_b/N_o requirement on the channel. Figure 3.3.1 shows the performance comparison, in relation to BER and E_b/N_o, of most of the channel codes described in this section. The implementation of any one of the mentioned codes offers a substantial coding gain as listed in Table 3.3.1, which compares the E_b/N_o at which the transmission can exhibit a BER of 10^{-6}.

Table 3.3.1: E_b/N_o requirements for 10^{-6} BER

Transmission	E_b/N_o (dB)
Channel Capacity	0.2
Un-coded	10.5
Convolutional	4.7
Reed Solomon	6.4
Turbo	1.1
LDPC	0.9

Implementing convolutional coding alone can provide satisfactory coding gain while requiring very little complexity for encoding and "acceptable" complexity for decoding. The RS code can also provide good coding gain; however, RS encoding and decoding are more complex than convolution in terms of hardware implementation, and the convolutional coding gain is better than that of RS coding.

This comparison concludes that the Turbo and LDPC codes offer the largest coding gain, therefore, allow for clean transmission at very low SNR. However, the recommended LDPC codes are subject to patent laws for non-CCSDS member agencies and Turbo codes are subject to general patent laws. Permission to use these codes should, therefore, be sought from the appropriate regulatory parties (CCSDS, 2012). As a result, ease of use and accessibility are considered as factors for the channel code selection.

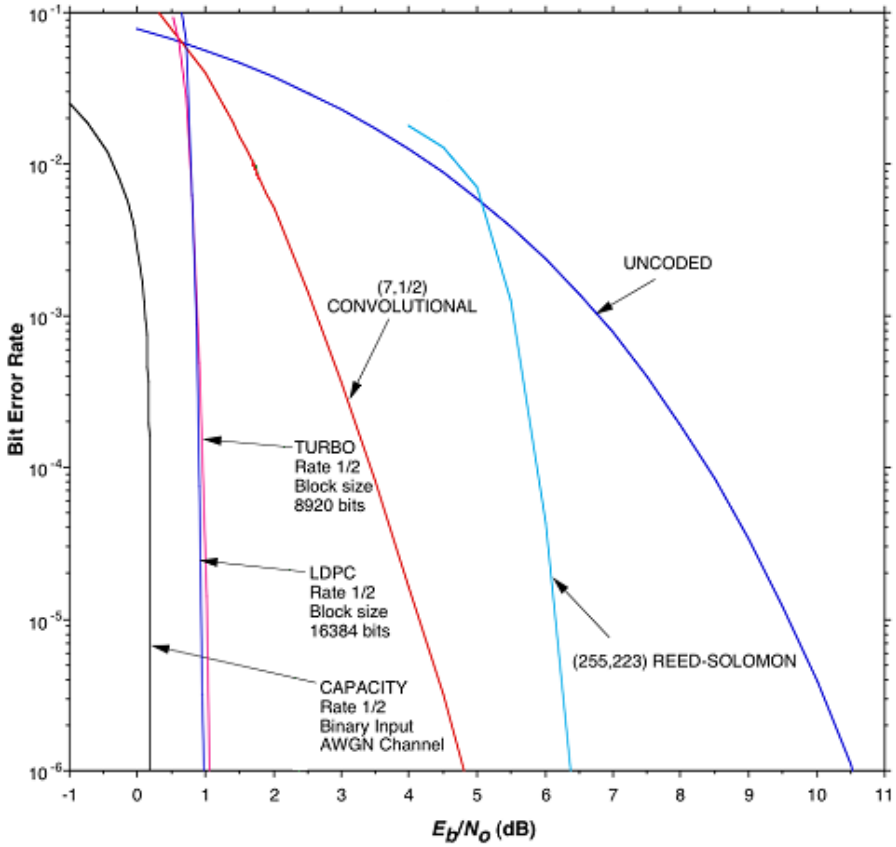


Figure 3.3.1: Performance comparison of ½ rate Convolutional, Reed-Solomon, Turbo and LDPC codes in relation to un-coded and capacity transmission

Adapted from (CCSDS, 2011)

3.3.2 Signal power vs. spectral density

Another parameter used to aid in the selection process is the power and spectral efficiency trade-off. Figure 3.3.2 shows the spectral efficiency for the recommended standard codes in relation to the required E_b/N_o. The figure shows that with strict power constraints, Turbo codes are suitable but they require large bandwidth expansion. On the other hand, when bandwidth is severely constrained, the LDPC codes are the best choice if the increased energy per bit is acceptable. Using this

metric, Turbo and LDPC codes would clearly be the obvious choice for a system within one of these extremes. However, nano-satellites are usually both bandwidth and power constrained; therefore, the ideal selection would be a code that results in an acceptable E_b/N_o with minimal bandwidth expansion.

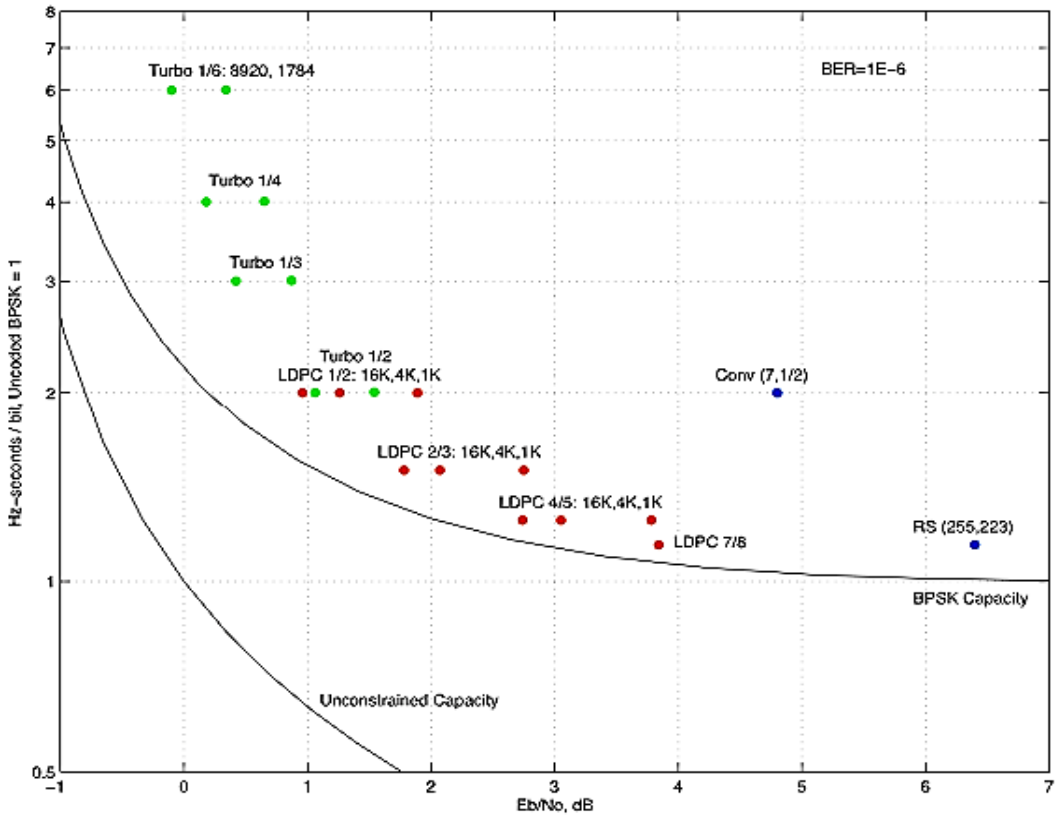


Figure 3.3.2: Power and spectral efficiency trade-off
Adapted from (CCSDS, 2012)

The left of the figure corresponds to codes that are suitable for severely power-constrained systems, while the bottom of the figure relates to codes that are suitable for bandwidth-constrained systems. An ideal code that exhibits both spectral efficiency and good E_b/N_o would be in the bottom left corner of Figure 3.3.2. But this does not exist, and a performance trade-off between spectral efficiency and E_b/N_o is necessary. The solution would be to either slightly compromise both parameters or tip the scales in favour of one according to the cost of each. The favourable LDPC and Turbo codes introduce decoder complexity, which requires significant hardware resources that are not readily available on nano-satellites. RS codes offer a spectral efficiency better than convolutional codes at the cost of a higher E_b/N_o, while convolutional codes require less E_b/N_o at the cost of spectral efficiency. The deciding factor would, therefore, be code complexity; in this case, convolutional codes would be preferred.

3.3.3 Summary

The studies in this chapter have shown that the LDPC and Turbo codes outperform the other codes in every aspect. However, their implementation comes at a cost of high complexity, and large computation and memory requirements; traits that are undesirable for hardware implementation with limited resources, especially in nano-satellites.

As stated in previously, the less complex convolutional and Reed-Solomon codes are more suitable for the desired nano-satellite application. Taking into account the error correction strengths, the complexity of the decoders, and the learning curve for grasping the concepts involved in encoding and decoding, convolutional codes have been selected over RS codes for implementation in this project. Also, in terms of coding gain, convolutional codes outperform RS codes by achieving an approximate coding gain of 5dB for BPSK modulation and at a BER of 10^{-6}, compared to 4dB for the RS codes. An introduction to the convolutional encoding and decoding process is, therefore, given in the next section. The design and development of the convolutional encoder and Viterbi decoder are outlined in Chapters 4 and 5.

3.4 Convolutional Coding

Convolutional coding has been selected as the code to be implemented in this project. This section outlines the encoding and decoding processes to be used in the design. Convolutional codes are identified using their constraint length (***K***) and rate (***r***) parameters, which determine the complexity and performance of the code. As such, the performance of a convolutional code is directly proportional to ***K*** and inversely proportional to ***r***, whereas the complexity of the code is inversely proportional to ***r*** and exponentially proportional to ***K***. In this instance, ***K*** represents the number of consecutive input bits that contribute directly to the definition of the output symbols, and ***r*** is the ratio of input bits to output symbols expressed as a fraction (O'Dea, 2013; CCSDS, 2012).

3.4.1 Encoding

The recommended CCSDS convolutional encoder requires a simple circuit for implementation as illustrated in Figure 3.4.1. The implementation involves a shift register with ***K*** stages, which outputs are connected by ***r*** modulo-2 generator vectors producing 2 outputs C_1 and C_2 that are alternately switched to the output through

switch S1. The CCSDS recommends the inversion of one of the output symbols to ensure the sufficient handling of an all “0” or an all “1” input vector, resulting in the output sequence $C_{1_1}\overline{C_{2_1}}\,C_{1_2}\,\overline{C_{2_2}}\cdots C_{1_N}\,\overline{C_{2_N}}$.

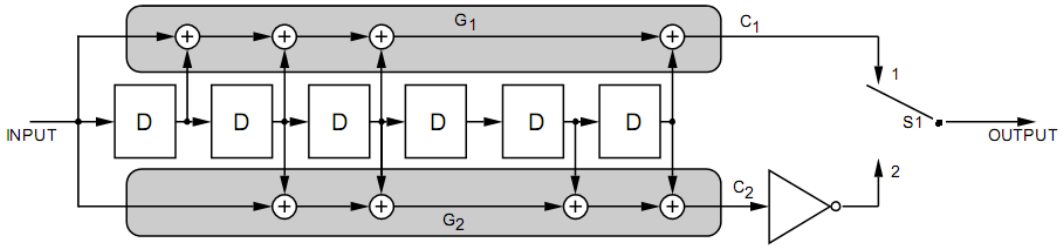


Figure 3.4.1: Recommended (7, ½) convolutional encoder block representation Adapted from (CCSDS, 2012)

For each input bit, the information in the registers is shifted to the right when the input is shifted into the left-most position. The outputs C_1 and C_2 are determined using connection vectors of length ***K*** + 1 and the encoder algorithm is described in 3 steps:

1. Initialise the memory registers with zeroes.
2. At input time instance t,
    a. calculate C_1 and C_2 using combination vector and memory cell contents as defined by generator polynomials G_1 and G_2;
    b. shift the components of the memory elements to the right;
    c. shift input into left-most memory register; and
    d. sample C_1 and C_2 using the output switch.
3. Repeat step 2 for each input time instance.

The relationships among the input bits and the states of the memory elements of the encoder, as a function of time, can be illustrated using a trellis diagram. A trellis diagram can be defined as a graphical representation of the states of the encoder with the passage of time. The trellis diagram in Figure 3.4.3 represents a simple encoder with only 2 memory elements (Figure 3.4.2) for illustration purposes, as the trellis diagram for the recommended (7, ½) CCSDS encoder is very large and complex. The trellis diagram shows all the possible states of memory elements at any given time instance in the encoding process. The unique data sequence is represented by a particular path along the trellis and this property is used by the decoder to infer the path with the highest likelihood representing the initial data sequence (Calhan et al., 2007). The dashed lines in the diagram represent a branch along the path resulting from an input of 0 and the solid lines show transitions due to an input of 1.

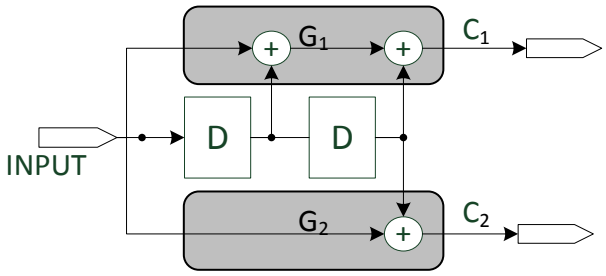


Figure 3.4.2: Simple (3, ½) convolutional encoder block representation

Adapted from (Viterbi, 1971)

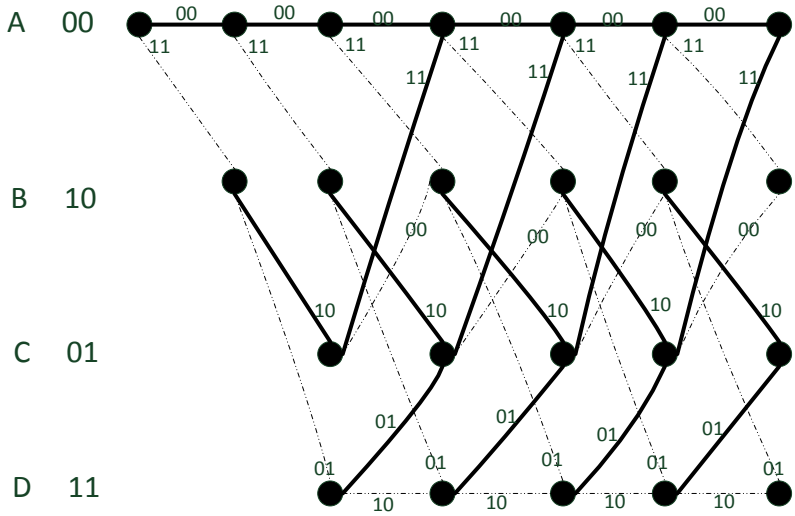


Figure 3.4.3: Convolutional code trellis diagram for ½ rate code with a constraint length of 3

Adapted from (Viterbi, 1971)

3.4.2 Decoding

Every encoder requires a decoder to interpret the encoded symbols and subsequently determine the initial contributing input. The CCSDS recommended decoder for the convolutional encoder is the Viterbi decoder, which uses a maximum likelihood algorithm to determine the initial input using the available encoded symbols. The Viterbi algorithm is used because it improves communication efficiency by 4 to 6dB at a BER of 10^{-5} (Jerrold & Jacobs, 1971).

As mentioned earlier, the Viterbi decoder can be implemented either as a hard decision decoder or a soft decision decoder. Implementing a soft decision decoder results in a higher level of decoding accuracy as illustrated in Figure 3.4.4; however,

the multiple bit quantisation increases the complexity of the practical implementation (CCSDS, 2012).

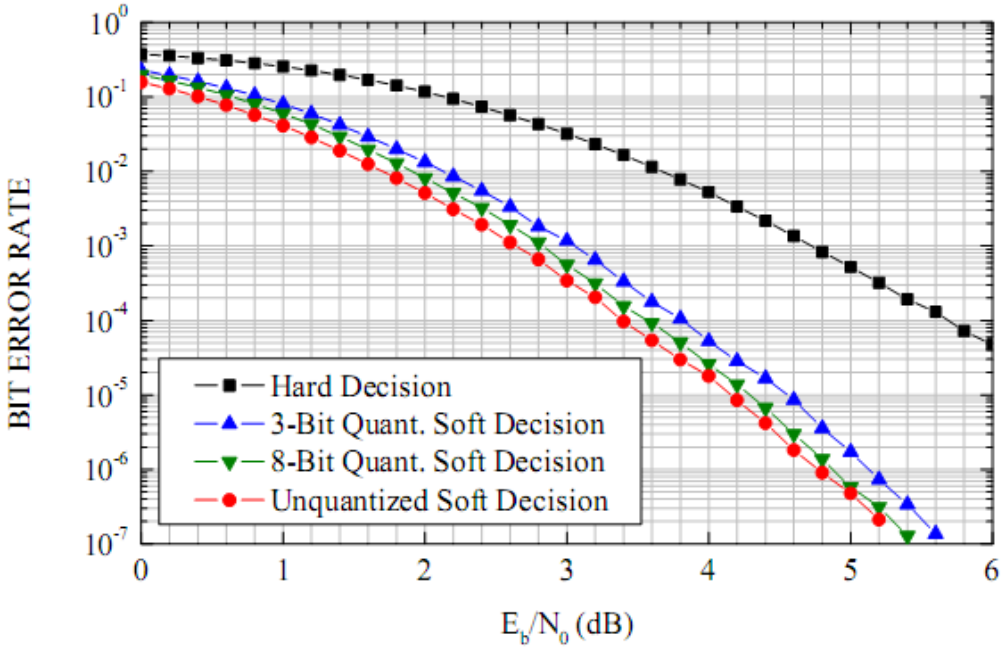


Figure 3.4.4: Convolutional coding error performance
Adapted from (CCSDS, 2012)

The decoder uses the information defined by the trellis diagram to function, and therefore, a fixed amount of memory is required to store the information (Calhan et al., 2007). The decoder is also required to compute and compare the likelihood probabilities of every possible path along the trellis. The results of this comparison are then used to determine the path with the highest likelihood of representing the original information. This also requires memory, as the path likelihood probabilities of all the paths along the trellis for the duration of the data sequence must be stored for the comparison (Morelos-Zaragoza, 2002).

Decoding a large received sequence would, therefore, require extremely large amounts of memory, which is practically unachievable. Attempting to decode the complete received sequence in one go would also result in very long latency, because all the paths along the trellis are to be compared before yielding an output. To avoid the use of large amounts of memory and high latency, the practical decoder divides the information sequence into blocks with a reasonable truncation length **D.** The recommended value of **D**, which results in negligible decoder degradation, is set at approximately 5 times the constraint length of the encoder (5***K***) (Morelos-Zaragoza, 2002). Figure 3.4.5 shows that the BER performance of a convolutional

code is also dependent on **D**. The length of **D** can be increased to a certain point where, increasing it further, would not improve the BER performance, as seen in Figure 3.4.5 (for example, setting **D** = 60 results in the same BER as the ideal case of **D** approaching infinity).

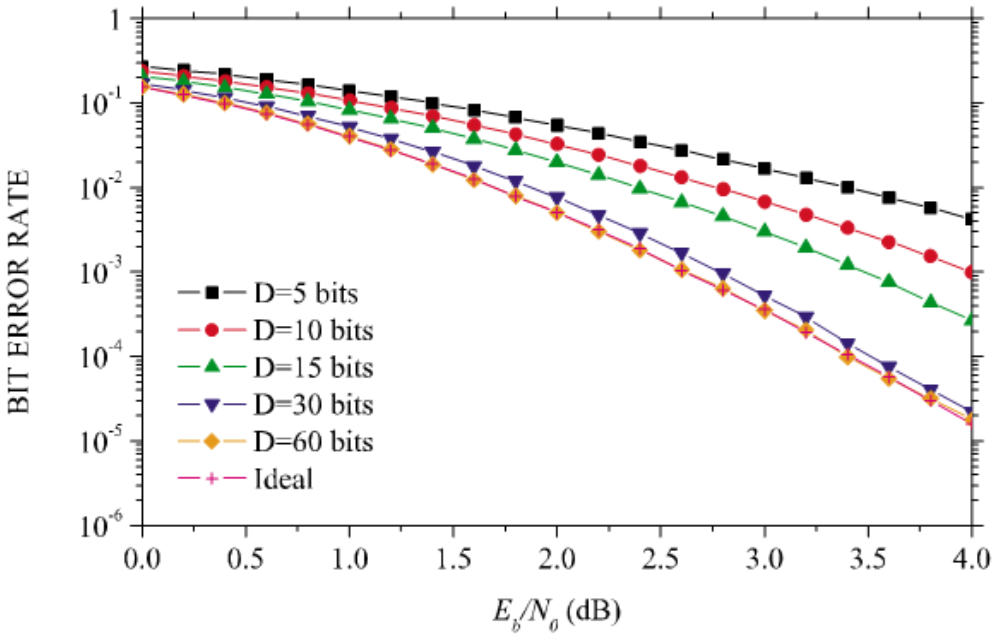


Figure 3.4.5: Truncation length (D) versus BER

Adapted from (CCSDS, 2012)

CHAPTER FOUR
CONVOLUTIONAL ENCODER

4.1 Introduction

The CCSDS standard recommends a convolutional encoder with a constraint length of 7 and a rate of ½. The encoder is made up of a shift register with 6 memory cells, which outputs are combined and manipulated by modulo-2 addition in a predetermined combination. This gives two separate data streams as illustrated in Figure 4.1.1. The figure includes the shift register stages labelled as D and the combinations of the register outputs obtained from the generator polynomials G0 and G1 for modulo-2 addition. Also shown in the figure are the input and output locations within the system as well as the direction of the flow of signals along the circuit. The encoder generates two streams of data from a single input data stream as a result of the ½ code rate. A logical inverter is seen at one of the output data streams (out_1), as recommended to ensure that the code is efficient even with an all '0' or all '1' input stream. The final stage of the encoding process is a selector switch to multiplex the two encoder outputs into one output data stream.

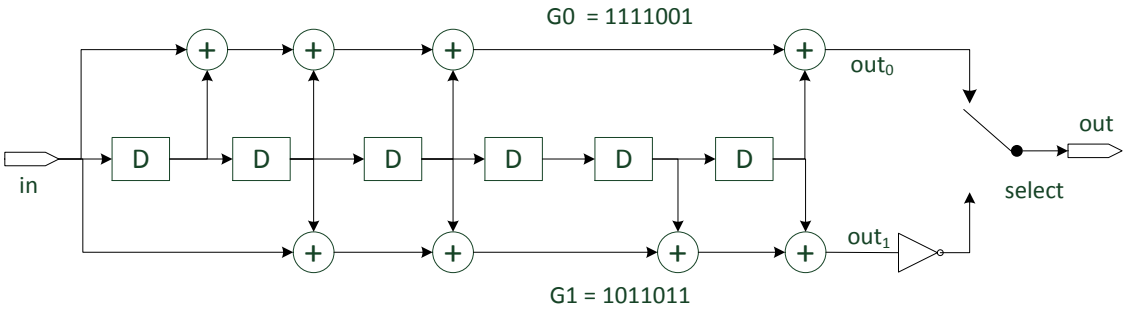


Figure 4.1.1: CCSDS recommended (7, 1/2) convolutional encoder
Adapted from (CCSDS, 2011)

4.2 VHDL Design

The encoder is to be implemented on an FPGA platform and, therefore, its architecture is to be described in VHDL and verified using the process illustrated in Figure 4.2.1.

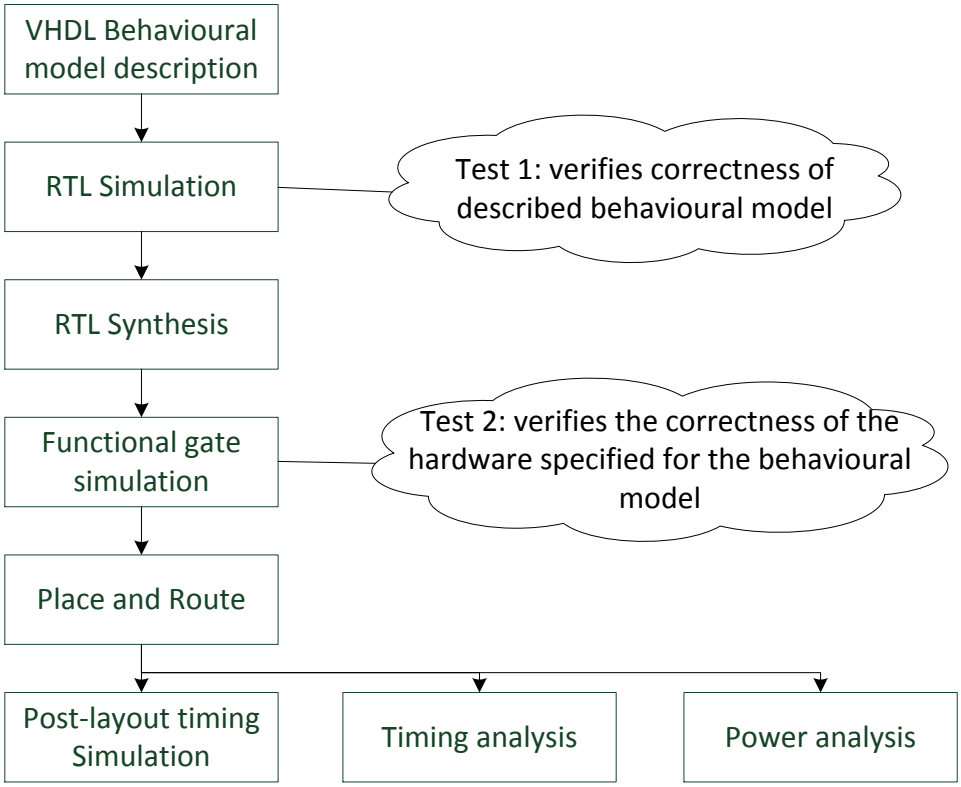


Figure 4.2.1: VHDL design procedure

Adapted from (Perry, 2002)

The VHDL implementation of the encoder in Figure 4.1.1 can be illustrated using the block diagram in Figure 4.2.2. The encoder consists of 3 main I/O signals, which are the clock, data input and data output.

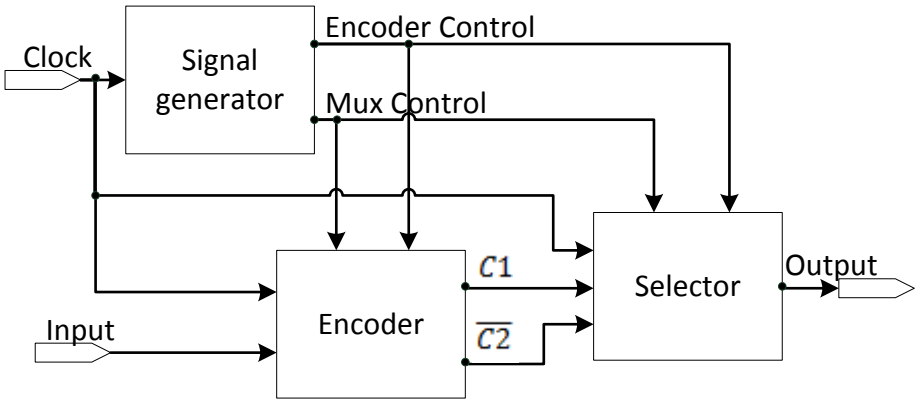


Figure 4.2.2: Encoder implementation block diagram

The encoder comprises 3 blocks/components and the roles of each block are outlined in Table 4.2.1 below. The encoder's 6 memory elements are implemented as shift registers and the modulo-2 adders are implemented as XOR logic gates.

Table 4.2.1: Encoder implementation blocks/components explained

Component	Responsibility
Signal generator	Generates the 2 signals used to synchronise the encoder operation.
Encoder	Executes all the encoder calculations illustrated in Figure 4.1.1, which includes shift register logic, XOR logic and *C2* inversion.
Selector	Implements the parallel-to-serial convertor, which enables the encoder to produce a single data output stream.

The encoder architecture is completely synchronous; therefore, all the processes and calculations are controlled by the input clock signal. Figure 4.2.3 shows the encoder timing diagram, which illustrates the associations among the encoder input and output in relation to the clock and control/synchonisation signals. The signals named encoder control (*enc_ctrl*) and mux control (*mux_ctrl*) are used to synchronise the processes within the encoder in order to ensure that all the calculations and logical operations use the correct data inputs.

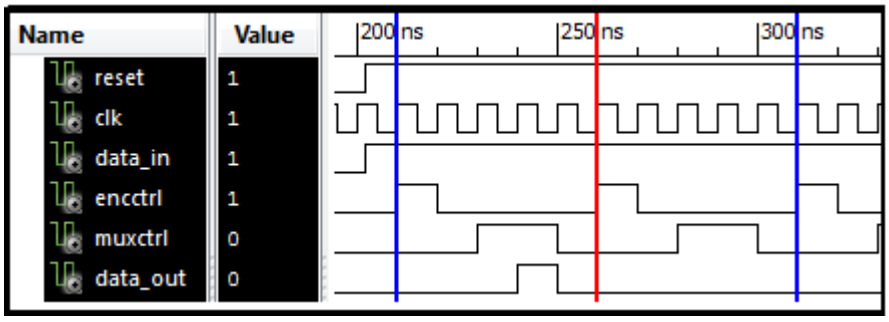


Figure 4.2.3: Encoder timing diagram

The significance and implications of the timing diagram in Figure 4.2.3 can be further visualised using the truth table described in Table 4.2.2, which shows one full encoder cycle. A complete encoder operation requires 5 clock cycles, as can be seen from the truth table. The 'X' indicates that the particular state of that signal is insignificant to the encoder operation. The 'Valid' label denotes when the input signal is used in the encoder calculations and indicates that the input should not be changed at this time. This means that the input should be made available before the indicated time instance within the encoder cycle. For the output signal, the 'Valid' label indicates when the output signal is available to be recorded. This means that the output should be recorded within the availability window indicated.

Table 4.2.2: Encoder signals truth table

Clock cycle	Clock	Encoder control	Mux Control	Input	Output
1	↑	0	0	X	X
2	↑	1	0	Valid	X
3	↑	0	0	X	X
4	↑	0	1	X	Valid $C1$
5	↑	0	1	X	Valid $\overline{C2}$

4.3 Analysis

A spreadsheet model that computes the encoder calculations for each stage is created and used to verify the behavioural correctness of the VHDL implementation. A set of randomly generated data bits is introduced into the spreadsheet model to generate the expected encoder outputs. The same input data set is then used during the behavioural simulation of the model where the output is compared to the expected values from the spreadsheet model. The MATLAB random data generator creates a text file that is used in the VHDL behavioural simulation. This verification process is illustrated in Figure 4.3.1. The spreadsheet model is used for the encoder verification as it provides a simple method of visualising and following the progression of the arithmetic and logic operations involved in the encoding process.

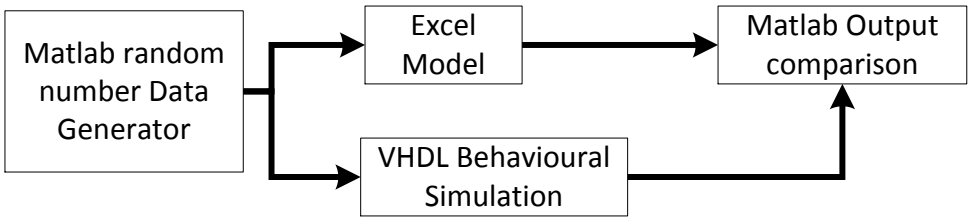


Figure 4.3.1: Encoder behavioural correctness verification

If the behavioural model simulation results match the spreadsheet model output, the VHDL design is then fed into a synthesiser, which creates the hardware netlist for a specified device. The encoder is to be implemented on hardware that exists on communication system modules to reduce the cost of implementation; therefore, netlists are created for 3 FPGA devices from different vendors. This is done to determine the cost of implementing an encoder on a system with one of these 3 FPGA devices that at this time may be found in CubeSat subsystems.

The major players in FPGA manufacturing are Xilinx and Altera, collectively controlling approximately 80% of the FPGA market value (Global Market Insights, 2016); therefore, a device is selected from the two manufacturers' low power, low-cost device families. From the Xilinx 7 series 28nm technology, the Artix-7 family has low power, low-cost devices that are suitable for cost-sensitive applications (Xilinx, Inc., 2014). A device from the Artix-7 family is, therefore, used in this project to verify the implementation of the encoder on Xilinx devices. The Altera family FPGAs are designed for different user needs, with the Cyclone family tailor-made for low power, low-cost applications (Altera Corporation, 2012). A Cyclone V device is, therefore, used to verify the implementation on Altera devices. Microsemi's IGLOO2 as of 2014 was the lowest power FPGA family in the industry (Microsemi Corporation, 2014); therefore, a device from this family is also selected for use in this project. The specific devices used are listed in Table 4.3.1.

Table 4.3.1: Test FPGA devices

Manufacturer	Family	Device	Design environment
Altera	Cyclone V	5CEBA2F17C7	Quartus II & ModelSim Altera starter edition
Microsemi	Igloo2	M2GL010T	Libero SoC v11.4 & ModelSim ME 10.3a
Xilinx	Artix7	xc7a100t-3fgg484	ISE design suite 14.7

The VHDL design is ported onto the different design platforms listed in Table 4.3.1. This process illustrated in Figure 4.3.2 is followed to obtain the required analysis results. Validation on these three platforms also brings a certain amount of confidence that the code is technology-independent and will be re-usable on any new family that may become available in the future.

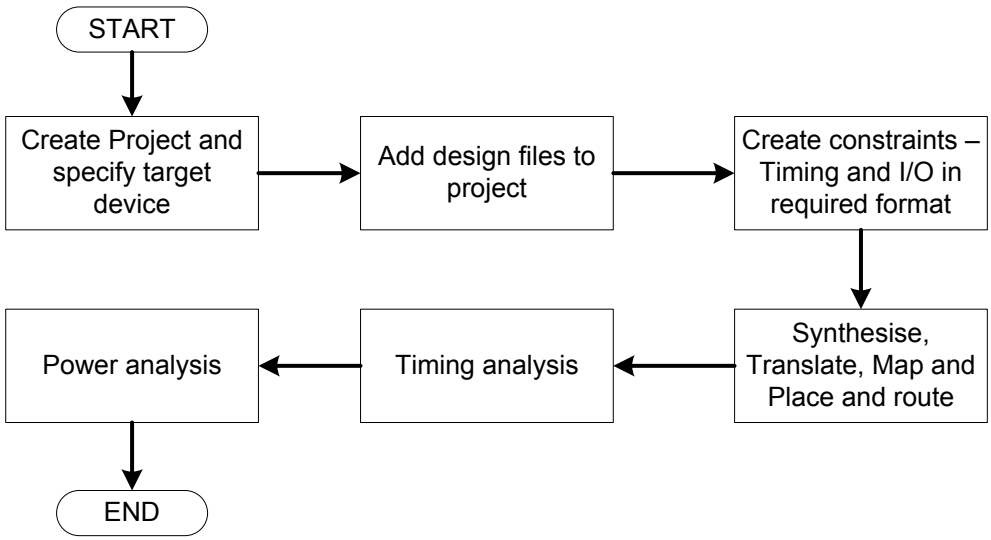


Figure 4.3.2: Encoder design verification procedure

4.3.1 Device resource usage

To determine the resource usage for the 3 test devices, the post-layout/place-and-route (PAR) reports are analysed. The resources, in this case, refer to the number of FPGA slices, look-up tables (LUT's) and global resources. Table 4.3.2 shows that the encoder makes use of a very small percentage of the available resources for each device. It can be noted that the encoder uses 4 I/O ports, which are the clock, data input, data output and reset ports. The implemented encoder has an asynchronous reset, which is not mentioned in the encoder design section. The Cyclone V device uses 4 registers more than the other devices, which are used as routing optimisation registers.

Table 4.3.2: Encoder device resource usage

Device	Resource Type	Used	Total	%
Igloo2	Slice Flip Flops	15	12084	0.12
	4 input LUT's	12	12084	0.099
	Global	2	8	25
	I/O	4	231	1.73
Artix7	Slice Registers	15	126800	0.01
	Slice LUT's	9	63400	0.01
	Global (BUFG's)	1	32	3
	I/O	4	285	1
Cyclone V	Registers	19	37736	0.05
	LUT's	13	18860	0.07
	Global	1	16	6
	I/O	4	128	3

4.3.2 Timing analysis

The critical paths of the design are analysed to obtain the operating frequency of the encoder. The timing report obtains the pin-path-pin traversing delays and determines the worst case delays that are used to determine the highest operating frequency. The result of the timing analysis gives an indication of the possibility of implementing the design on the selected hardware (Perry, L.D, 2010).

The technology used in each of the test FPGA's results in a different maximum operating frequency for each device. However, these values are not for comparison as such a study would need a different kind of methodology to be put in place beyond the scope of this study. The post-layout/place-and-route timing analyses for each of

the devices give the minimum system clock period, which translates to the maximum frequencies recorded in Table 4.3.3.

Table 4.3.3: Encoder maximum system clock frequencies

Device	Parameter	Values	
Igloo2	System frequency (MHz)	505.051	
	Input data rate (Mbps)	101.01	
Artix7	System frequency (MHz)	576.037	
	Input data rate (Mbps)	115.207	
		Best case	**Worst case**
Cyclone V	System frequency (MHz)	841.75	487.57
	Input data rate (Mbps)	168.35	97.514

The maximum system clock frequency values for each device are obtained when a constraint of 50MHz is set on the clock. The timing analysis tools give results according to the devices' operating conditions. For the Cyclone V device, the best case scenario for the encoder timing is obtained from the fast 0ºC model and the worst case is obtained from the slow 85ºC device model. These models are generated as a result of the commercial device minimum and maximum operating temperatures. The time quest analyser used in Quartus II for the Cyclone V device gives the maximum frequencies for a device operating in the best and worst case conditions. The maximum frequencies given for the other devices are based on the devices operating under worst case conditions.

The maximum system clock frequency can be used to determine the input and output frequency. As stated during the encoder design, a complete encoder operation requires 5 system clock cycles to complete. The encoder input frequency can, therefore, be calculated as:

$$Input\ frequency = \frac{system\ frequency}{5} \qquad \textbf{4.3.1}$$

4.3.3 Power analysis

The design tools can be used to generate circuit representations of the encoder implementation on the FPGA device. These representations are used to estimate the power drawn by the designed circuit using activity values estimated by the power analysis tool. In order to obtain an accurate power estimation, a VCD (Value Change Dump) file describing the circuit activity in response to a particular stimulus has to be

generated during post-PAR simulation. The power analysis tools determine two types of power dissipation values, namely quiescent/static and dynamic power. These two values give the power drawn by the circuit when the device is powered up and idle, and the power dissipation due to circuit activity, respectively. The total estimated power required for the encoder is, therefore, the sum total of the dynamic and quiescent power.

A random data stream is generated using MATLAB and used as test data for the encoder/decoder system. In order to ensure decoding of the entire data stream, a sequence of 7 zeroes is appended at the end of the input data stream to flush out the contents of the encoder shift registers; thus, terminating the encoding process. The generated input is 3200 bits wide and, therefore, the resultant bit stream is 3207 bits wide due to the termination bits.

To determine the approximate encoder power requirement, VCD files are generated from simulations with operating frequencies between 25MHz and 300MHz. The switching activity observed here gives an indication of the power requirement for an encoder under typical operation. Figure 4.3.3 shows that the dynamic power usage increases linearly with operating frequency for each of the test FPGA's. The static power varies with the device junction temperature as illustrated in Figure 4.3.4. The total power required for the encoder operation is the sum total of dynamic power and static power. This is determined from the diagrams in Figure 4.3.3 and Figure 4.3.4.

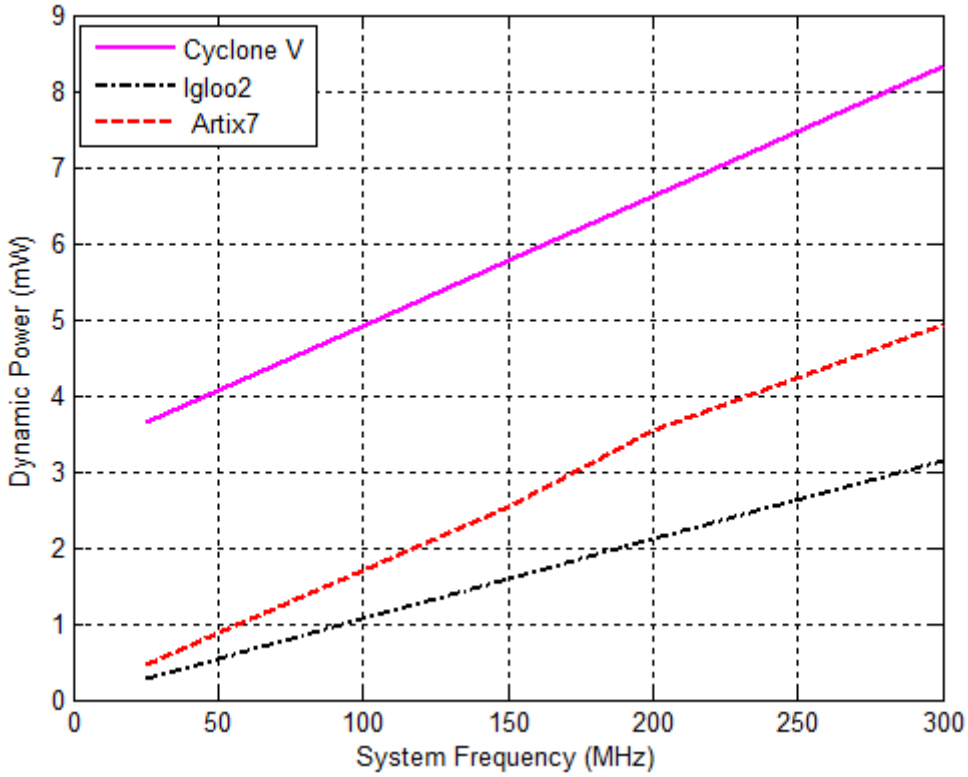


Figure 4.3.3: Encoder dynamic power requirement for system frequencies between 25MHz and 300MHz

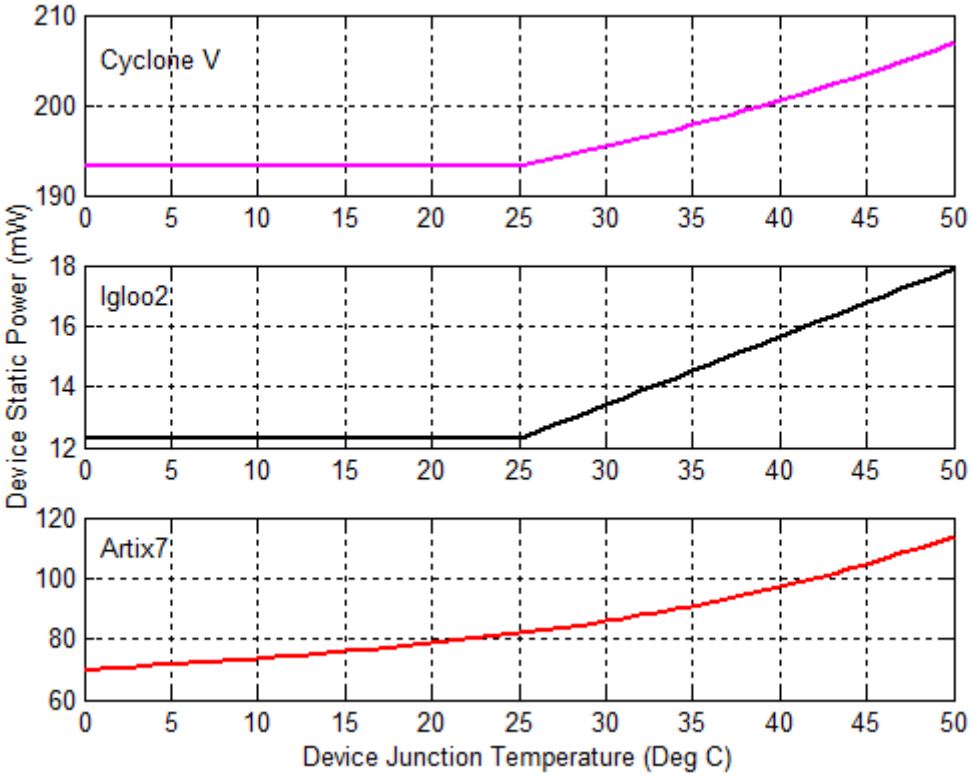


Figure 4.3.4: Encoder static power requirement versus junction temperature

4.4 Conclusion

The encoder requires the use of negligible resources as demonstrated by the analysis done in this chapter. The least implementation cost is observed for the Igloo2 device, which displays a low average static power requirements of 13 mW for typical operational conditions between 15ºC and 25ºC, and a dynamic power requirement of ~5mW between 25MHz and 300MHz. The limitations on the system performance will come not from the encoder, but from its counterpart, the *decoder*.

CHAPTER FIVE
VITERBI DECODER

5.1 Introduction

On the receiving side of an encoded communication link, a decoder is required to reverse the effects of the encoder so as to determine the original data. In this project, a hard decision trace-back Viterbi decoder is used to reverse the effects of the (7, ½) convolutional encoder used on the transmission side of the communication system. The decoder is designed using a hierarchical VHDL design methodology for implementation on an FPGA device.

5.1.1 Decoder outline

The description of a Viterbi decoder can be obtained from a trellis diagram, such as the one illustrated in Figure 3.4.3. The trellis diagram represents all the possible time-related state transitions of a specific convolutional code for a given time period. The branches along the trellis depict the encoded data at the particular time instance; therefore, a received encoded data stream follows a unique path along the trellis. This property is used in the description of the decoder to determine the original input.

In order to visualise the decoding process, an input data stream is encoded using the encoder in Figure 3.4.2, where the generator polynomials are $G1 = in \oplus s1 \oplus s2$ and $G2 = in \oplus s2$. The next 3 sections outline the step-by-step process implemented in the decoder to determine the input data stream from the received parity codeword.

5.1.1.1 Branch metrics and the trellis diagram

As stated earlier, the branches along the trellis give information on all the possible encoder outputs for all the possible encoder states in relation to the input and time, as shown in Figure 5.1.1.

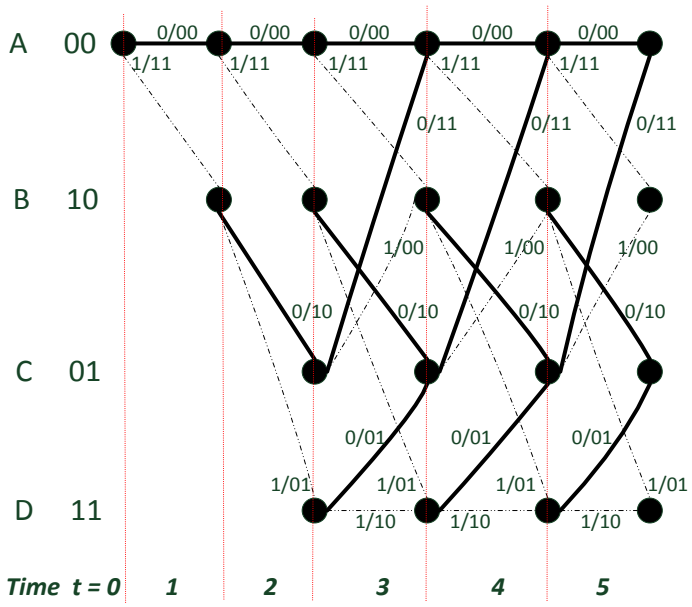


Figure 5.1.1: Trellis diagram for encoder with *K* = 3 and rate = ½

The first step in the decoding algorithm is the calculation of branch metrics, which can be defined as the Hamming distance between the branch-words received by the decoder and the expected/transmitted branch-words. The branch metrics can be calculated using the information available on the trellis, as illustrated in Figure 5.1.2. The code used for encoding is a ½-rate convolutional code. Consequently, there are four possible branch-word combinations **00**, **01**, **10** and **11** on the trellis diagram, which shall forthwith be referred to as *ideal* branch-words. The branch metrics corresponding to the ideal branch-words shall be labelled as **BM0**, **BM1**, **BM2** and **BM3**, respectively, as shown in Table 5.1.1.

Table 5.1.1: Branch metric reference table

Branch-word	BM0	BM1	BM2	BM3
00	0	1	1	2
01	1	0	2	1
10	1	2	0	1
11	2	1	1	0

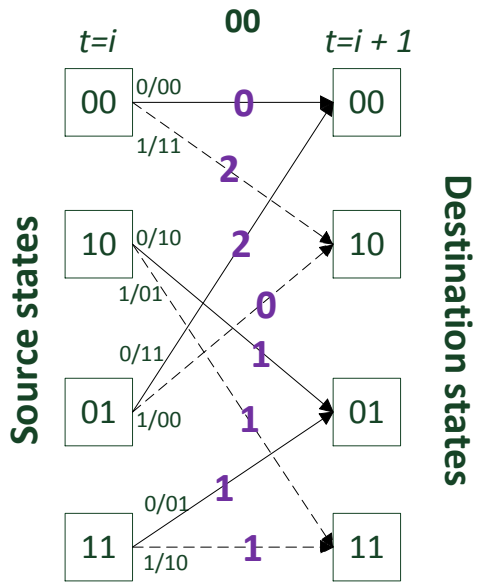


Figure 5.1.2: Example branch metrics for branch-word 00

Adapted from (MIT, 2010b)

5.1.1.2 Path metric accumulation and trellis butterflies

Stage 2 in the decoding process involves the calculation of the accumulative path metrics along the trellis. An encoded data stream follows a unique path along the trellis and, therefore, the path metrics are used to determine that path. The path metrics (PM) can be defined as the sum of the branch metrics along a particular path along the trellis, which accumulate with time. This value indicates the number of bit errors in the particular path obtained by comparing the received branch-words to the ideal branch-words from time t = 0 up to the current time (MIT, 2010b). Since the Viterbi decoder employs a maximum likelihood algorithm, the path along the trellis with the smallest path metric is the path with the greatest likelihood of representing the unique path resulting from the input data stream.

The ½-rate code has a symmetry property, whereby each source state has 2 possible destination states. These source/destination state pairs can be grouped to form trellis butterflies, such as the one shown in Figure 5.1.3. The trellis butterflies simplify the calculation of accumulative path metrics.

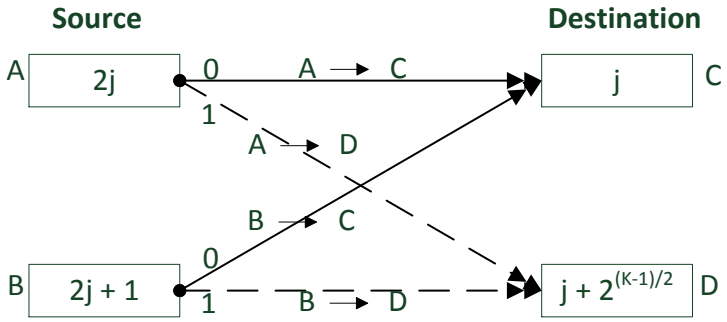


Figure 5.1.3: General ½-rate trellis butterfly

On the trellis butterfly, the paths to be updated during the add-compare-select (ACS) process are on the right-hand side of the diagram. Each butterfly is used to update 2 path metrics using equation 5.1.1, with reference to the labelling of the trellis butterfly in Figure 5.1.3:

$$\mathrm{PM(C)} = \min(\,\mathrm{BM(A \rightarrow C)} + \mathrm{PM(A)}, \mathrm{BM(B \rightarrow C)} + \mathrm{PM(B)}) \qquad \textbf{5.1.1}$$
$$PM(D) = \min(\,\mathrm{BM(A \rightarrow D)} + \mathrm{PM(A)}, \mathrm{BM(B \rightarrow D)} + \mathrm{PM(B)})$$

The ½-rate code with *K*=3 has 4 states and the trellis in Figure 5.1.1 can be translated into 2 butterflies, as illustrated in Figure 5.1.4. There are 2 source states for each destination state. To obtain the path with the highest likelihood representing the input data stream at that time instance, the source path with the smallest metric is selected as the winner path. It can be seen from Figure 5.1.4 that the ideal branch-words that result from a 0/1 input are the same for each source state in a butterfly. This property is used to determine the source input, which resulted in the transition from the particular source state to the winner path at a particular time instance.

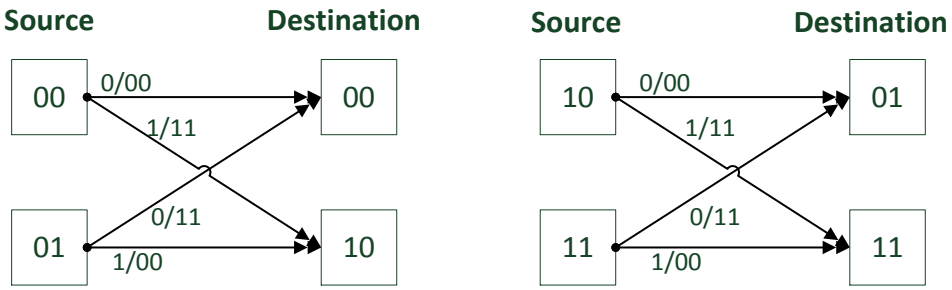


Figure 5.1.4: *K* = 3, *R* = ½ trellis butterflies

The accumulative path metrics for the example *K* = 3, *r* = ½ code can therefore be calculated using the formulae in equation 5.1.2:

$$\begin{aligned} \mathrm{PM0} &= \min(\mathrm{BM0} + \mathrm{PM0}, \mathrm{BM3} + \mathrm{PM2}) \\ \mathrm{PM1} &= \min(\mathrm{BM3} + \mathrm{PM0}, \mathrm{BM0} + \mathrm{PM2}) \\ \mathrm{PM2} &= \min(\mathrm{BM2} + \mathrm{PM1}, \mathrm{BM1} + \mathrm{PM3}) \\ \mathrm{PM3} &= \min(\mathrm{BM1} + \mathrm{PM1}, \mathrm{BM2} + \mathrm{PM3}) \end{aligned} \qquad \textbf{5.1.2}$$

It is important at this stage of the decoding process to take note of the survivor branches, which correspond to the minimum path metric at every time instance, as this information is used to determine the decoder output. These branches are referred to as the *decision* branches.

5.1.1.3 Decoder output

With the path metric information and the decision branches, the decoder has all the information required to complete the decoding process. After the decoder has received the complete parity codeword, the trellis is read in reverse, starting with the state at the end of the trellis with the smallest path metric to trace the path most likely to have been followed by the input during the encoding process. The most significant bit (MSB) of the state along the reverse trellis path is recorded as the output bit at that particular time instance.

Figure 5.1.5 shows the decoding process for a received codeword **11 10 00 10 11** encoded using the *K = 3, r = ½* encoder. The first figure shows the branch metric information for each branch-word of the input codeword and the accumulation of path metrics. The figure also highlights the survivor paths at each time instance. The accumulative path metrics are updated in the squares, which represent the nodes of the trellis. It is important to note that initially the path metric for state 00 is 0 and the rest of the states have a path metric of infinity due to the fact that the encoder states are initialised to zero at time t = 0. When the path metric for the two paths entering a node is identical, they are equally likely and, therefore, the survivor path is randomly selected (MIT, 2010b).

The second figure highlights the possible paths that may represent the input, eliminating the survivor branches from the previous diagram that do not contribute to a complete path traversing from the beginning of the trellis to the end. The final drawing highlights the unique path, which represents the input message, starting at the node with the lowest path metric. It can be noted that all the accumulative path

metrics along the path are 0, which occurs when the received codeword has no errors and correctly represents the encoded message (MIT, 2010b).

If the received codeword contains errors, the decoder selects the path with the lowest path metric, which represents the codeword that most closely represents the transmitted codeword, as illustrated in Figure 5.1.6. As a result of the encoder generator polynomials and the length of the input message (N), the encoder has a finite number of possible resultant codewords 2^N (MIT, 2010a). The codeword **01 10 11 10 11** used in the decoder illustration is not valid, because it cannot be obtained using any of the possible input combinations for a 5-bit message. There are two possible decoder outputs, as shown in the last 2 diagrams in Figure 5.1.6. The 2 paths have the same path metric, which makes them equally likely. The final output is, therefore, randomly selected.

Many Viterbi decoders have been developed over the years; however, out of necessity, the decoder described in this document has been developed from scratch. Many existing decoders are available as IP cores, but require paid licenses and have, therefore, not been considered for this project. The open source accessible Viterbi decoder solutions imposed a great learning curve for understanding the VHDL description as the free available code was not working correctly and no support was made available, despite our many requests. The decision was, therefore, made to develop a new VHDL architecture and description from scratch.

It must be noted that the Viterbi decoder developed is unique to the encoder with the generator polynomials described in this document. However, a different code of the same rate and constraint with different generator polynomials can be decoded using the same decoder by changing the branch metric values, which are linked to each trellis butterfly.

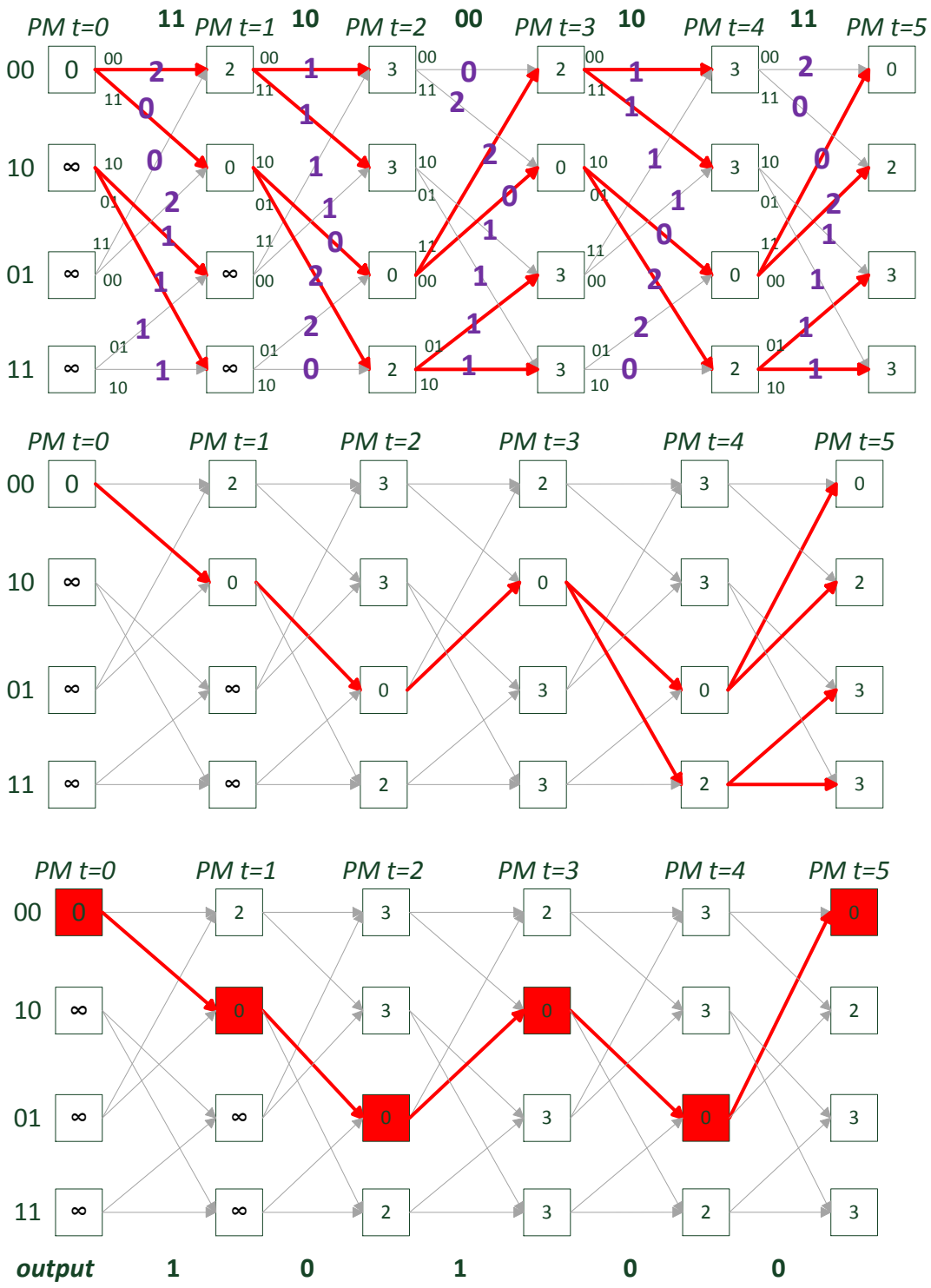


Figure 5.1.5: Viterbi decoder decoding the valid codeword 11 10 00 10 11 producing the transmitted message 10100

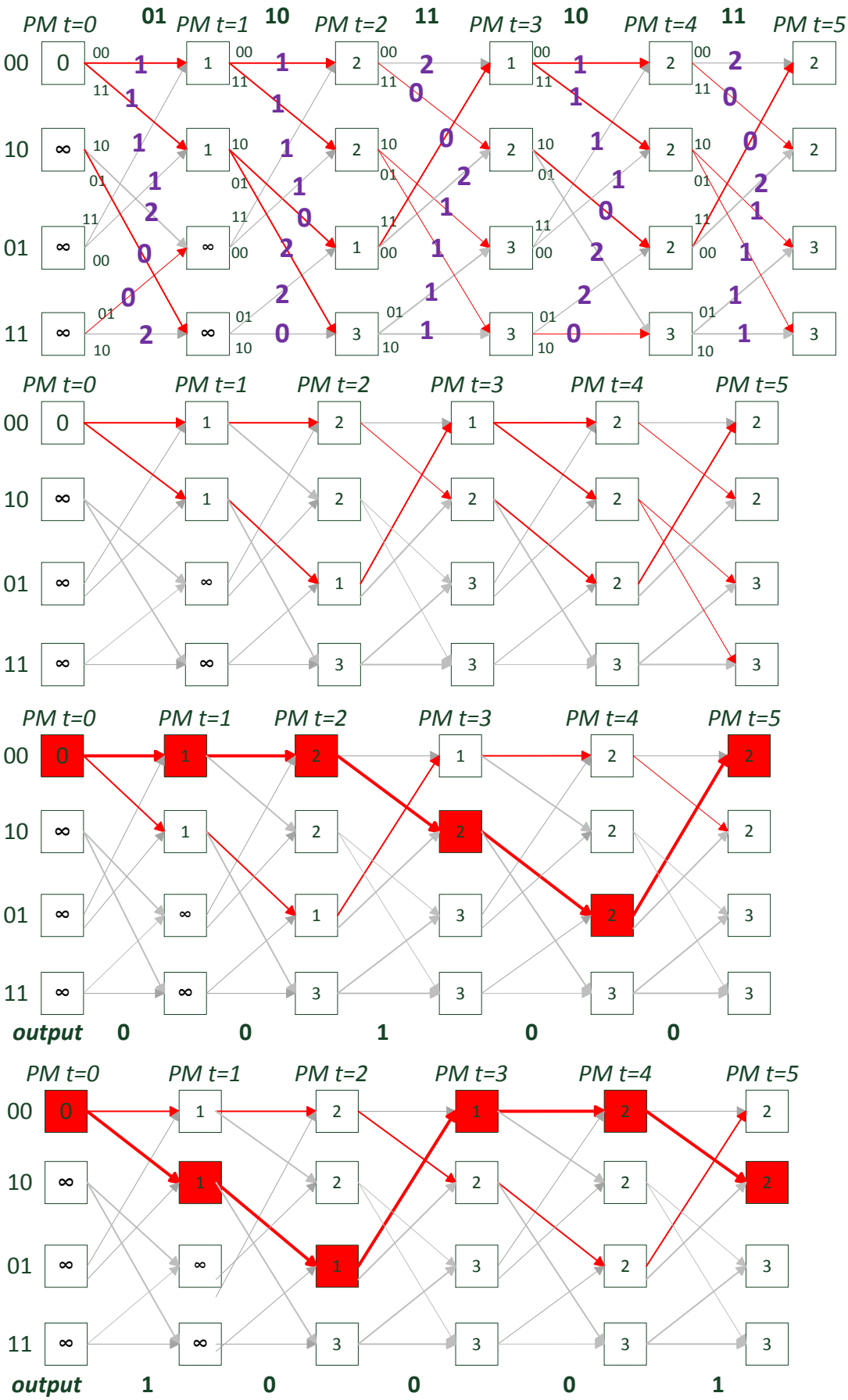


Figure 5.1.6: Viterbi decoder decoding the invalid codeword 01 10 11 10 11

5.1.2 Algorithm

The process of Viterbi decoding can now be described using an algorithm with the following steps, which practical implementation in VHDL is explained in detail throughout this chapter:

1. Initialisation: Set the trellis to the zero state.
2. Branch Metric calculation (BM): Calculate the branch metric for each branch in the trellis for a particular time instance. The branch metric is obtained by calculating the distance between the received branch-word and all possible branch-words. The number of different bits between compared branch-words is used as the branch metric value.
3. Add – Compare – Select (ACS): Calculate and update the path metric values for each path entering a node in the trellis and discard loser paths. The loser paths along the trellis at a given time instance are defined as the paths with the largest path metrics.
    a. Add obtained branch metrics to the relevant path metrics;
    b. Compare metrics of paths entering a node;
    c. Select a survivor path, which is given as the path with the smallest metric; and
    d. In case of a tie (equal metric), randomly select survivor path.
4. Determine output (TB):
    a. Use survivor paths from ACS to select the ultimate survivor path; and
    b. Derive the output from the ultimate survivor path.

5.2 Decoder Design

The decoder is made up of three major blocks, namely, the branch metric unit (BMU), the add-compare-select unit (ACSU) and the trace-back unit (TBU) as shown in Figure 5.2.1. For implementation purposes and verification simplicity, the blocks are treated as separate entities until their independent functionality is achieved. They are then combined to form one decoder entity with multiple components.

Figure 5.2.1: Viterbi decoder general block diagram

Each of the major components has a contributing role in the identification and correction of errors through the sharing of information among blocks. As a result, all the relevant blocks must communicate with each other, or use additional blocks to facilitate the flow of signals among the major blocks, as illustrated in Figure 5.2.2.

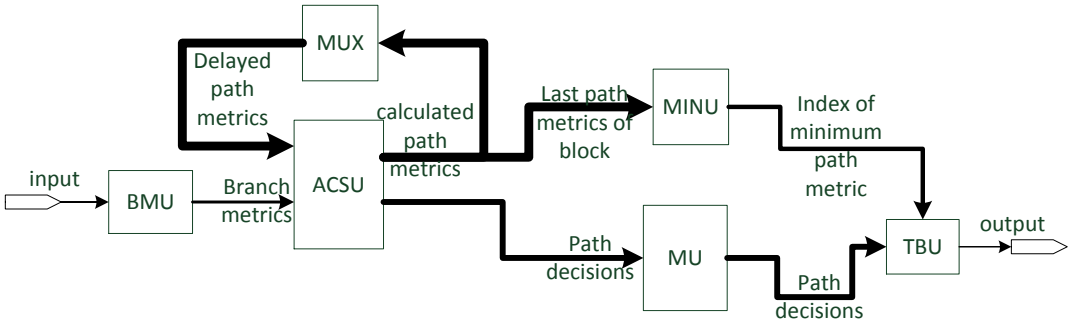


Figure 5.2.2: Major decoder blocks with additional synchronisation blocks

The direction of flow of information among the decoder blocks is as shown by the arrows in Figure 5.2.2. Each block needs to be activated when the correct input is available from the preceding block. To achieve the desired signal flow, a control unit block is introduced to activate the relevant blocks as desired; thus, synchronising the calculations, I/O control and overall decoder operation as illustrated in Figure 5.2.3. Each block represents a set of related processes, which contribute to the overall decoding process as described in Table 5.2.1. The description of the blocks is then used to define the VHDL behavioural architecture for the FPGA implementation. The decoder is implemented as a completely synchronous architecture; therefore, the clock is used in all the blocks in conjunction with the control signals.

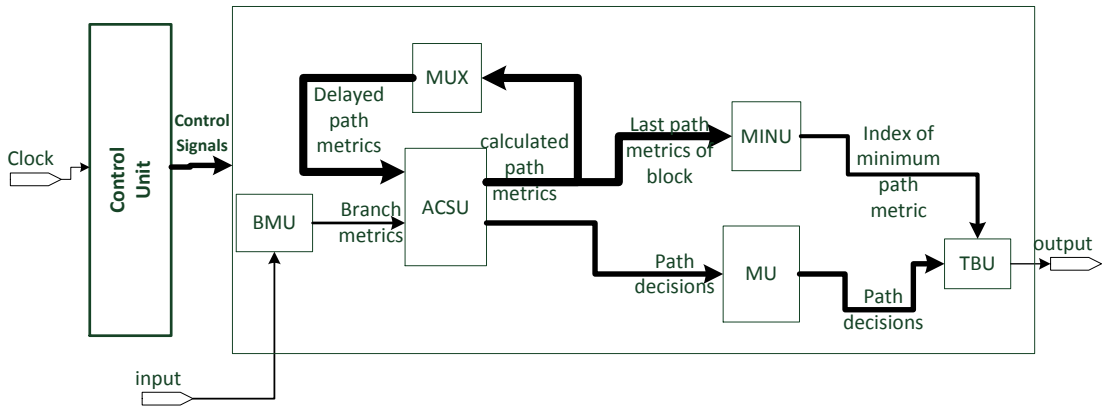


Figure 5.2.3: Interaction of decoder blocks with control unit included

Table 5.2.1: Viterbi decoder block description breakdown

Stage	Name	Description
1	Control	This block is responsible for creating control signals using the input clock signal to synchronise the input selection, calculations, and output selection processes.
2	BMU	Responsible for branch metric calculations required to determine the decoder output.
3	ACS	This block is used for calculating and updating the path metric values for each path entering a node in the trellis. This block is also responsible for making decisions determining the surviving paths along the trellis.
4	MUX	This block represents the block that selects the correct path metric values to be used in ACS unit calculations. It is an additional block used for signal synchronisation.
5	MINU	This block calculates the minimum path metric required for determining the state where the trace-back path begins.
6	MU	Required for the storage of the decisions made by the ACS unit.
7	TBU	Responsible for following the ultimate survivor path and determining the decoder output.

5.2.1 Control unit

The control unit generates the control signals, which synchronise the selection of block inputs and outputs as well as control the activation of the processes within these blocks. Multiple control signals are created using the system clock to control the propagation of information through the different blocks. The signals produced by the control unit are synchronised according to the timing requirements of the rest of the decoder system.

Figure 5.2.4 shows the relationship between the system clock and the control signals, which indicates the parallel and sequential nature of the decoder system. The control signals are used to activate/enable one or more blocks in the system.

Table 5.2.2 lists the control unit signals and the blocks for which they are responsible.

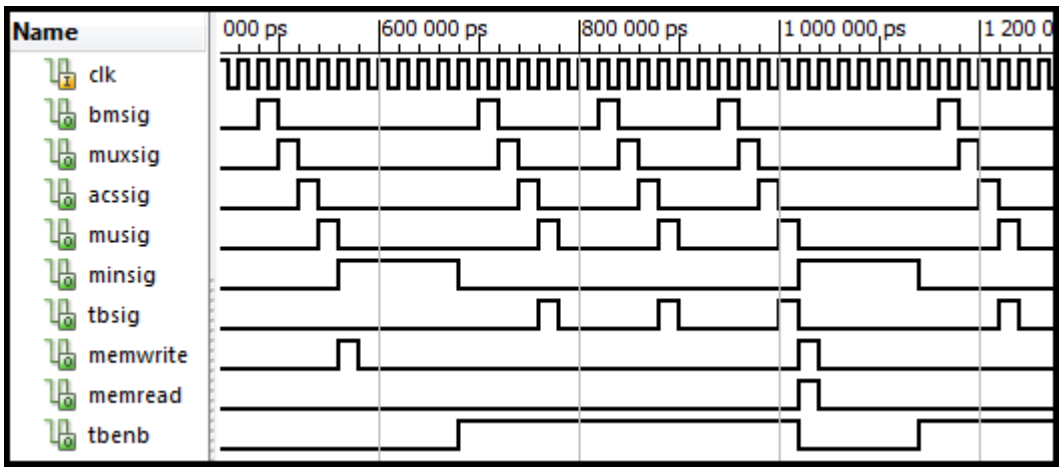


Figure 5.2.4: Control unit signal synchronisation

Table 5.2.2: Decoder control signals

Signal	System
bmsig	Enables calculation of branch metric.
muxsig	Enables MUX unit for ACS path metric input selection.
acssig	Controls the ACS unit to update the path metric value used for computations.
musig	Enables the writing of decision data from the ACS to memory.
minsig	Controls the calculation of the minimum path metric, which gives the trace-back starting point.
tbsig	Controls the saving of the valid decoder outputs.
memwrite	Controls the selection of the RAM block to be used for writing the ACS decision vectors.
memread	Controls the selection of the RAM block to be read from during trace-back.
tbenb	Activates and deactivates the entire trace-back unit.

5.2.2 Branch metric unit (BMU)

A metric defines the distance between elements and gives an indication of how closely placed the elements are in a particular space. The BMU is, therefore, responsible for calculating the distance metrics of the received branch-words and the ideal branch-words represented by the branches in a trellis diagram.

As mentioned in earlier sections, the BMU is responsible for calculating the difference between the received branch-word and the possible codewords for each of the trellis branches. The resultant value from the calculations represents the number of errors in the received branch-words. There are 2^n BM calculations for a code with rate *K*/n;

therefore, for a ½-rate decoder, the BMU carries out 4 calculations for each trellis branch.

The BMU accepts an input that is 2 bits wide and produces 4 outputs, namely, BM0, BM1, BM2 and BM3, which represent the Hamming distance between the input branch-word and the ideal branch-words. The operation of the BMU is controlled by the global clock in conjunction with the BMU control signal as shown in Figure 5.2.5.

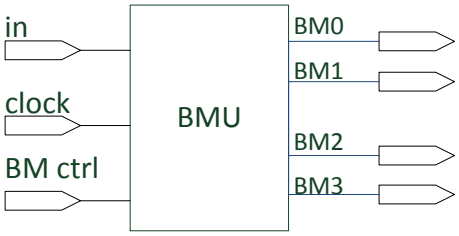


Figure 5.2.5: Branch metric unit

The arithmetic and logic operations implemented to obtain the branch metric results are as illustrated in Figure 5.2.6.

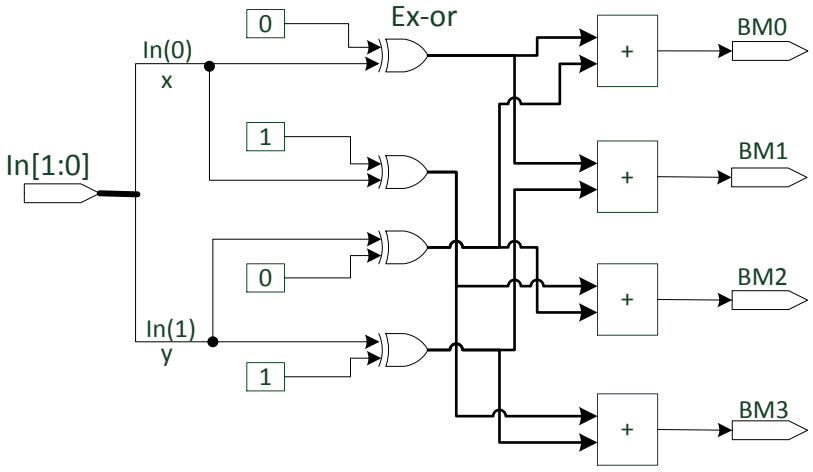


Figure 5.2.6: BMU hard decision calculations

There are, therefore, 3 possible branch metric values that can be obtained from the BMU for each trellis branch as displayed in Table 5.2.3.

Table 5.2.3: Hard decision BMU expected outputs

In[1:0]	x	x⊻0	x⊻1	y	y⊻0	y⊻1	BM0	BM1	BM2	BM3
00	0	0	1	0	0	1	0	1	1	2
01	0	0	1	1	1	0	1	0	2	1
10	1	1	0	0	0	1	1	2	0	1
11	1	1	0	1	1	0	2	1	1	0

5.2.2.1 Testing

With reference to Table 5.2.3, the expected BMU outputs are known; hence, to verify the BMU operation, a set of 2-bit signals is generated and used as inputs and the result recorded as in Figure 5.2.7. The results show that a new branch metric is only calculated on the rising edge of the global clock signal when the branch metric control signal (bmsig) is active, verifying the operation of the BMU.

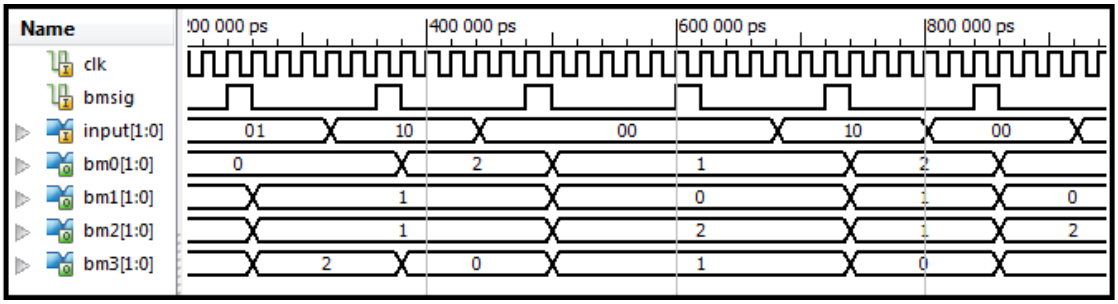


Figure 5.2.7: BMU test results

5.2.3 Add compare select unit (ACSU)

The ACSU is responsible for calculating the possible trellis survivor paths during each decoder time instance. The ACSU operates, assuming a trellis in a steady-state where more than one path enter a node at any given time. The VHDL behavioural description for the ACSU is a hierarchical model that is made up of multiple parallel ACS units.

5.2.3.1 ACS units

There are N/2 butterflies for an N state trellis; therefore, the trellis with 64 states can be broken down into 32 butterflies, such as the one illustrated in Figure 5.1.3. The trellis butterflies for the implemented decoder are illustrated in APPENDIX B: TRELLIS BUTTERFLIES. Each butterfly is represented by an ACS unit for the path metric update computations. The length of the input sequence used for the likelihood estimations determines the amount of storage required for the path metric information. As mentioned in Section 3, the reasonable truncation length for the encoded sequence is 5 times the constraint length of the encoder. In this case, the recommended truncation length is 35, which implies that updated path metrics for 35-time instances need to be saved and used to determine the decoder output. This places a large storage requirement on the decoder, since the path metric results are multiple bit values. To attempt a reduction in the decoder memory required, the

truncation length is reduced and the decoder performance at the reduced truncation length is noted. The results showing the impact of the reduced truncation length on the decoder performance are given in Section 5.3.

The trellis butterfly has a property that allows the use of single bits, referred to as decision bits, to represent the survivor path metrics to be saved. From Figure 5.2.8a and Figure 5.2.8b it can be seen that the LSB for source state A is 0 and the LSB for source state B is 1; a property that is true for all the trellis butterflies. The trellis butterfly states can, therefore, be labelled as in Figure 5.2.8c. If the surviving path emanates from state A, the decision bit is set to '0' and if it is from state B a '1' is saved. The same branch metric values are used for the state transitions from both source states, as seen in Figure 5.2.8a and Figure 5.2.8b, resulting in the new branch metric labels, as seen in Figure 5.2.8c. The decision bits can, therefore, be calculated using the formula in equation 5.2.1:

$$\begin{aligned} &\text{dec0} = 1 \text{ if } \text{PMA} + \text{BMtop} > \text{PMB} + \text{BMbottom} \text{ else } 0 \\ &\text{dec1} = 1 \text{ if } \text{PMA} + \text{BMbottom} > \text{PMB} + \text{BMtop} \text{ else } 0 \end{aligned} \qquad \textbf{5.2.1}$$

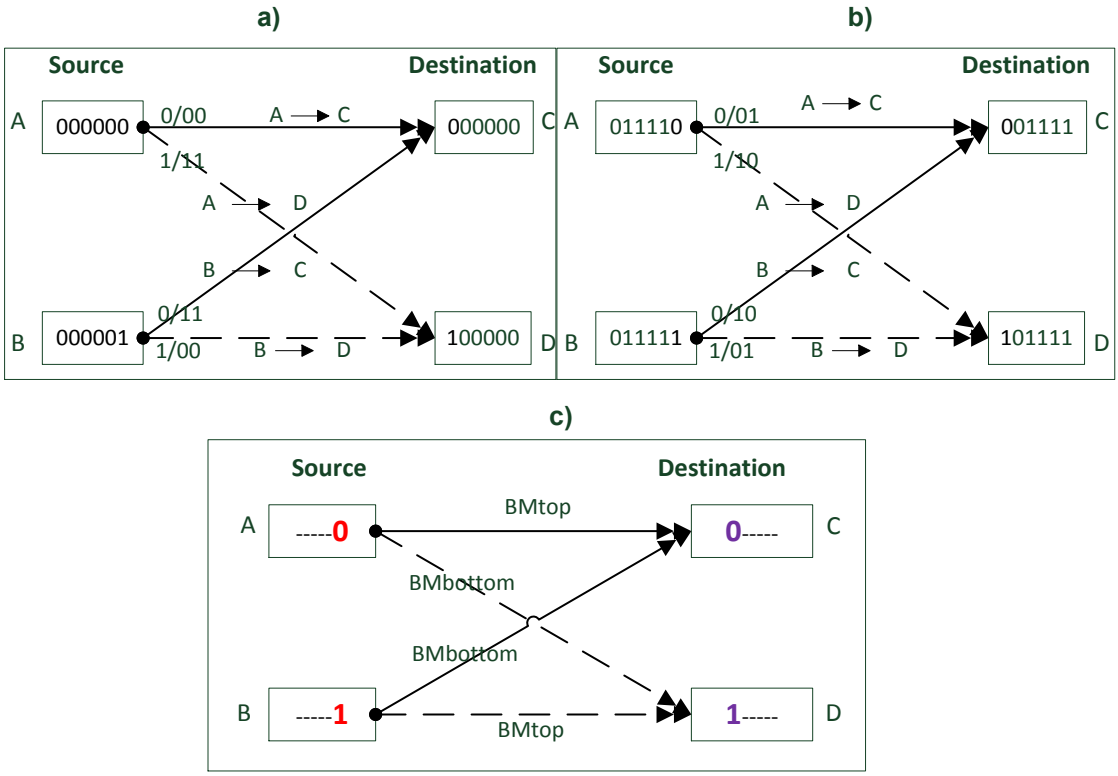


Figure 5.2.8: Trellis diagram for generating decision bits a) trellis butterfly for updating state 0 and 1, b) trellis butterfly for updating state 15 and 47, c) generic trellis butterfly diagram for all states showing the LSB and MSB pattern for source and destination states

The updated path metric is still required for use in the next time instance and is, therefore, temporarily stored and replaced when the next PM calculation is executed. As a result, each ACS unit is designed to give four outputs; 2 decision bits to be used by the decoder trace-back component and 2 path metric values for updating the path metric for the next time instance.

32 ACS units with a structure as shown in Figure 5.2.9 are, thus, implemented in parallel to give the desired path selection and decision bit results for each trellis butterfly. Each unit is responsible for updating the path metric for two states according to the trellis butterfly relationships and determining the corresponding decision bits denoting survivor paths.

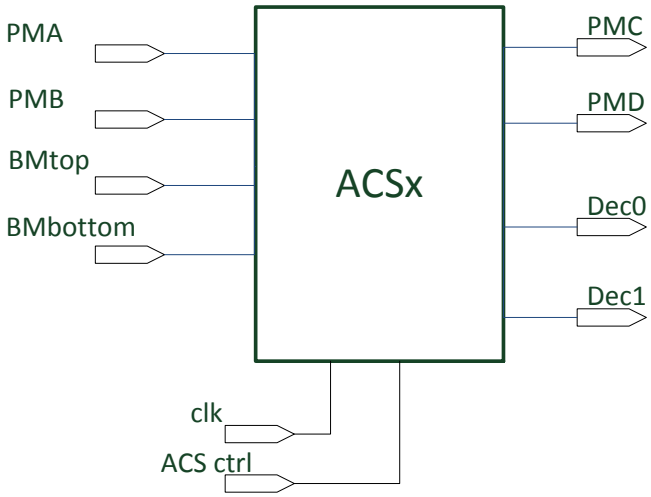


Figure 5.2.9: Individual ACS unit structure

To implement the ACS units, the arithmetic and logic operations as illustrated in Figure 5.2.10 are used. Each ACS unit requires 4 adders and 2 comparators.

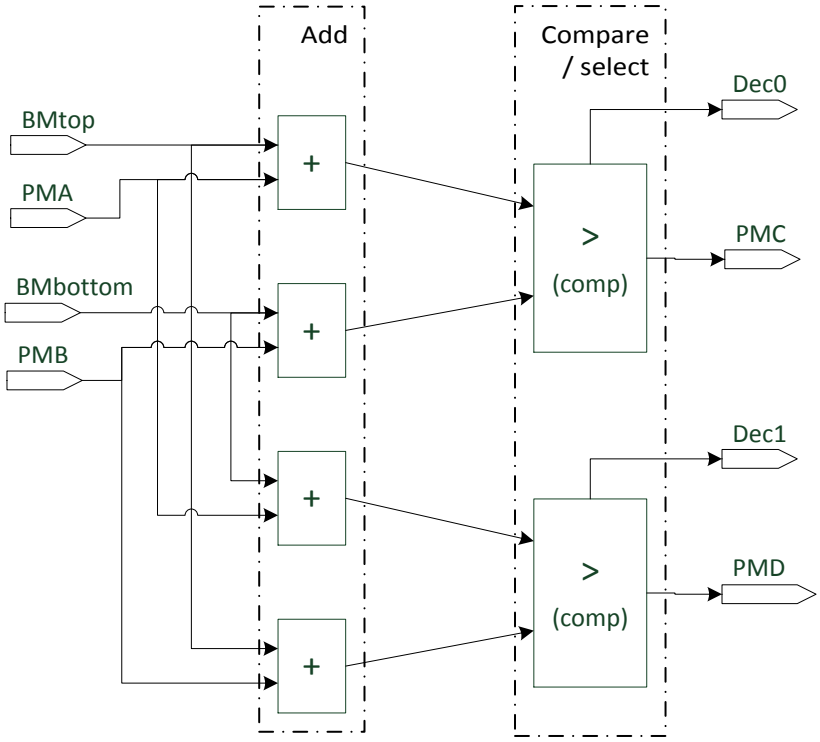


Figure 5.2.10: ACS unit arithmetic and logic operations

5.2.3.2 ACS top

The ACS module is a hierarchical model with multiple interconnected components; the top entity (APPENDIX C: ADD COMPARE SELECT (ACS) MODULE INTERCONNECTION) is responsible for defining the connections among the different components as determined by the trellis butterfly relationships. All the individual decision bits obtained from each of the parallel units are merged into a data bus containing 64 decision bits, which are saved for future use. This implies that a memory unit is required to store the decision bits.

5.2.3.3 Temporary path metric storage unit (Mux)

A temporary path metric storage unit (MUX) is used to save the survivor path metric values calculated by the ACS units, which are to be used at the next time instance as the ACSU inputs. The interaction between the MUX unit and the ACSU is as illustrated in Figure 5.2.11.

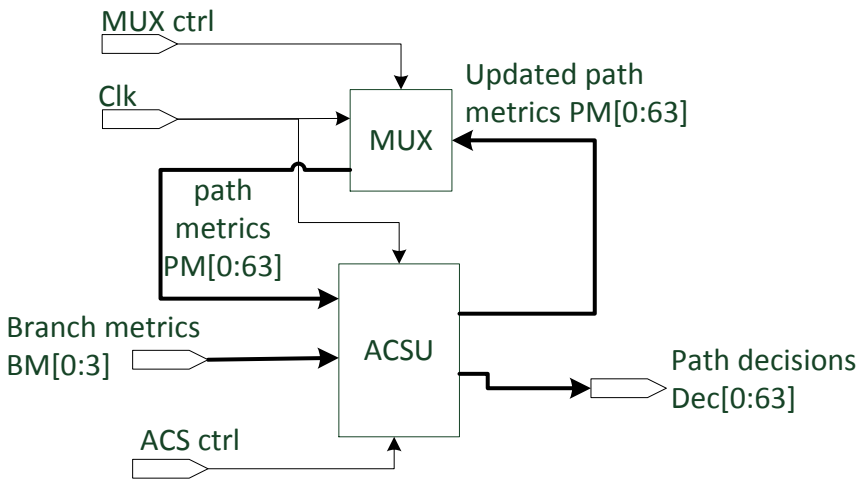


Figure 5.2.11: ACSU and MUX interaction

At time t = 0, only one state exists on the trellis as illustrated in Figure 3.4.3. The only existent state is the zero state, which has a path metric of zero, as it has no path feeding into it. The rest of the states are non-existent and, hence, their path metrics do not exist. For ease of calculation, the path metrics for the states that are not present at t=0 are set to a high value, which represents a path metric of infinity. In this implementation, the number 200 is used to represent a path metric of infinity. The path metric values defined here are used as the initial inputs to the ACS units to be used for the first accumulative path metric calculation at time t = 1.

The MUX unit is responsible for selecting the input to the ACSU as required at a particular time instance, as shown in Figure 5.2.12. At the beginning of the decoding process, the initial path metric as described above is used as the ACSU input.

Figure 5.2.12 illustrates the hardware description of the MUX unit, which is made up of a memory register and a 2-to-1 multiplexer, and represented as a selector switch controlled by an external control input.

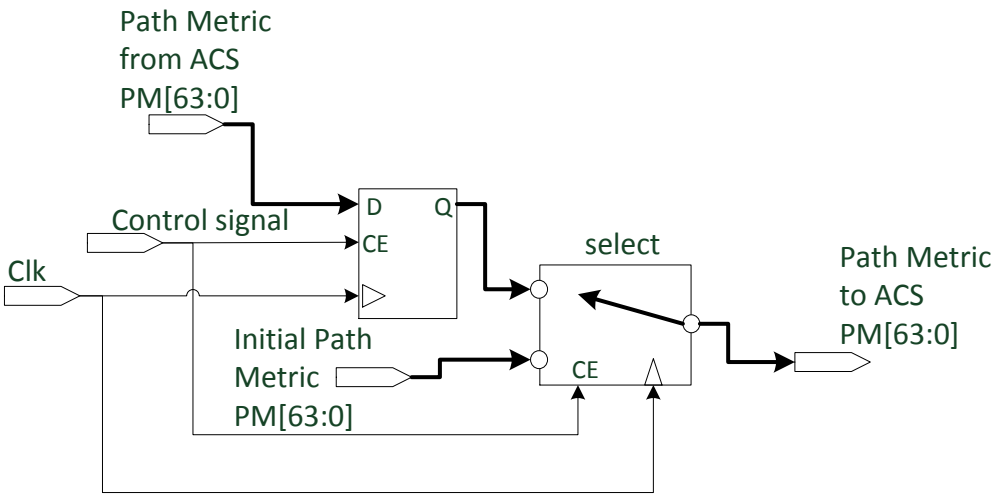


Figure 5.2.12: MUX unit path metric selection functional block diagram

5.2.3.4 Testing

To verify the successful implementation of the ACS unit, a spreadsheet reference model is used to give the expected decision bit output values after each stage. The developed ACS module is tested using the branch metric results obtained from the BMU functionality test. To determine the success of the implementation, the expected results from the spreadsheet are compared to the simulation results.

5.2.4 Decoder output selection unit

The decoder output selection unit is made up of the trace-back (TBU), memory (MU) and minimum path metric (MINU) units as shown in Figure 5.2.13. Each of these units contributes to the decoder output calculation as described in the following sub-sections.

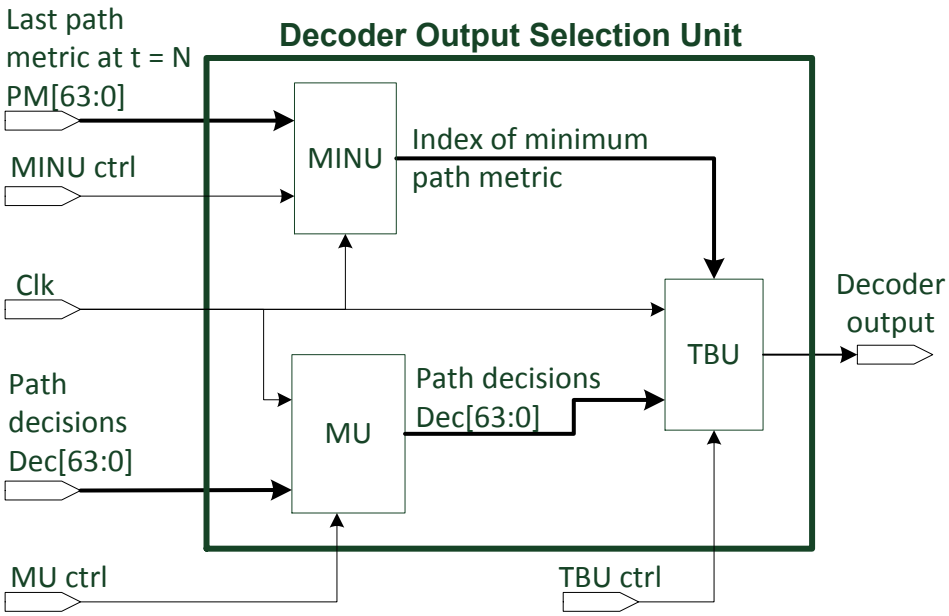


Figure 5.2.13: Decoder output unit

5.2.4.1 Memory unit (MU)

The output of the ACSU for long-term storage has been established to be a 63-bit wide signal containing survivor path metric decision bits. At each time instance, decisions are made using the path metrics from the previous time and the branch metrics corresponding to the given input. The obtained decisions are stored to be used later in determining the decoder output sequence. A last-in-first-out (LIFO) random access memory (RAM) is created for storing the required decision vectors. Three blocks of RAM, which size is determined by the trace-back depth (truncation length) of the decoder, are used for the storage. The stored decision bits are used in the trace-back block where the decoder output is generated.

As shown in Figure 5.2.15, the MU reading and writing sequences are controlled by control signals from the decoder control unit. The selection of the RAM block to be used in write mode at a particular time instance is done using the finite state machine (FSM) illustrated in Figure 5.2.14a. The FSM state transitions are controlled by the memory unit write/read select control signals (memwrite/memread ctrl) obtained from the decoder control unit. The write select FSM gives a 3-bit output and each RAM block uses one bit as an enable signal as, shown in Figure 5.2.15.

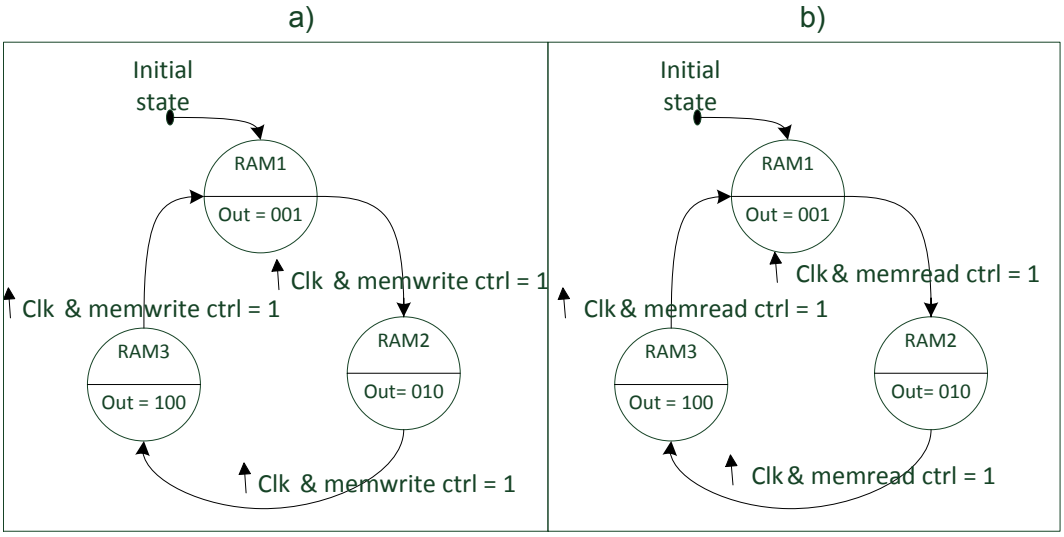


Figure 5.2.14: RAM selection FSM state diagrams a) Write RAM select b) Read RAM select

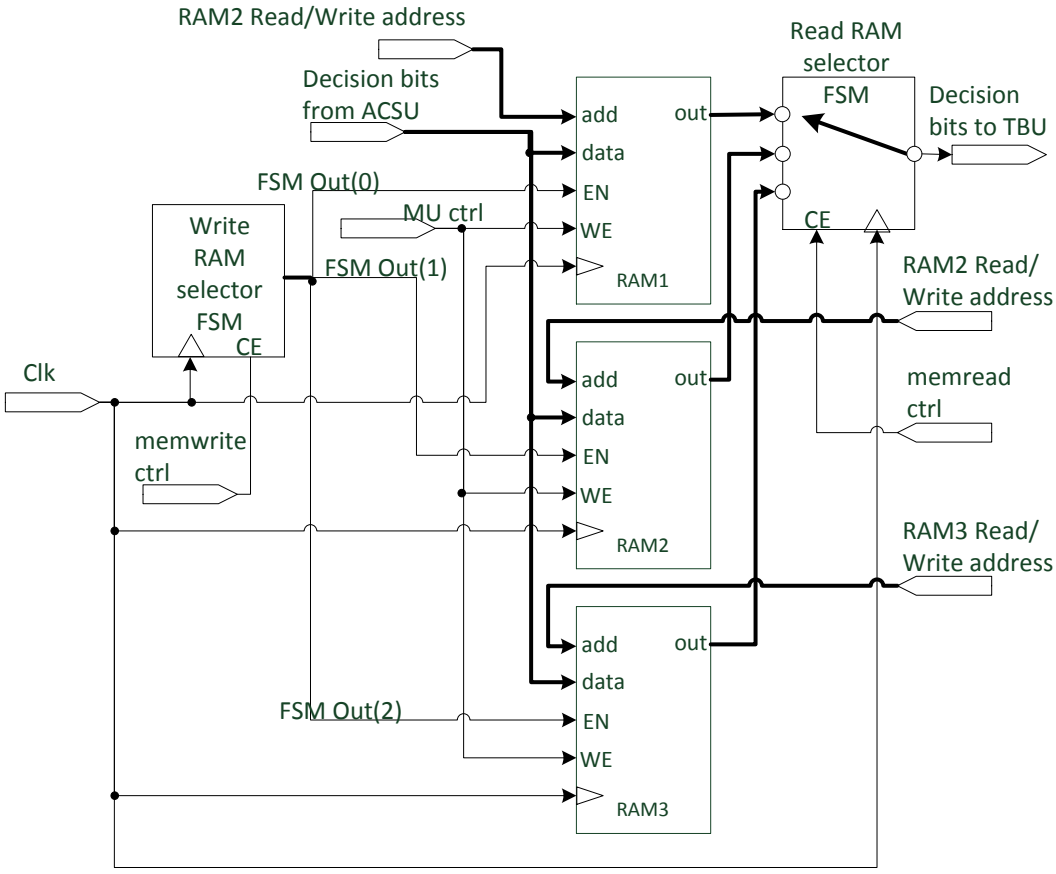


Figure 5.2.15: Memory unit block configuration

5.2.4.2 Minimum path metric calculation (MINU)

At the end of each trellis, the minimum path metric must be calculated to determine the starting point for the decoder trace-back along the trellis to select the path that most likely represents the correct output. The minimum path metric value is

calculated and the corresponding index (between 0 and 63) of that value is used to determine the starting state and the decision bit for that state. All the accumulated metrics for each of the 64 paths are compared to determine the one with the least numerical value. To reduce the time consumed by the minimum path metric calculation, the pipeline method involving both parallel and sequential instructions is employed as illustrated in Figure 5.2.16. This unit introduces a delay of 5 system clock cycles, which are required to complete the minimum path metric calculation.

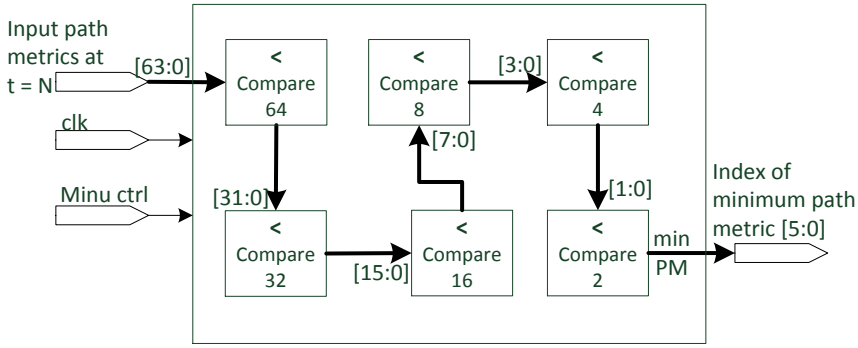


Figure 5.2.16: Minimum path metric calculation block diagram

5.2.4.3 Trace-back unit

The TB unit is the decoder stage where the decoder output is generated using the decision bits saved in the RAM. These bits are accessed and used to infer the information sequence with the highest likelihood of having originated from the data source.

As established earlier, the trellis butterflies help with trellis navigation and determining the most probable path followed with respect to a particular input stream. As such, the trellis butterflies in APPENDIX B: TRELLIS BUTTERFLIES, if read from right to left, with help from the decision bit information, can be used to determine the decoder output. Figure 5.2.8 shows the generic pattern followed by the LSB and MSB of states in relation to the butterfly source and destination states. Using this pattern, the trellis butterfly can be broken down to show only one destination state with its source states as shown in Figure 5.2.17. The forward directional butterfly shows that the MSB of the destination state is **0**, given that the input was a **0**, and **1**, given that the input was a **1**. It can also be seen that, if the destination state is reached from a state of which the LSB is **1,** the decision bit is set to **1.** The same logic applies if the destination state is reached from a state of which the LSB is **0**; the decision bit is set to **0**. As such, when the butterflies are read backwards, the preceding state can be deduced using the decision bit and the decoder output obtained from the state MSB.

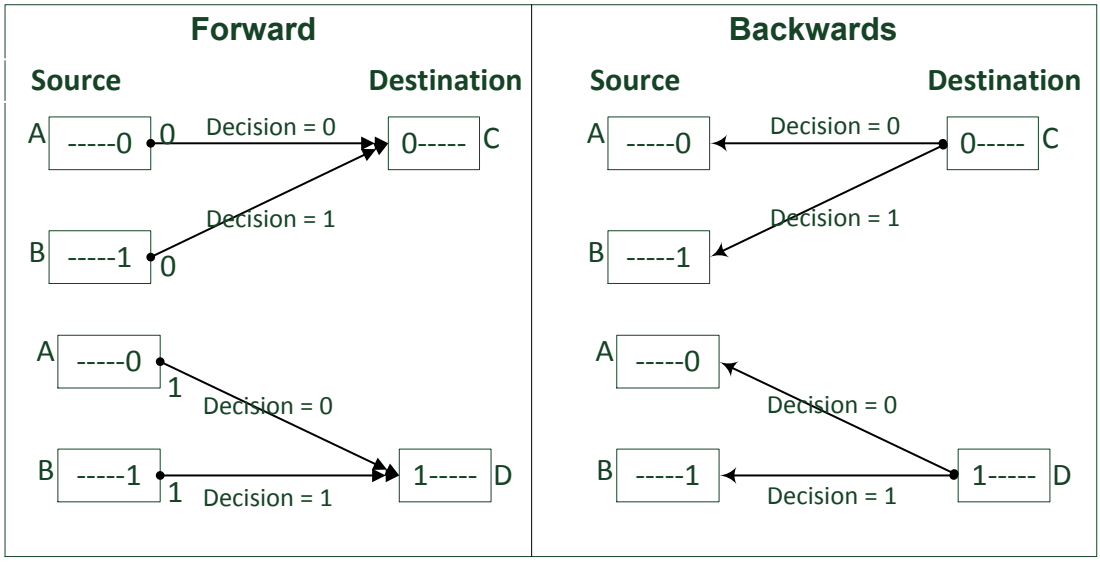


Figure 5.2.17: Forward and backward trellis butterfly relationships

The backward trellis butterfly relationships as shown in Figure 5.2.17 are used during this stage of the decoder to determine the trace-back path along the trellis.

Figure 5.2.18 shows the configuration of the TB unit and the interconnectivity of the TB sub-blocks. The implemented decoder is completely synchronous, and all the sub-blocks of the TB unit are controlled using the global clock in conjunction with a control signal.

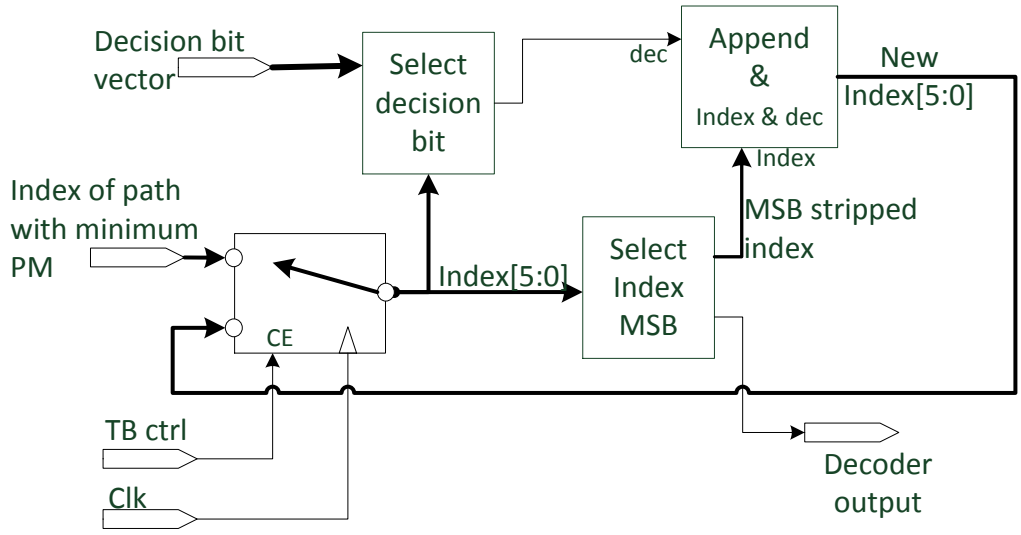


Figure 5.2.18: Trace-back unit configuration block diagram

The trace-back algorithm can be described as follows:

Step 1. At the beginning of the trace-back process (path end) select the index of the path with the minimum path metric as input index; otherwise, use the index as generated in the previous TB unit calculation (*New Index*).

Step 2. Use the index from step 1 to select the decision bit corresponding to that index.

Step 3. Remove the index MSB of the index from step 1; **NB**: this is the decoder output at that time instance.

Step 4. To get the preceding node in the path, append the decision bit from step 2 to the stripped down index from step 3. The decision bit is added as the LSB forming the new index.

Step 5. Go back to step 1

To further illustrate the steps of the TB algorithm, if the path index in step 1 is 15 (**001111**), then the decoder output in step 3 would be **0**, and the stripped down index would be **01111**. If the decision bit corresponding to the index is **0**, the result from step 4 would be **011110**. This state transition relationship can be confirmed using the trellis butterfly in Figure 5.2.8b; thus, verifying the accuracy of the TB algorithm.

The trace-back unit moves through the trellis in reverse order; hence, the output is in reverse order, which means that the output has to be reordered before use. The process of reordering is omitted in the VHDL description of the decoder and implemented externally as part of the overall system test.

To test the functionality of the output unit, the expected values are obtained from the spreadsheet model and compared to the results obtained from the VHDL simulation. The output of the ACSU is used as the input to this block during the testing phase.

5.2.5 Truncation length

As mentioned earlier in the document, the trace-back (TB)/truncation length has an effect on the overall decoder performance. It is, therefore, important to note the effects of changing the trace-back length on the overall decoder architecture. The TB length only has an effect on the RAM memory requirement and has no other effect on the rest of the decoder architecture. The TB length also affects the decoder timing requirements. This effect is explained in the decoder analysis Section 5.3. The main reason the TB length affects the decoder performance is that a longer TB provides

more historical path metric data and, therefore, increases the accuracy of the likelihood estimations.

5.3 Analysis

The decoder analysis and testing involve 3 stages, which are responsible for

- verifying the decoder's ability to replicate a given input bit stream after it is subjected to convolutional encoding;
- determining the error correction capability of the decoder; and
- quantifying the coding gain estimations attainable using the implemented encoder/decoder pair.

A test system as illustrated in Figure 5.3.1 is used to execute the required analysis. The diagram shows the subsystems required for each analysis stage, as well as the platform used to perform the operations. The testing algorithm involves VHDL and MATLAB simulations as labelled in Figure 5.3.1.

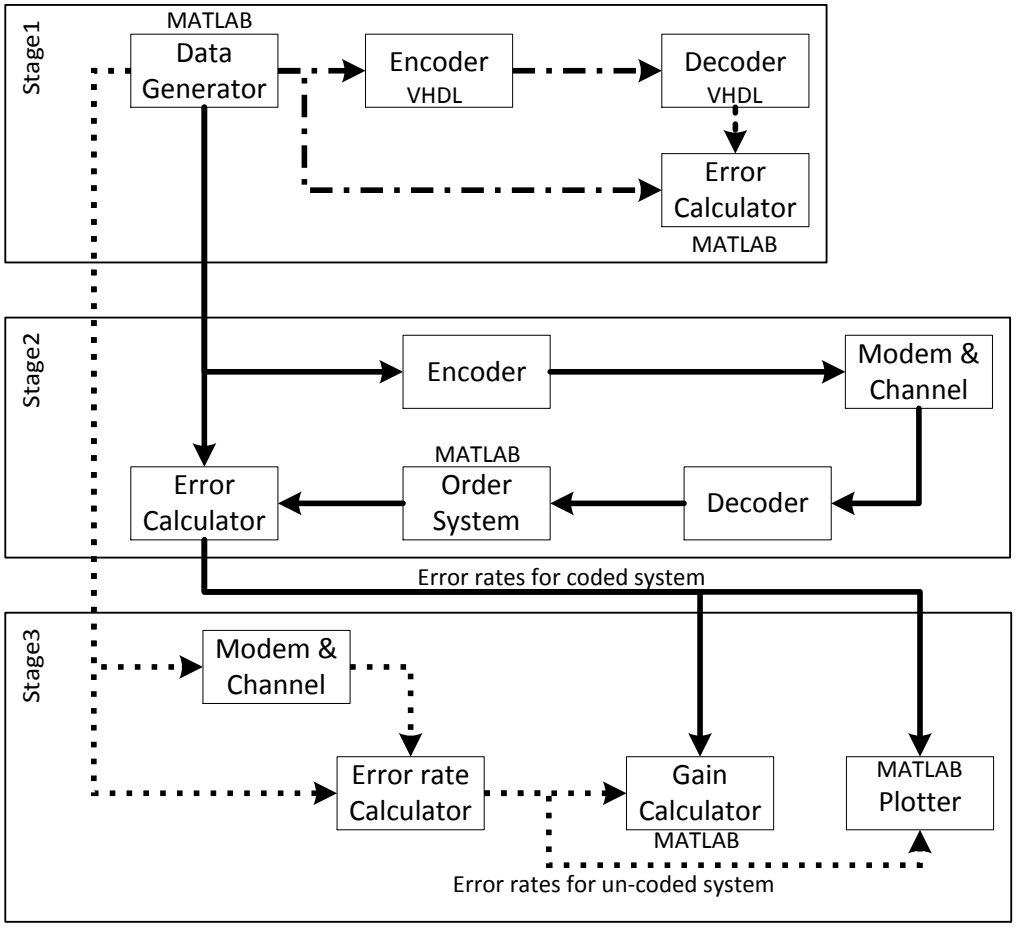


Figure 5.3.1: Decoder test and analysis system diagram

The first step in the analysis is to verify that the decoder can replicate the initial input sequence as can be seen in Figure 5.3.1. The MATLAB data generator creates a text file, which is used in the VHDL test bench. The error calculator stage 1 of the decoder test should yield a result of zero as there should be no errors during this testing phase.

After successful verification of the decoder's basic operation, the error correction capability is tested. Estimations of the achievable coding gain, as a result of the encoder and decoder pair, are obtained using the sequence labelled stage 3 in the test system diagram. The estimates are obtained by comparing the error values from the decoder error correction capability test and the error values from the un-coded information.

5.3.1 Error correction

Testing the error correction capability requires the introduction of errors, which typically occur during space communications. For the introduction of these errors, a MATLAB AWGN channel model is used. In order to mimic a 'real-life' communication channel, a MATLAB BPSK modem is also implemented in the test system. The communication scenario is simulated for an AWGN channel with an E_b/N_o ranging from -1 dB to 12 dB, and the error rates for the decoder observed.

The truncation length/trace-back depth of the decoder affects the error correction capability of the Viterbi decoders as seen in Figure 3.4.5. The ideal trace-back depth for ideal error correction is infinity, which is not practically feasible as this would require infinite memory and introduce infinitely long latency (CCSDS, 2012). The encoder data stream is, therefore, broken up into reasonably sized blocks during the decoding process for practically achievable Viterbi decoding. The block size *aka* truncation length or trace-back depth is varied to determine the value to be used in this decoder.

The results obtained are as illustrated in Figure 5.3.2, which shows the error correction performance of the decoder over 6 different truncation lengths (TB).

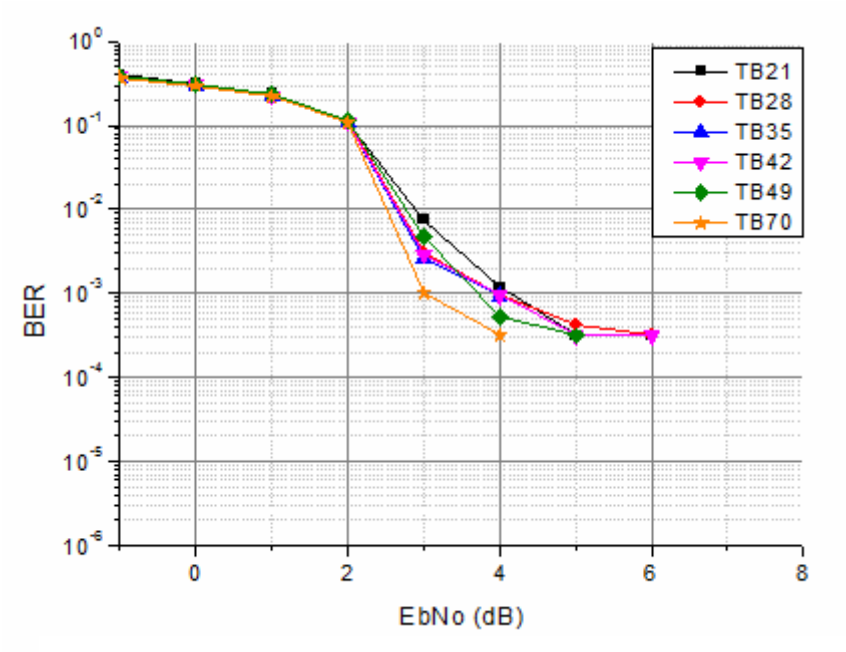


Figure 5.3.2: Decoder error rates for variable trace-back depths for constraint length *K* = 7 (TB = 3K, 4K, 5K, 6K, 7K and 10K)

According to Viterbi decoding literature, the acceptable trace-back depth is usually approximately 5 times the constraint length (CCSDS, 2012). To allow clearer analysis of the error correction capability, each of the trace-back depth values is compared to the recommended truncation length of 35 and the results illustrated in Figure 5.3.3.

Figure 5.3.2 and Figure 5.3.3 show that the decoder error correction capability is nearly identical using all the different TB values when E_b/N_o is less than 2dB. TB35, TB49, and TB70 are identified as having the best BER performance when the E_b/N_o value is 3 and are, therefore, selected to be used for further analysis. TB42 has a BER performance, which is extremely close to the recommended TB35 for all E_b/N_o values, and is not considered for further analysis. Figure 5.3.4 illustrates the BER performance of the selected TB values. The 3 selected TB values are applied in further decoder analyses described in the following sections to determine their respective costs of implementation in terms of coding gain, resource and power utilisation.

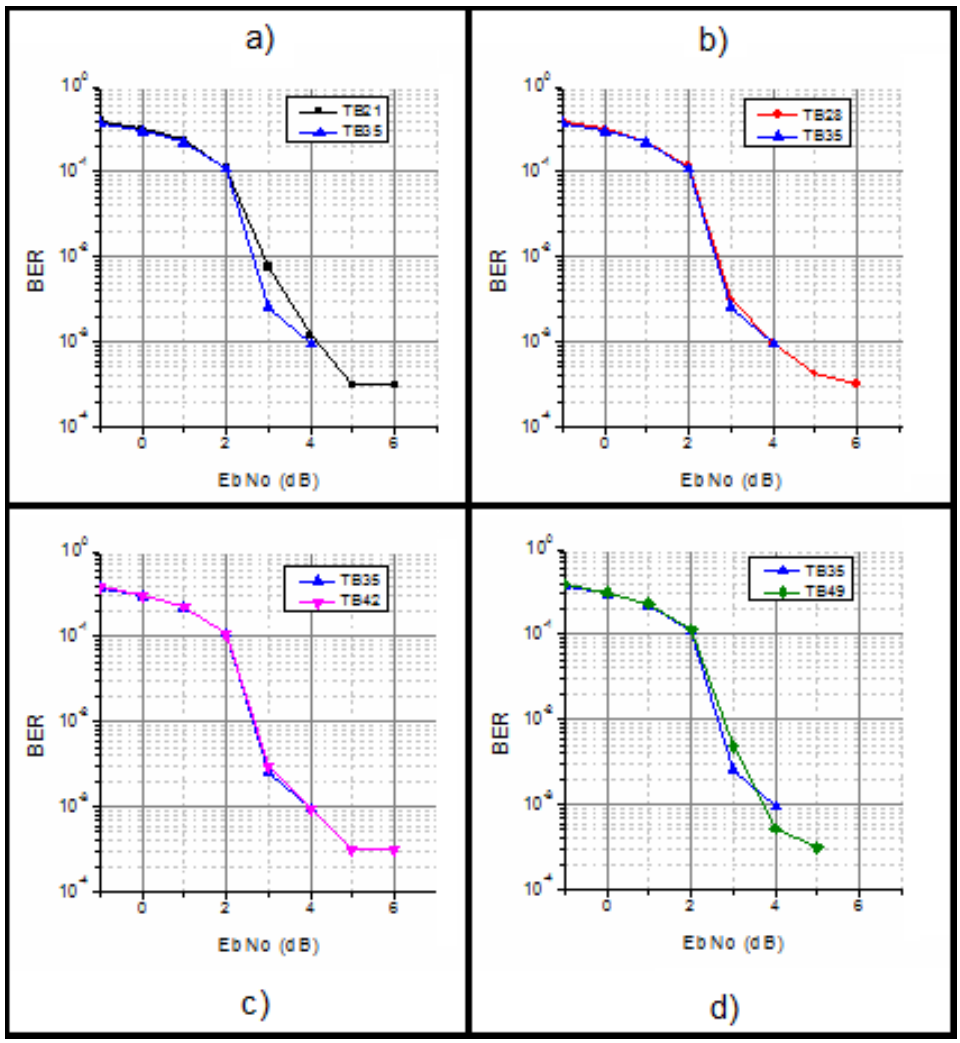


Figure 5.3.3: Error correction for TB21 to TB49 compared to recommended Trace-back TB35

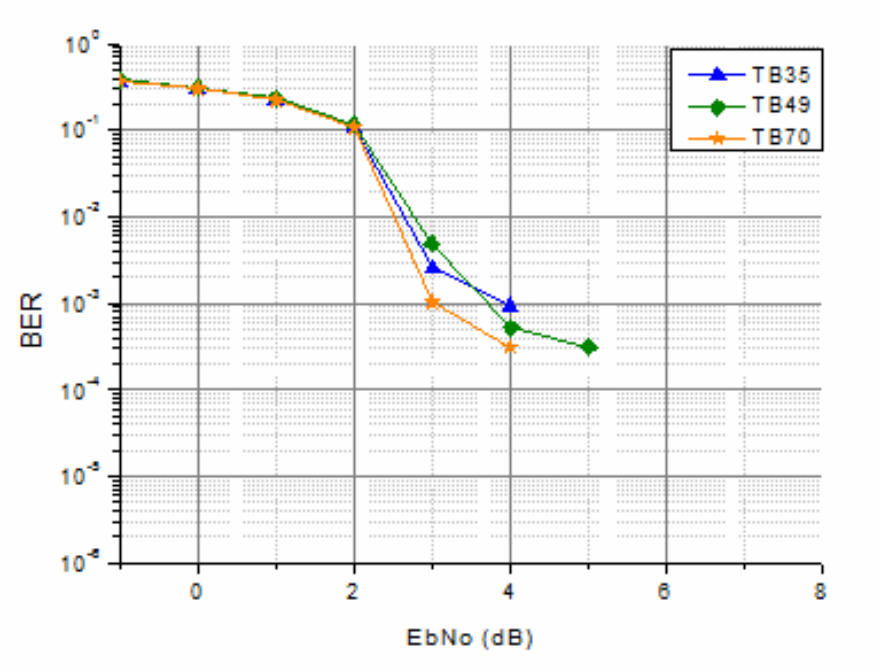


Figure 5.3.4: Error correction for Trace-back depth TB70, TB35 and TB49

5.3.2 Coding Gain

The same input sequence from the data generator is passed through the AWGN model with identical properties to the one used in stage 2 of the test system to determine coding gain. The results obtained are as illustrated in Figure 5.3.5, which shows that implementation of coding on a channel with E_b/N_o values below 2dB increases the resultant occurrence of errors as opposed to reducing them. A coding gain of approximately 3dB is observed at a BER of 10^{-3}.

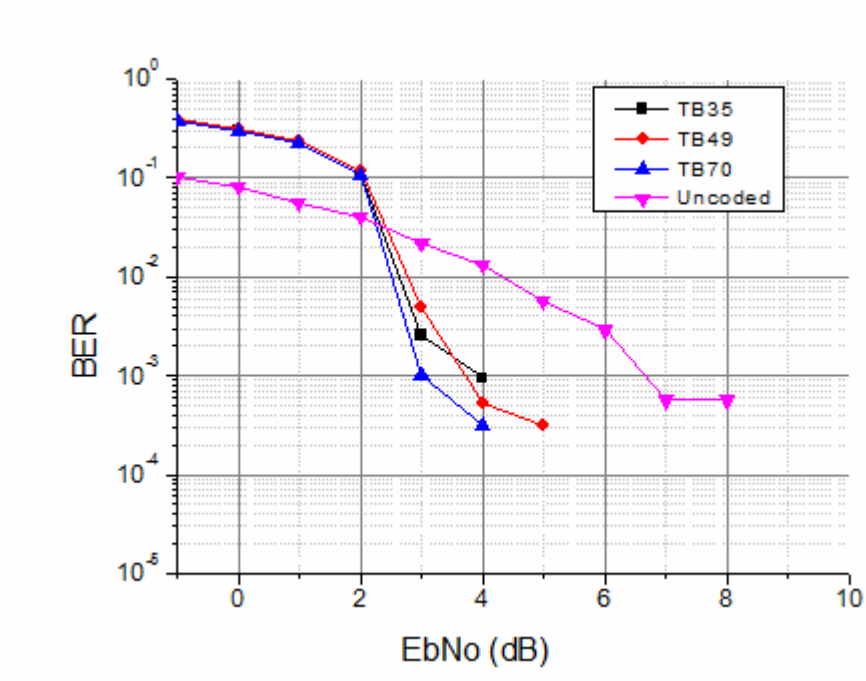


Figure 5.3.5: Un-coded BER performance versus Coded BER performance for TB35, TB49, and TB70

5.3.3 Resource utilisation

In this section, the decoder resource utilisation of the test devices is observed for the selected TB values. Table 5.3.1 shows a summary of the resource utilisation when implemented with the 3 selected TB values. It should be noted that the number of dual port RAM modules on the Igloo2 device, which was used for the encoder analysis, are insufficient for the decoder implementation with TB = 70. A larger device (M2GL025T) from the same family and with the same architecture and packaging specifications is, therefore, selected for TB70 decoder analysis from this point forward.

A change in TB value only has an effect on the decoder memory requirement; however, a miniscule difference in look-up table (LUT) utilisation is observed for TB35 and TB49, which have an identical utilisation of registers as expected. The LUT utilisation difference might be attributed to a change in the constant stored in order to instruct the decoder of the truncation length to be used, but a more in-depth analysis is required. TB70, on the other hand, shows a difference in the number of LUT's and registers used and this difference might also be attributed to the synthesiser inference of logic and optimisation methods employed. The noted differences may also be attributed to the logic used to interface the RAM and the rest of the decoder.

The same design is implemented on all the devices; however, the number of registers and LUT's used differ as a result of the different synthesisers used to infer the hardware from the VHDL description. This can be attributed to differences in the synthesiser synthesis parameters and optimisation methods. The difference in the number of LUT's and registers for the same VHDL description on different devices can also be attributed to the differences in the configuration of the technology available on each device and how logic resources are grouped.

The logic distribution analysis reveals that the maximum resources are used by combinatorial logic (look-up table) and that many registers are still available. Furthermore, less than 3% of the logic blocks are used for routing purposes. This means that, with careful floor planning and maybe some architectural modification, the implementation can fit in smaller circuits. The difference between the number of registers of the various architectures may be a result of different logic synthesisers and implementation usage of the register replication feature. Logic replication duplicates the same logic in order to improve the speed of the circuit.

The memory utilisation difference for the various TB values is only substantial if the number of memory bits used in the utilised memory blocks is considered. The memory blocks used in each instance for the selected devices are not utilised to capacity, as the different TB values require memory bits given in Table 5.3.3. The results of the memory utilisation analysis given in Table 5.3.4, show that a small percentage of the bits available in the memory blocks are allocated to the decoder. Similar to the registers and LUT's, the RAM implementation can be optimised to increase utilisation efficiency.

Table 5.3.1: Decoder resource utilisation

Device	Type	TB35	TB49	TB70	Available
Artix7	Registers (Flip Flops)	2812	2812	2,813	126,800
	LUT's	5262	5,210	5,174	63,400
	Global	1	1	1	32
	I/O	6	6	6	285
	RAM blocks	3	3	3	135
Cyclone V	Registers (Flip Flops)	3661	3661	3655	37736
	Logic LUT's	5569	5558	5563	18860
	Global	2	2	2	16
	I/O	6	6	6	128
	Memory blocks RAM (10k)	6	6	6	176
Igloo2		**M2GL010T**			
	Registers (Flip Flops)	3252	3252	-	12084
	Logic LUT's	6112	6113	-	12084
	Global	2	2	-	8
	I/O	6	6	-	231
	RAM blocks	12	12	-	22
		M2GL025T			
	Registers (Flip Flops)	-	-	3684	27696
	Logic LUT's	-	-	6561	27696
	Global	-	-	2	16
	I/O	-	-	6	265
	RAM blocks	-	-	24	34

Table 5.3.2: Flip-flop and LUT usage and distribution

Device	Type	TB35	TB49	TB70
Artix7	**LUT's**	**5,262**	**5,210**	**5,174**
	Logic	5,126	5,125	5,130
	Route through	136	85	44
	Flip Flop LUT pairs	**5,801**	**5,833**	**5,932**
	Fully used	2,032	1,999	1,910
	With unused FF	3,230	3,211	3,264
	With unused LUT	539	623	758
Cyclone V	**LUT's**	**5569**	**5558**	**5563**
	Logic	5454	5454	5458
	Route through	115	105	105
	Registers	**3661**	**3661**	**3655**
	Design	3536	3536	3536
	Routing	125	125	119
Igloo2	**LUT's**	**6112**	**6113**	**6561**
	Fabric logic	5680	5681	5697
	RAM interface logic	432	432	864
	Flip Flops	**3252**	**3252**	**3684**
	Fabric logic	2820	2820	2820
	RAM interface logic	432	432	864

Table 5.3.3: Decoder RAM requirement

	Required memory bits		
	TB35	**TB49**	**TB70**
Width	64	64	64
Length	35	49	70
Memory bits	6720	9408	13440

Table 5.3.4: Utilisation of allocated RAM blocks

	Used Device memory			
	Artix7	**Cyclone V**	**Igloo2 xxxx025T**	**Igloo2 xxxx025T**
Block Size (kb)	36	10	1	1
# of blocks	3	6	12	22
Total # of bits	110592	61440	13824	25344
	% of memory bits utilised by decoder			
TB35	6.08	10.94	48.61	-
TB49	8.51	15.31	68.06	-
TB70	12.15	21.88	-	53.03

5.3.4 Timing analysis

The critical path of the decoder is the path with the longest delay from net to net between registers with the same clock input. The logic along this slowest path determines the system clock frequency of the decoder. The maximum frequency of operation of the decoder is determined for the different trace-back depth values and the results are as shown in Table 5.3.5 for the selected TB values and test devices. The frequency values displayed below are obtained when the design timing constraint is set to 50MHz. By default, the Quartus II time quest analyser uses multiple process operating condition models for timing analyses and provides the results of each analysis. These models give the best and worst case maximum frequencies where the best (fastest) case is obtained using a model for the highest mobility silicon at 0^0C for the device speed grade. The worst case is obtained using the model for the lowest silicon mobility at 85^0C (Altera Corporation, 2010). The timing analysis tools used for the Artix7 and the Igloo2 devices consider the change in the process operating conditions and give the obtained worst case maximum frequencies (Microsemi Corporation, 2011; Xilinx, Inc, 2012).

Table 5.3.5: Maximum decoder system frequencies (in MHz)

TB	Cyclone V		Artix7	Igloo2
	Best Case	Worst Case		
35	267.24	123.76	123.198	117.233
49	267.95	120.29	126.550	113.934
70	256.61	119.49	113.186	117.800

Although the Viterbi decoding algorithm is continuous, its practical implementation is not. The decoder accepts its input sequence divided into blocks of a specified length. The truncation length determines the throughput of the decoder and ultimately the suitable data rate of the system connected to the decoder. The length of the blocks, therefore, determines the delay between the first decoder input and first usable decoder output. The data rate of the system responsible for feeding data into the decoder is determined by the delay introduced during the calculation of the minimum path metric for trace-back calculations at the end of every input block.

It takes 6 clock cycles for a decoder input, which is made up of 2 bits, to be processed through all the decoder sub-blocks and providing the decision bit information for the TB unit. The first valid decoder output is available after the decoder has decision bits for an input block of length TB, as described in the decoder algorithm. In order to start obtaining the decoder outputs, the TB unit requires the computation of a minimum path metric using a pipeline method that requires 5 clock cycles. The latency/lag T_d of the decoder can, therefore, be calculated as

$$T_d = (6\mathrm{TB} + 5)\, T_{clk} \qquad \textbf{5.3.1}$$

where T_{clk} is the decoder system clock period. The results of the calculation for the 3 selected TB values are given in Table 5.3.6. The table also shows the minimum latency in nanoseconds obtained using the deduced decoder maximum frequency. The worst case maximum frequency is used for the Cyclone V minimum latency calculation.

Table 5.3.6: Decoding latency in clock cycles and minimum latency for obtained maximum frequencies

	TB35	TB49	TB70
Latency (clock cycles)	215	299	425
	Minimum latency (ns)		
Artix7	1745.16	2362.7	3754.88
Cyclone V	1737.2	2485.66	3556.78
Igloo2	1833.95	2624.33	3607.81

5.3.5 Power analysis

Based on the gate count of the placed design, the gate level power analysis gives an indication of the amount of the power required for the design. The best way to obtain an accurate power analysis is to use the switching activity from gate level simulation results, which are difficult to obtain for large designs (Flynn, 2004). In such cases, the vectorless estimation technique is applied for power analysis. The power analysis tools use statistical methods to determine the most probable activity for the nets in the design (Microsemi Corporation, 2014). The analysis process is iterative and stops when all the signals have some form of activity; thus, making the result the worst case power consumption.

As discussed earlier, the total power consumed by a device for a particular design is the sum of the dynamic and static power. The dynamic power refers to the power consumed as a result of the circuit activity, while the static power is attributed to the leakage power consumed when the device is idle. The static power can be used to determine the minimum power required by the selected device. The combination of the static and dynamic power gives the possible maximum power required when the design is at maximum activity (Xilinx, Inc, 2013).

Figure 5.3.6 through Figure 5.3.8 illustrate the variation of the total power usage for the three FPGA devices as a function of system frequency. These figures are labelled as displaying total thermal power, because the total power dissipated within the device is also referred to as thermal power. This total thermal power is generally made up of static power, dynamic power and I/O power (Intel Corporation, 2015).

Vectorless estimation is used for the Igloo2 power analysis, whereas Value Change Dump (VCD) and Switching Activity Interchange Format (SAIF) files are created from the post-PAR simulation for the other Cyclone V and Artix7 devices, respectively. The Igloo2 device used the least amount of power for the same design with a larger percentage of the total power attributed to dynamic power. This might be attributed to the difference of technologies. First, the Igloo2 is specifically designed for low power applications. Second, the Cyclone and Artix families use static RAM to store their configurations, whereas the Igloo2 family uses flash memory that does not consume any power at all to retain its values.

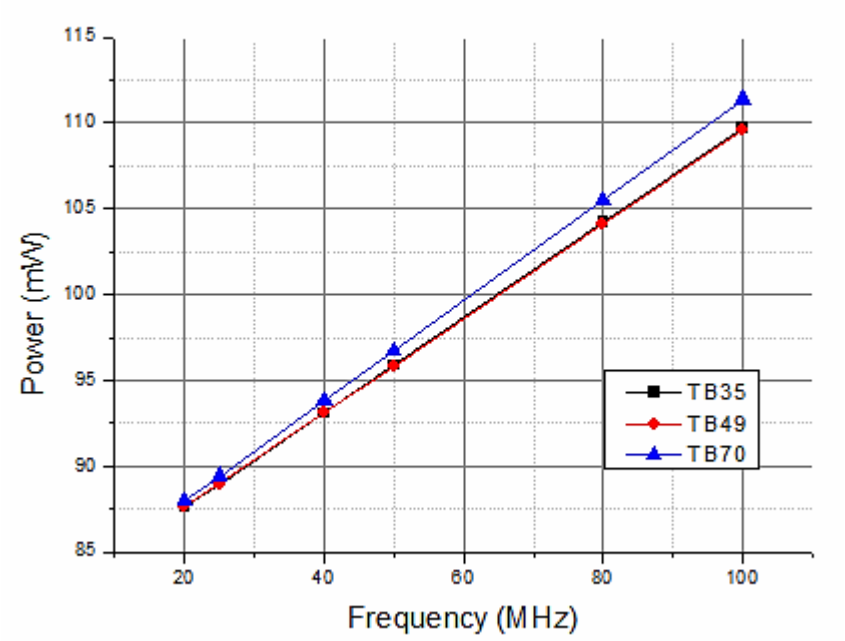


Figure 5.3.6: Artix7 total thermal power consumption

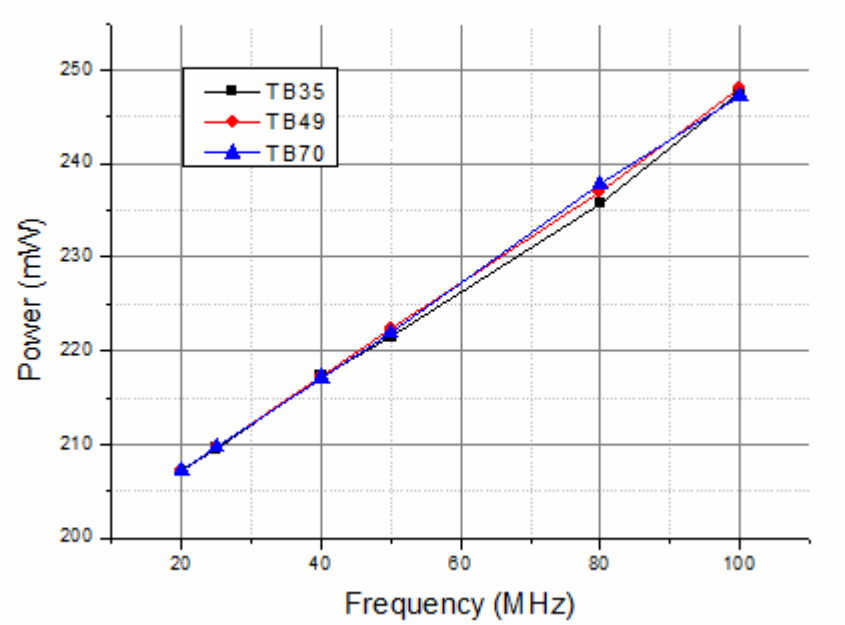


Figure 5.3.7: Cyclone V total thermal power consumption

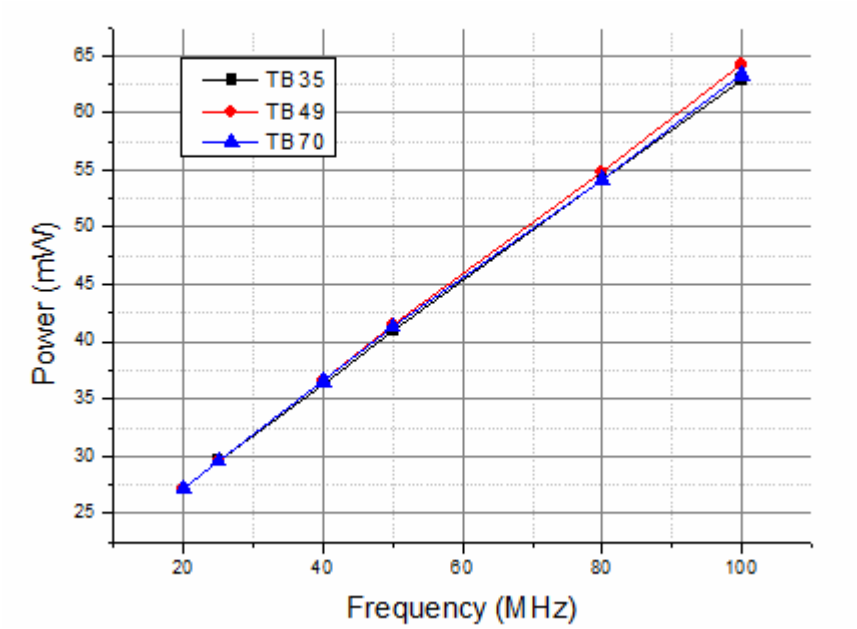


Figure 5.3.8: Igloo2 total thermal power consumption

The dynamic power is the cause of the linear increase observed in the figures above, since the static power and I/O power components are not affected by changes in operating frequency. To determine the effects of the TB variation on the power consumption, Figure 5.3.9 through Figure 5.3.11 illustrate the variation of dynamic power consumption with frequency for the 3 test FPGA devices.

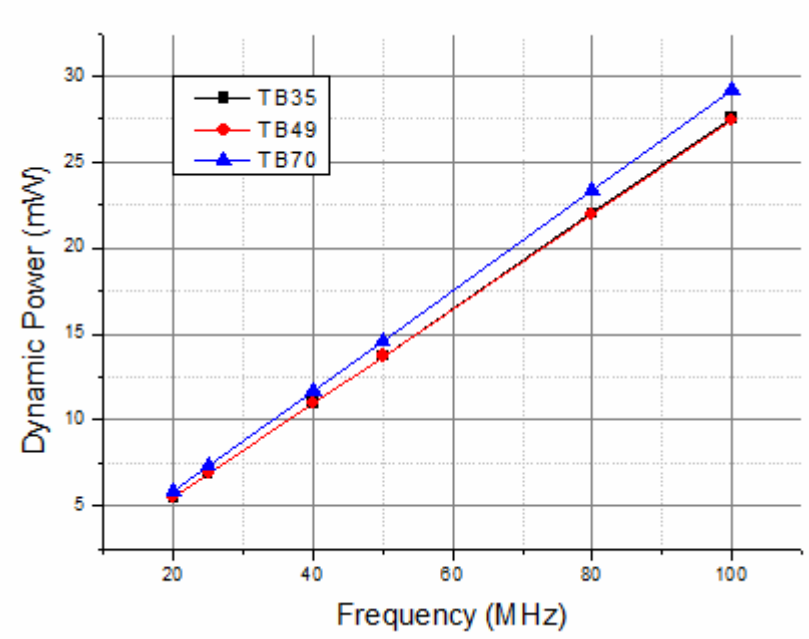


Figure 5.3.9: Artix7 dynamic power consumption

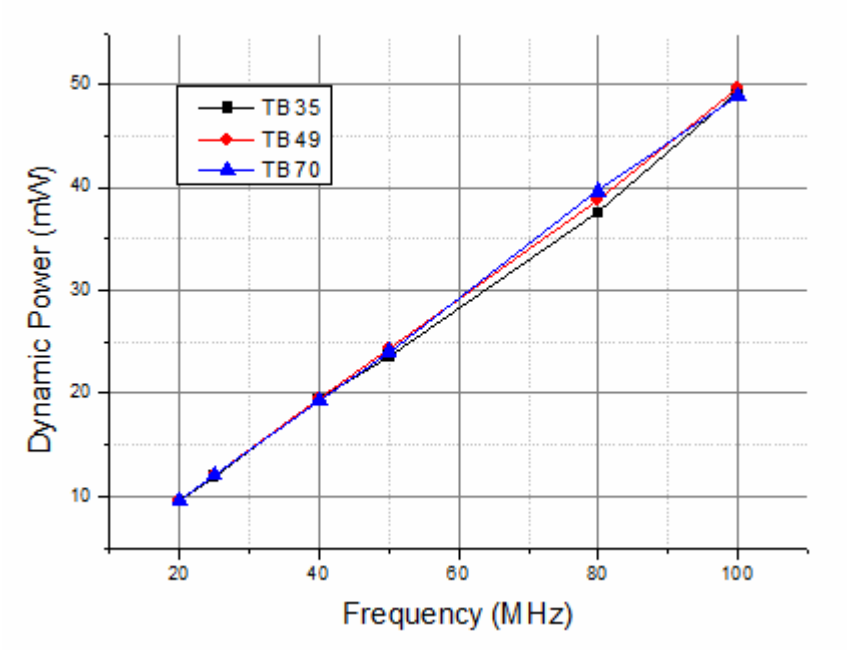


Figure 5.3.10: Cyclone V dynamic power consumption

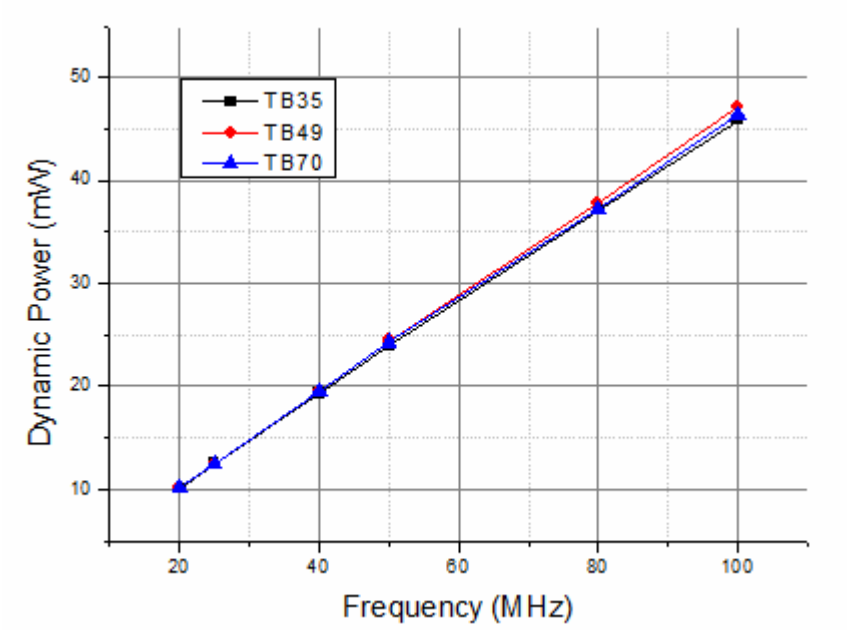


Figure 5.3.11: Igloo2 dynamic power consumption

The variations in dynamic power are minimal over the TB variation for all the 3 devices. In the Artix7 device, however, TB70 exhibits a visibly higher power requirement over TB35 and TB49. The decoder suitable for this device would be one with either a truncation length of 35 or 49. For the other 2 devices, the selection of decoder parameters would rely on the decoding lag introduced by the TB values. A reasonably fast and low-cost implementation for each device would be a decoder with a truncation length of 35 with a system frequency of ~60 MHz. This decoder would incur a decoding lag of approximately 3583.3 ns.

5.4 Conclusion

The decoder architecture and VHDL code have been successfully developed for implementation on an FPGA device. The selection of the decoder parameters can be conducted using the results obtained from the decoder simulations given in this chapter. The variation of decoder truncation length has minimal effect on the resource requirement of the decoder. This is due to the methods employed during the decoder design phase. Future improvements can, therefore, be made to make the decoder more compact and device-specific. Possible future considerations are given in the following chapter.

CHAPTER SIX
CONCLUSION & RECOMMENDATIONS

An operational channel convolutional coding algorithm and its corresponding decoder, which are described in the CCSDS recommended TM synchronisation and channel coding standard, are successfully developed for implementation on an FPGA. The encoder and decoder algorithms are designed for implementation on three test FPGA devices from different popular vendors to determine the cost of implementation on devices that have a high likelihood of being found in nano-satellite communication systems. This reduces the cost of implementation by eliminating the need for additional hardware in the existing communication systems.

Concluding remarks are made in reference to the research questions posed in this work.

In response to the research sub-question *"Are there CCSDS channel coding techniques defined for implementation on nano-satellites?",* it is evident from Chapter 3 that the CCSDS has no specific encoding/decoding algorithm explicitly recommended for nano-satellites. The implemented convolutional code is, therefore, selected as a result of the performance and trade-off analysis outlined.

To answer the subsequent sub-question *"How is FEC coding evaluated and how does that affect the criteria used to select one for implementation in a satellite communication system?",* the basic CCSDS recommended FEC codes (LDPC, Reed-Solomon, Turbo and Convolutional) are compared in terms of coding gain, code rate and BER performance, as well as implementation complexity. With reference to Figure 3.3.1 and Figure 3.3.2, LDPC and Turbo codes exhibit superior performance. However, their implementation has a high cost in terms of complexity, as well as large computation and memory requirements. These are undesirable for the implementation on nano-satellites where resources are limited. The less complex convolutional and Reed-Solomon codes are, thus, more suitable for nano-satellite implementations.

Convolutional codes are selected for implementation as a result of a trade involving error correction strengths, decoder complexity, and the learning curve involved in grasping the encoder and decoder implementation concepts. Convolutional codes, furthermore, outperform RS codes by achieving an approximate coding gain of 5dB for BPSK modulation at a BER of 10^{-6}, compared to the 4dB observed for Reed-Solomon codes. Convolutional coding provides satisfactory coding gain, while requiring reasonable resources in terms of implementation complexity, power and spectrum, as seen from Table 3.3.1 and Figure 3.3.2.

The selected convolutional code exhibits an acceptable E_b/N_o with minimal bandwidth expansion, which is desirable for bandwidth and power constrained nano-satellites implementations.

To respond to the other research sub-question *"What resources are required for FEC implementation and how can their use be optimised or minimised?",* the implementation of FEC coding in a communication system requires the availability of sufficient memory, power and physical space. As mentioned in the objectives, the algorithm is to be implemented on an adaptable and reliable platform that takes into account the nano-satellite constraints; therefore, an FPGA is selected as the implementation platform. FPGA's are also reasonably sized and used in numerous modern nano-satellite communication systems, making them a suitable platform for implementing the FEC algorithm on existing communication systems. They also have large memory capacity, which makes them suitable for FEC decoding algorithms that require a significant amount of memory. Low power test FPGA's have been selected for the encoder/decoder implementation, which makes them suitable for nano-satellite implementations.

To answer the sub-question *"What is the trade-off between performance and resource utilisation appropriate for a nano-satellite?", o*ne of the main aims of this research has been to establish the cost of implementing error correction codes on-board a satellite communication system. Convolutional encoding adds maximum power consumption of less than ~9mW to an existing communication system due to encoding activity, as has been shown in Figure 4.3.3. From Table 4.3.2, it is evident that the encoder occupies a negligible area on the FPGA device, occupying a maximum of ~0.099% of the device LUT's and ~0.12% of the device registers on the smallest igloo test FPGA. The cost of implementing the encoder is, therefore, reasonably low and makes it feasible for nano-satellite implementations.

On the other hand, the decoder requires a significant percentage of the FPGA resources as can be deduced from Table 5.3.1. For instance, implementing the decoder with a truncation length of 49 on the Igloo2 device requires ~51% of the device LUT's and ~26.9% of the device registers. The decoder also adds power consumption of up to ~50mW due to decoder activity when the decoder is operating at 100 MHz as deduced from the illustrations in Section 5.3.5. The cost of implementing the decoder is, therefore, significantly higher than that of the encoder. However, the selection of the decoder parameters with a feasible cost in relation to the available communication system can be conducted using the results presented in Section 5.3. With the availability of the decoder parameters, the communication system

designer has the ability to add it to their design and evaluate the resources available for the whole system.

The results of the decoder analysis in Figure 5.3.5 confirm the notion that at some SNR or E_b/N_o values, adding error correction to the communication system results in the increase of BER as opposed to the desired reduction. According to literature, hard decision convolutional codes introduce a coding gain of approximately 3dB, which is also confirmed by the performance evaluation of the implemented decoder in Figure 5.3.5. This fulfils the objective pertaining to the improvement of communication system reliability and data transmission rates.

In a nutshell, this research successfully answers the primary research question *"How can a reliable and well-performing CCSDS compliant forward error correction encoder and decoder be selected and implemented on a nano-satellite?"* The main project objective *"to implement a forward error correction (FEC) algorithm, which conforms to existing standards on an adaptable and reliable platform"* has also been successfully realised.

As a recommendation, soft decision decoding is more accurate and, therefore, introduces a greater coding gain than the implemented hard decision decoding. Further studies and investigations can be made into evaluating the cost of implementing a soft decision decoder on the same FPGA devices. The translation, mapping and routing of the design onto the FPGA devices can be optimised, resulting in a reduction in power and area usage. Optimisation methods can, therefore, be investigated and implemented to improve the decoder performance and reduce its cost of implementation. The implemented encoder and decoder have been designed such that they are portable across FPGA platforms; defining the hardware using device specific descriptions would also improve the resource utilisation efficiency.

The power consumption results given in this document have been obtained using simulations involving I/O signal activity and numerous statistical assumptions. As a further recommendation, determining the actual device performance requires the acquisition of the physical hardware and measuring the power consumption to improve the accuracy of the implementation cost data.

REFERENCES

Altera Corporation. 2010. *Guaranteeing Silicon Performance with FPGA Timing Models*. https://www.altera.com/en_US/pdfs/literature/wp/wp-01139-timing-model.pdf [14 September 2017].

Altera Corporation. 2012. *Reducing Total System Cost with Low-Power 28-nm FPGAs*. https://www.altera.com/en_US/pdfs/literature/./wp-01180-system-cost-low-power.pdf [16 September 2016].

Altera Corporation 2016. *Cyclone V Device Overview*., Document (v2016.06.10).

Anderws, K., Dolinar, S. & Thorpe, J. 2005. Encoders for Block-Circulant LDPC Codes. *International Symposium on Information Theory*. Adelaide, 2005. IEEE [2 September 2016]

Andrews, K.S., Divsalar, D., Dolinar, , Hamkins, J., Jones, C.R. & Pollara, F. 2007. Turbo and LDPC Codes for Deep-Space Applications. *Proceedings of the IEEE*, 95(11), 2142-56.

Divsalar, D., Abbasfar, A., Jones, C.R., Dolinar, S.J., Thorpe, J.C., Andrews, K.S. & Yao, K. 2009. *Encoders for Block-Circulant LDPC codes*. https://www.google.com/patents/US7499490 [2 September 2016]

Atlanta RF 2013. Link Budget Analysis: Error Control & Detection. http://www.atlantarf.com/Error_Control.php [9 April 2016]

Calhan, , Ceken, C. & Erturk, I. 2007. Comparative performance analysis of forward error correction echniques used in wireless communications. *Third international conference on wireless and mobile communications, ICWMC '07*. Guadeloupe, 2007. IEEE

Calhan, A., Ceken, C. & Erturk, I. 2009. A teaching demo application of convolutional coding techniques for wireless communications. *Application of information and communication technologies, International conference on*. Baku, 2009. IEEE

California Polytechnic State University 2009. *CubeSat design specification*. http://www.cubesat.org/images/developers/cds_rev12.pdf [2 October 2013]

Cape Peninsula University of Technology: F'SATI. n.d. http://www.cput.ac.za/blogs/fsati/cubesat/ [1 July 2016].

CCSDS 2007. *Low Density Parity Check Codes for Use In Near-Earth and Deep Space Applications*., CCSDS Secretariat Washington NASA. [31 July 2013]

CCSDS 2011a. *TM Synchronization and channel coding*., CCSDS Secretariat Washington NASA. [6 July 2013]

CCSDS 2011b. *TM Channel coding profiles*., CCSDS Secretariat Washington NASA. [31 July 2013]

CCSDS 2012. *TM Synchronization and channel coding - Summary of concept and rationale*., CCSDS Secretariat Washington NASA. [31 July 2013]

Chen, L., Tang, S. & Ma, X. 2013. Progressive Algebraic Soft-Decision Decoding of Reed-Solomon Codes. *IEEE Transactions on Communications*, 61(2), 433 - 442.

Cillibot, E.P., Grant, C.C., Kekez, D.D. & Zee. 2005. Formation Flying Demonstration Mission Enabled by CanX Nanosatellite Technology. *19th Annual AIAA/Utah State University Conference on Small Satellites*., 2005.

Costello, D.J.J. & Forney, D.G.J. 2007. Channel Coding: The Road to Channel Capacity. *Proceedings of the IEEE*, 95(6), 1150-77.

CPUT *see* Cape Peninsula University of Technology.

de Milliano, M. & Verhoeven, C. 2010. Towards the next generation of nanosatellite communication systems. *Acta Astronautica*, 66(9–10), 1425-33.

de Villiers, & van Zyl, R. 2015. *ZACube-2: The successor to Africa's first nanosatellite*. http://www.amsatsa.org.za/ZACube-2%20%20The%20successor%20to%20Africa%E2%80%99s%20first%20nanosatellite.pdf [17 September 2016].

Divsalar, D. & Pollara, F. 1995. *Turbo Codes for Deep-Space Communications*. http://www-leland.stanford.edu/class/ee379b/class_reader/jpl1.pdf [26 August 2016].

Flynn, J. 2004. *Accurate power-analysis techniques support smart SOC-design choices*. http://www.edn.com/Home/PrintView?contentItemId=4328121 [25 October 2016].

Foerster, J. & Liebetreu, 2000. *FEC Performance of Concatenated Reed-Solomon and Convolutional Coding with Interleaving*., IEEE 802.16. http://ieee802.org/16/tg1/phy/contrib/802161pc-00_33.pdf. [25 May 2013]

Garg, V.K. & Wang, Y.-C. 2005. Intoduction to digital communication and communication networks. In W.-K. Chen. (ed). *The Electrical Engineering Handbook*. Elsevier Academic press. 949-50.

Global Market Insights. 2016. *Field Programmable Gate array (FPGA) Market Size By Application (Data Processing, Industrial, Automotive, Consumer Electronics, Telecom, Military & Aerospace), Regional Outline, Application Potential, Competitive Market Share & Forecast, 215-2022*. https://www.gminsights.com/industry-analysis/field-programmable-gate-array-fpga-market-size [17 May 2016].

Haiman, M. 2003. *Notes on Reed-Solomon Codes*. https://math.berkeey.edu/~mhaiman/math55/reed-solomon.pdf [14 May 2013].

Intel Corporation. 2015. *Power Overview*. https://www.altera.com/support/support-resources/operation-and-testing/power/pow-overview.html [14 September 2017].

Jerrold, H.A. & Jacobs, I.M. 1971. Viterbi Decoding for Satellite and Space Communication. *IEEE transactions on communication technology*, COM-19(5), 835-48. http://ieeexplore.ieee.org.libproxy.cput.ac.za/stamp/stamp.jsp?arnumber=1090711 [30 September 2013].

Koetter, & Vardy, 2003. Algebraic Soft-Decision Decoding of Reed–Solomon Codes. *IEEE Transactions on Information Theory*, 49(11), 2809 - 2825.

Kramer, H.J. n.d. *ZACUBE-1 (South Africa CubeSat-1) / TshepisoSat*. https://directory.eoportal.org/web/eoportal/satellite-missions/v-w-x-y-z/zacube-1 [1 July 2016].

Kumar, S. & Gupta, R. 2011. Performance comparison of different forward error correction coding techniques for wireless communication systems. *International journal of computer science and technology*, 2(3), 553-7.

Lafleur, C. 2004. *The Spacecrafts Encyclopedia*. http://claudelafleur.qc.ca/Spacecrafts-index.html#Stats [30 September 2015].

Li, Z., Chen, L., Zeng, L., Lin, & Fong, W.H. 2006. Efficient Encoding of Quasi-Cyclic Low-Density Parity-Check Codes. *IEEE Transactions on Communications*, 54(1), 71-81.

Lu, Y.-K., Chung, S.-M. & Shieh, M.-D. 2014. Low-complexity Architecture for Chase Soft-decision. *VLSI Design, Automation and Test (VLSI-DAT), 2014 International Symposium on*. Hsinchu, 2014. IEEE

Madhow, U. 2008. *Fundamentals of Digital Communication*. New York: Cambridge University Press.

Microsemi Corporation. 2011. *Advanced Static Timing Analysis Using SmartTime*. https://www.microsemi.com/document-portal/doc_view/129843-ac379-advanced-static-timing-analysis-using-smarttime-app-note [14 September 2017].

Microsemi Corporation 2014. SmartFusion2 System-on-Chip FPGAs. *Engineers' Guide to FPGA and PLD Solutions*, 27. [17 May 2016].

Microsemi Corporation 2014. *SmartPower for Libero SoC Software v11.4*.

Microsemi Corporation 2016. *IGLOO2 FPGAs Product Brief*., Product brief (v13).

Microsemi Corporation 2016. *SmartFusion2 SoC and IGLOO2 FPGA Fabric*., User guide (v4).

MIT 2005. Introduction to convolutional codes. https://ocw.mit.edu/courses/electrical-engineering-and-computer-science/6-451-principles-of-digital-communication-ii-spring-2005/video-lectures/chap9.pdf [1 September 2017]

MIT 2010a. Convolutional Coding. http://web.mit.edu/6.02/www/f2010/handouts/lectures/L8.pdf [5 September 2017]

MIT 2010b. Viterbi Decoding of Convolutional Codes. http://web.mit.edu/6.02/www/f2010/handouts/lectures/L9.pdf [5 Septemeber 2017]

Mitchell, G. 2009. Investigation of Hamming, Reed-Solomon and Turbo Forward Error Correcting Codes. www.arl.army.mil/arlreports/2009/ARL-TR-4901.pdf [8 April 2014]

Mitra, M. 2005. *Satellite communication*. Eastern economy ed. New Delhi, India: Prentice-Hall of India.

Morelos-Zaragoza, R.H. 2002. *The Art of Error Correcting Coding*. Chechester: John Wiley & Sons Ltd.

O'Dea, 2013. *Telemetry Data Decoding*. https://deepspace.jpl.nasa.gov/dsndocs/810-005/208/208B.pdf [12 May 2016]

Perry, D.L. 2002. *VHDL: Programming by example*. 4th ed. McGraw-Hill Education.

Polaschegg, M. 2005. Study of a Cube-Sat Mission. physik.uni-graz.at/spacesciences/archive/files/ULG_II_Master_Thesis_Polaschegg.pdf [11 February 2014]

Riley, M. & Richardson, I. 1996. *Reed-Solomon Codes*. https://www.cs.cmu.edu/~guyb/realworld/reedsolomon/reed_solomon_codes.html [25 August 2016].

Rogers, A.Q. & Summers, R.A. 2010. Creating Capable Nanosatellites for Critical Space Missions. *Johns Hopkins APL Technical Digest*, 29(3), 283-8.

Rong, B., Wu, Y. & Gagnon, G. 2011. Information theory, Shannon limit and error correction codes for terrestrial DTV broadcasting. https://www.scribd.com/document/248688746/Shannon [26 February 2014]

Shannon, C.E. 1948. A Mathematical Theory of Communication. *The Bell System Technical Journal*, 27, 379–423, 623–656. [30 September 2013].

Sklar, B. 1997. A Primer on Turbo Code Concepts. *IEEE Communications Magazine*, 35(12), 94-102. [14 February 2014].

Sklar, B. 2001. *Digital communication: Fundamentals and applications*. 2nd ed. New Jersey: Prentice Hall.

Sklar, B. & Harris, F.J. 2004. The ABCs of linear block codes. *IEEE Signal Processing Magazine*, 21(4), 14 - 35.

Swartwout, M.A. n.d. *CubeSat Census*. https://sites.google.com/a/slu.edu/swartwout/home/cubesat-database/census [30 September 2015].

Sweeney, P. 2002. *Error control coding: from theory to practice*. Chichester: John Wiley & Sons, Ltd.

Sweeting, M.N. & Underwood, C.I. 2003. Small Satellite Engineering and Applications. In P. Fortescue, J. Stark & G. Swinerd. (eds). *Spacecraft Systems Engineering*. 3rd ed. Chichester: John Wiley & Sons Ltd. 581-610.

Tse, D. & Viswanath, P. 2005. Capacity of wireless channels. In *Fundamentals of Wireless Communication*. 1st ed. Cambridge University Press. 166-227.

Van Zyl, R.R., Visser, F.D., Cilliers, P.J. & Opperman, B.D.L. 2013. ZACUBE-1 Space Weather Mission: Characterize the SuperDARN HF Radar Antenna Array at SANAE-IV. *Space Weather*, 11(2), 52-4. [2014].

Viterbi, A. 1971. Convolutional Codes and Their Performance in Communication Systems. *IEEE Transactions on Communication Technology*, 19(5), 751 - 772. http://ieeexplore.ieee.org.libproxy.cput.ac.za/stamp/stamp.jsp?arnumber=1090700 [1 September 2017].

Wertz, J.R. & Larson, W.J. (eds). 1999. *Space mission and analysis design*. 3rd ed.

Woellert, K., Ehrenfreund, P., Ricco, A.J. & Hertzfeld, H. 2011. Cubesats: Cost-effective science and technology platforms for emerging and developing nations. *Advances in Space Research*, 47(4), 663-84.

Xilinx, Inc. 2014. Artix-7 FPGAs. *Engineers' Guide to FPGA and PLD Solutions*, 28. [17 May 2016].

Xilinx, Inc. 2016. *7 Series FPGAs Memory Resources*., User Guide (v1.12).

Xilinx, Inc. 2012. *Timing Closure User Guide*. http://www-wjp.cs.uni-saarland.de/lehre/hadeprak/block_ss15/Upload/ug612.pdf [14 September 2017].

Xilinx, Inc 2013. *ISE Help*., Xilinx, Inc Xilinx, Inc.

Xilinx 2015. *7 Series FPGAs Overview*., Product Specification (v 1.7).

APPENDICES

APPENDIX A: CUBESAT LAUNCH STATISTICS

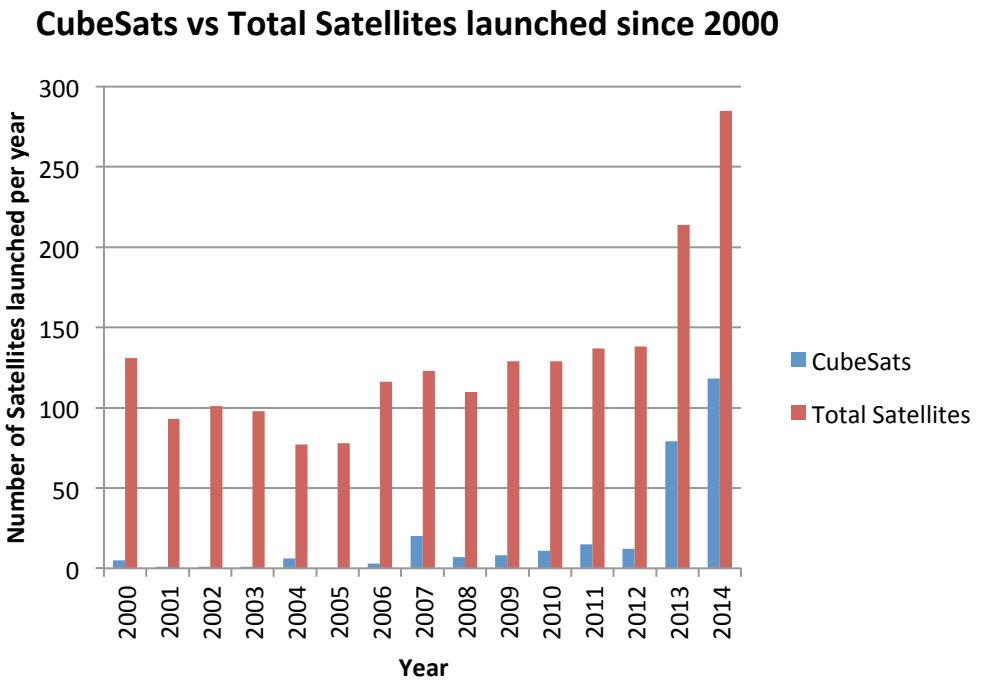


Figure A 1: Number of CubeSats launched versus the total number of satellites launched between 2000 and 2014

Adapted from (Swartwout, n.d.) and (Lafleur, 2004)

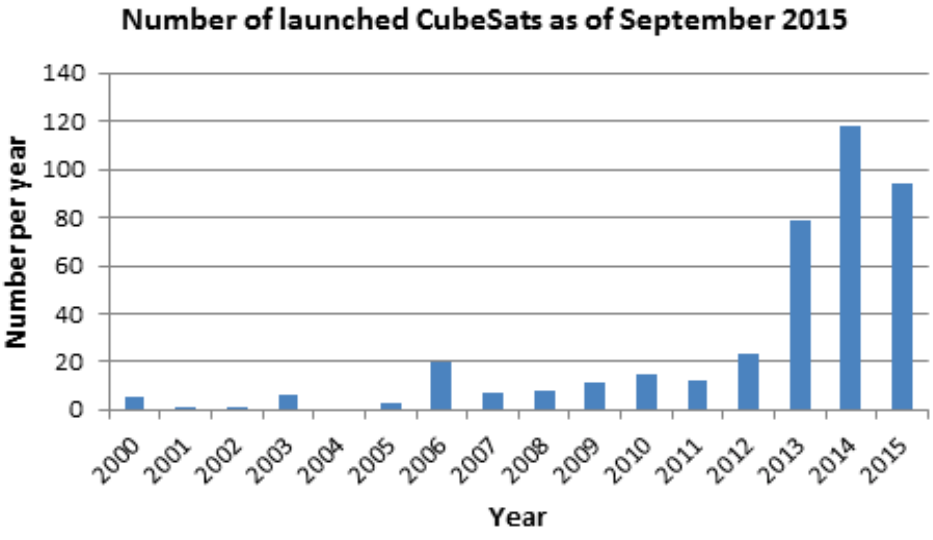


Figure A 2: CubeSats launched between 2000 and 2015

Adapted from (Swartwout, n.d.)

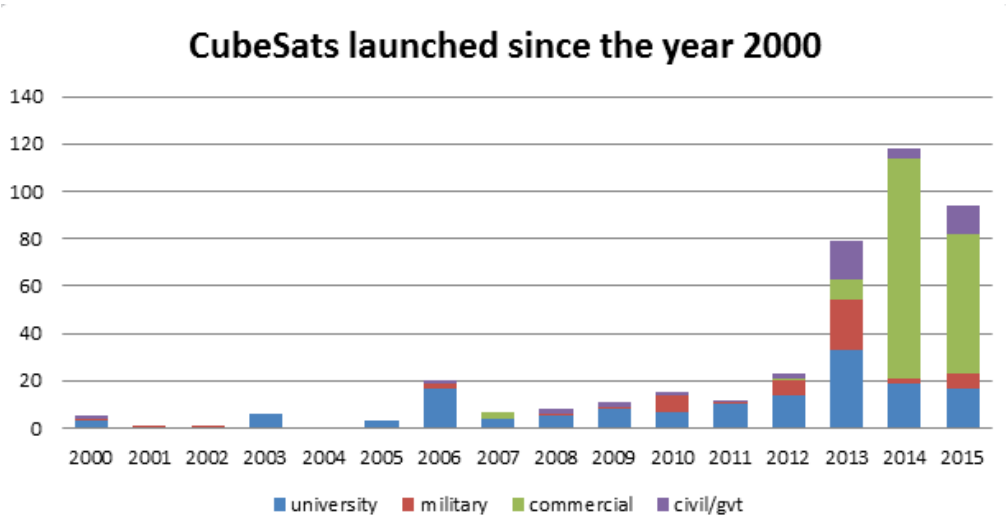


Figure A 3: Number of CubeSats launched between 2000 and 2015 indicating the developing entities

Adapted from (Swartwout, n.d.)

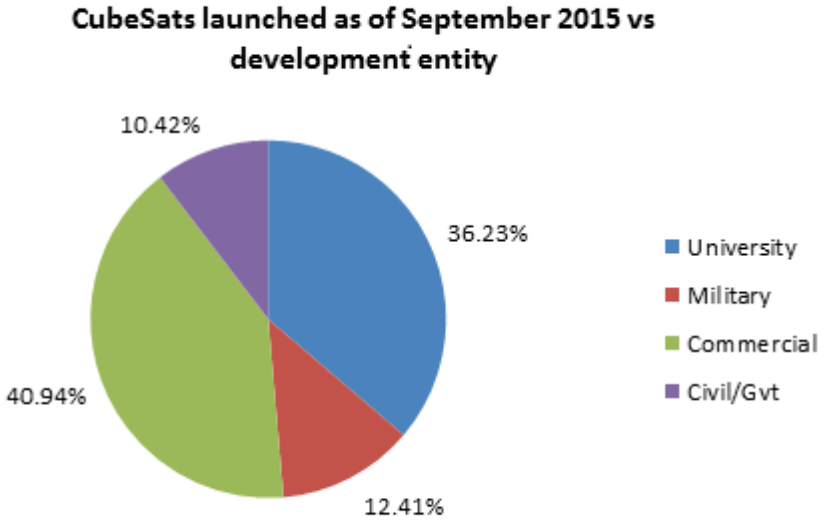


Figure A 4: Indication of entity involvement in CubeSat development using CubeSats launched between 2000 and 2015

Adapted from (Swartwout, n.d.)

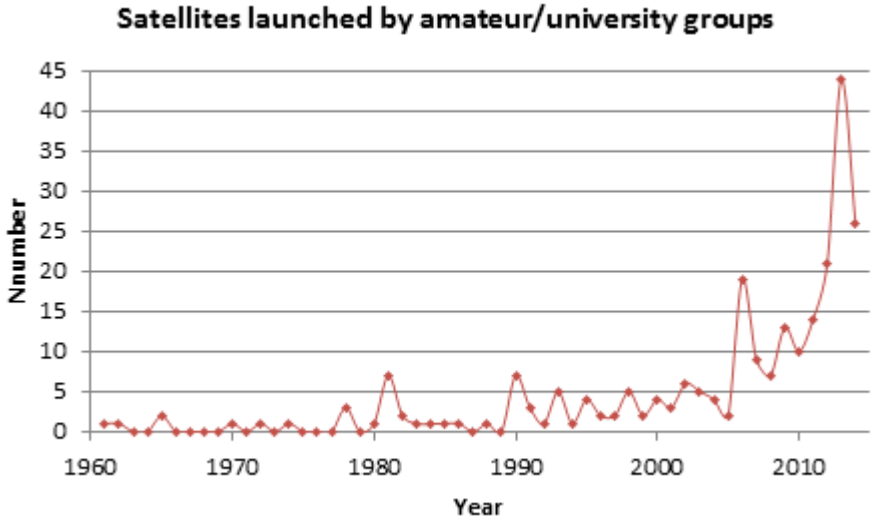
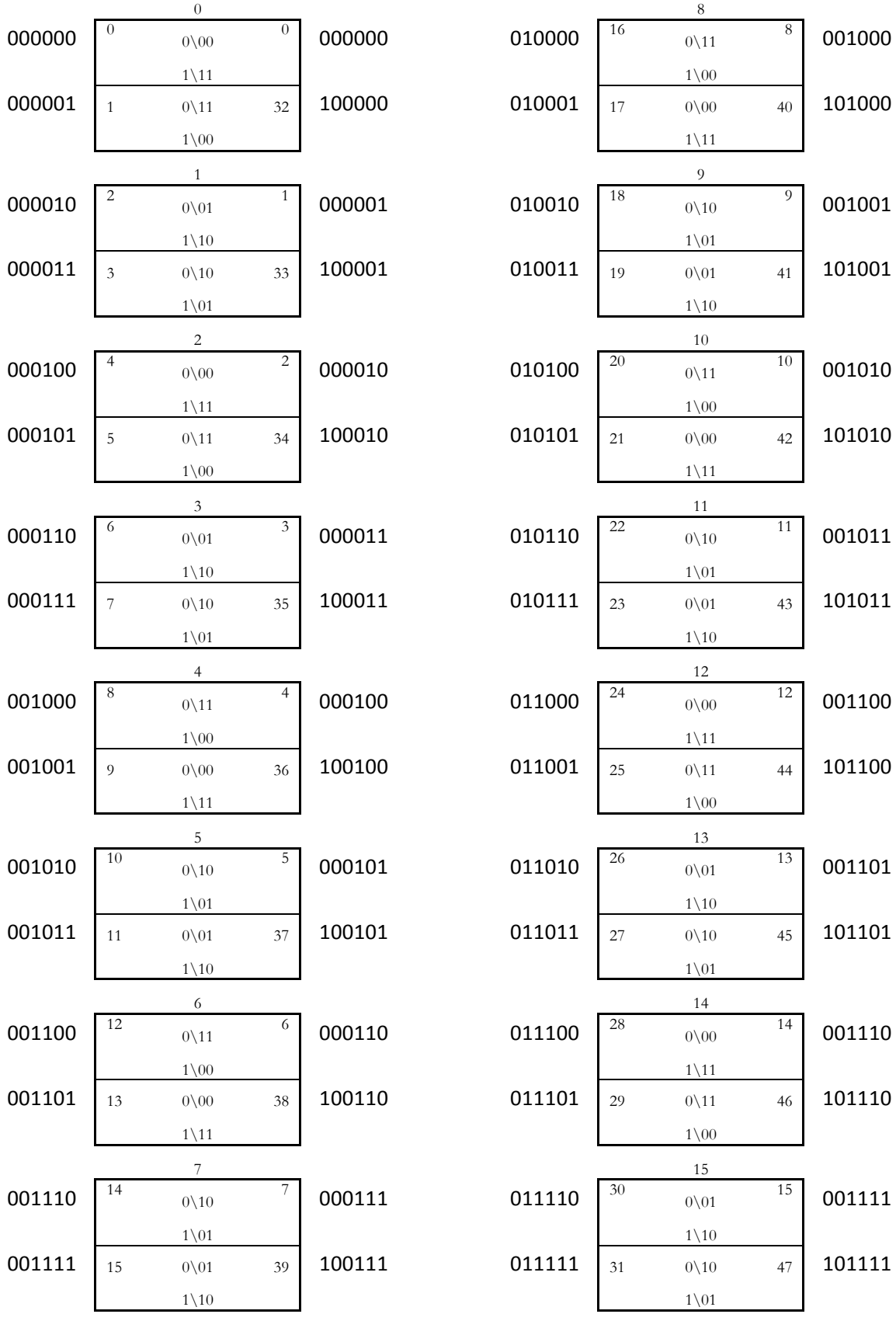
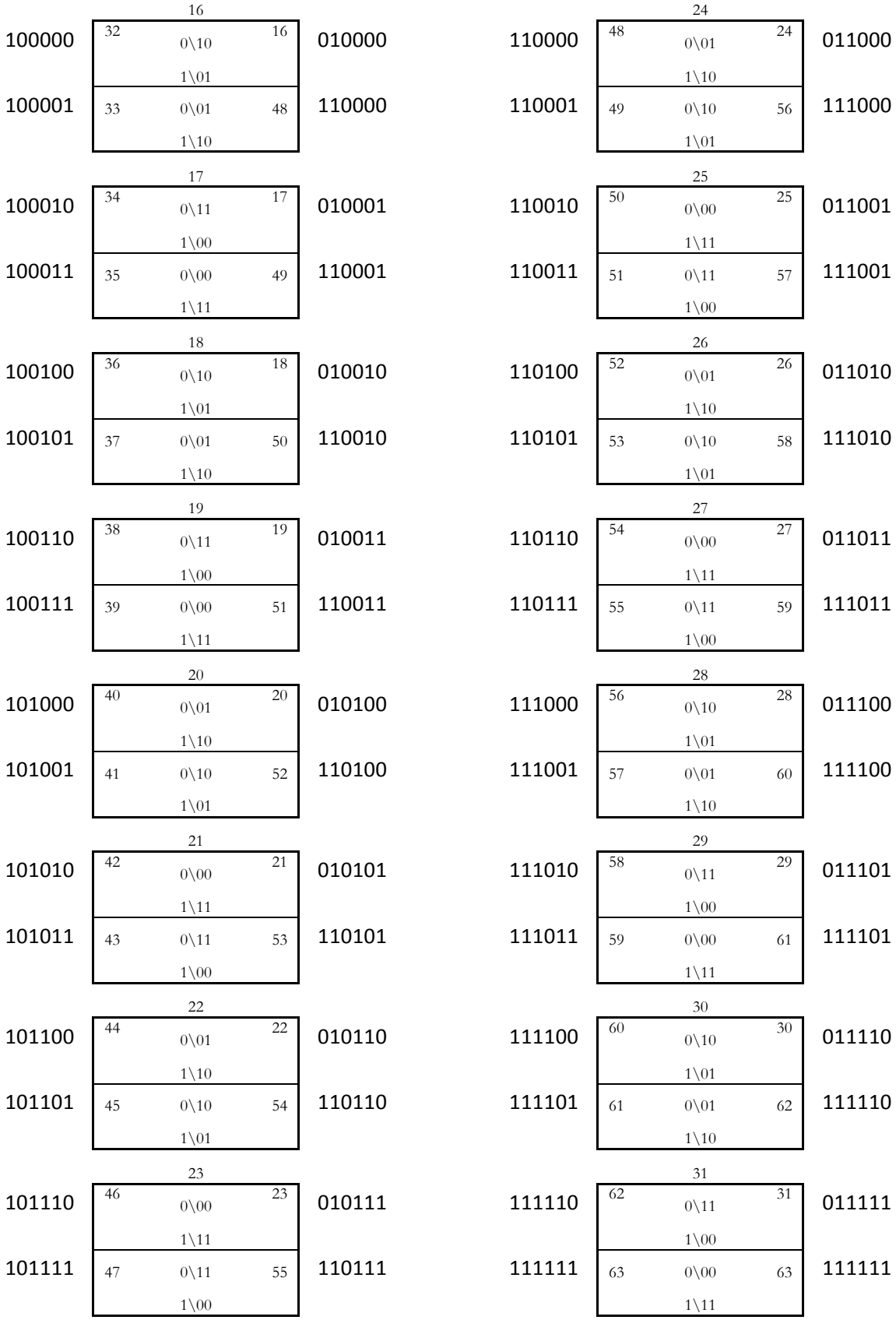


Figure A 5: Satellites launched by university/amateur organisations between 1961 and 2014

Adapted from (Lafleur, 2004).

APPENDIX B: TRELLIS BUTTERFLIES




16
100000
32
0\10
16
1\01
010000
100001
33
0\01
48
1\10
110000
17
100010
34
0\11
17
1\00
010001
100011
35
0\00
49
1\11
110001
18
100100
36
0\10
18
1\01
010010
100101
37
0\01
50
1\10
110010
19
100110
38
0\11
19
1\00
010011
100111
39
0\00
51
1\11
110011
20
101000
40
0\01
20
1\10
010100
101001
41
0\10
52
1\01
110100
21
101010
42
0\00
21
1\11
010101
101011
43
0\11
53
1\00
110101
22
101100
44
0\01
22
1\10
010110
101101
45
0\10
54
1\01
110110
23
101110
46
0\00
23
1\11
010111
101111
47
0\11
55
1\00
110111
24
110000
48
0\01
24
1\10
011000
110001
49
0\10
56
1\01
111000
25
110010
50
0\00
25
1\11
011001
110011
51
0\11
57
1\00
111001
26
110100
52
0\01
26
1\10
011010
110101
53
0\10
58
1\01
111010
27
110110
54
0\00
27
1\11
011011
110111
55
0\11
59
1\00
111011
28
111000
56
0\10
28
1\01
011100
111001
57
0\01
60
1\10
111100
29
111010
58
0\11
29
1\00
011101
111011
59
0\00
61
1\11
111101
30
111100
60
0\10
30
1\01
011110
111101
61
0\01
62
1\10
111110
31
111110
62
0\11
31
1\00
011111
111111
63
0\00
63
1\11
111111

APPENDIX C: ADD COMPARE SELECT (ACS) MODULE INTERCONNECTION