An LDPC Error Control Strategy for Low Earth Orbit Satellite Communication Link Applications

by

Francois Jacobus Olivier

Thesis presented in partial fulfilment of the requirements for the degree of Master of Science in Engineering at the Stellenbosch University

Department of Electrical and Electronic Engineering
Stellenbosch University
Private Bag X1, 7602 Matieland, South Africa

Supervisor: Dr. R. Wolhuter

December 2009

Declaration

By submitting this thesis electronically, I declare that the entirety of the work contained therein is my own, original work, that I am the owner of the copyright thereof (unless to the extent explicitly otherwise stated) and that I have not previously in its entirety or in part submitted it for obtaining any qualification.

December 2009

Abstract

An LDPC Error Control Strategy for Low Earth Orbit Satellite Communication Link Applications

F.J. Olivier

Department of Electrical and Electronic Engineering
Stellenbosch University
Private Bag X1, 7602 Matieland, South Africa

Thesis: M.Sc.Eng (Electronic)

December 2009

Low earth orbit (LEO) satellite communication presents a unique environment which inherently differs from most other communication channels. Due to the varying orbital patterns of LEO satellites the link exhibits varying link margins. Limited communication time windows need to be optimised to maximise the volumetric data throughput.

Large coding gains can be obtained by the implementation of forward error correction codes. This thesis presents a means for optimising the data throughput of LEO satellite communication through the implementation of a mission specific error control strategy. Low density parity check (LDPC) codes are versatile and present good error performances at many different code rates and block lengths. With power limitations on the space segment and remote ground stations, hardware utilisation efficiency must be optimised to reduce power consumption. In response to this requirement, this thesis evaluates various algorithms for LDPC decoders.

An iterative LDPC decoder, implementing an approximation algorithm, is presented as a low complexity solution with good error performance. The proposed solution provides a very good balance between required hardware complexity and coding performance. It was found that many parameters of the decoders and codes can be altered to allow the implementation of these codes in systems with varying memory and processing capabilities.

Samevatting

’n Laedigtheid Pariteitskode Foutbeheerstrategie vir Lae Wentelbaan Satellietkommunikasie.

(“An LDPC Error Control Strategy for Low Earth Orbit Satellite Communication Link Applications”)

F.J. Olivier

Departement Elektries en Elektroniese Ingenieurswese
Universiteit van Stellenbosch
Privaatsak X1, 7602 Matieland, Suid-Afrika

Tesis: M.Sc.Ing (Elektronies)

Desember 2009

Kommunikasiekanale van satelliete met lae wentelbane, bied ’n unieke omgewing wat inherent verskil van meeste ander kommunikasiekanale. As gevolg van veranderende wentelbaanpatrone, vertoon die kanaal ’n wisselende foutgedrag. Kommunikasievensters is beperk en moet geoptimeer word om die totale deurset van die stelsel te maksimeer.

Groot koderingswinste kan verkry word deur die implementering van foutkorreksie kodes. Hierdie tesis voorsien ’n metode om die datadeurset van satelliete met lae wentelbaan te optimeer, deur middel van implementering van ’n missie-spesifieke foutbeheer strategie. Lae digtheid pariteit toetskodes (LDPC) is veelsydige kodes, bied goeie foutbeheer en is doeltreffend vir verskillende kodekoerse en bloklengtes. Met drywingsbeperkinge op die ruimtesegment en afgesonderde grondstasies, moet hardeware komponente doeltreffend gebruik word om drywingsverbruik te verminder. Ten einde aan hierdie ontwerpsvereiste te voldoen, evalueer hierdie tesis verskeie LDPC dekodeerder-algoritmes.

Deur ’n iteratiewe LDPC dekodeerder met ’n benaderingsalgoritme te implementeer, word ’n oplossing van lae kompleksiteit aangebied, maar wat steeds goeie foutkorreksie eienskappe toon. Die voorgestelde oplossing bied ’n baie goeie balans tussen benodigde hardeware kompleksiteit en koderingsprestasie. Daar is gevind dat heelwat parameters van die dekodeerders en kodes aangepas kan word, ten einde implementering in stelsels met ’n wye verskeidenheid van geheuespasie en verwerkingsvermoëns moontlik te maak.

Acknowledgements

I would like to express the deepest appreciation to my excellent study leader Dr. Riaan Wolhuter. Thank you for all your wisdom and enthusiasm in my studies from the first day to the very final moments.

I would like to thank Gerda for her love, kindness and understanding. My mother, Marthie Olivier, and brothers Carel and George for their help with proofreading. My dad, Carel Olivier, for his wisdom and financial support, and also my brother Ockie, for being a great role model. Thank you all for your love and support during my studies.

I would like to thank all my friends in the DSP lab, especially John Gilmore and Shaun Lodder, for all your support and motivation stretching into long hours of many days, and random conversations over coffee. My colleagues and friends who worked with me on the project, especially Ewald van der Westhuizen and Kobus Botha. To Charlene Weimers, for all the uplifting conversations and help. Gert-Jan van Rooyen, for all his support and interesting discussions.

I would also like to thank Professor Holger Carl, for his kindness and inputs during my visit at the Georg Simon Ohm University of Applied Sciences Nuremberg.

D.J.C. MacKay for providing the parity check matrices used in this thesis, and all the useful resources he made available.

Financial support from the Oppenheimer Memorial Trust, Stellenbosch international office and the COE.

Finally and most importantly, I would like to exalt my Creator for everything He has given me.

Dedications

to my ouma Joey, for all her love, patience, inspiration, and the joy she brought

Contents

List of Figures

List of Tables

List of Abbreviations

ACM	Adaptive Coding and Modulation
ARQ	Automatic Repeat-Request
ASM	Attached Synchronisation Market
AWGN	Additive White Gaussian Noise
BER	Bit Error Ratio
BCH	Bose Chaudhuri Hocquenghem
BP	Belief Propagation
BPSK	Binary Phase Shift Keying
CAN	Controller-Area Network
CADU	Channel Access Data Unit
CCSDS	Consultative Committee for Space Data Systems
CD	Compact Disk
CRC	Cyclic Redundancy Check
DSP	Digital Signal Processing
ECSS	European Cooperation for Space Standardization
ESA	European Space Agency
FEC	Forward Error Correction
FER	Frame Error Rate
FPGA	Field Programmable Gate Array
GF	Galois Field
ID	Identifier
ISE	Integrated Software Environment
IS-HS	In-Situ-Hyperspectral
LDPC	Low Density Parity Check
LEO	Low Earth Orbit

LLR	Log Likelihood Ratio
MDS	Maximum Distance Separable
ML	Maximum Likelihood
NASA	National Aeronautics and Space Administration
OBC	On-Board Computer
OSI	Open Systems Interconnection
QC	Quasi-Cyclic
RAM	Random Access Memory
RF	Radio Frequency
RS	Reed-Solomon
RTL	Register Transfer Level
SCID	Spacecraft Identifier
SECDED	Single Error Correction Double Error Detection
SEL	Single Event Latchup
Si	Silicon
SNR	Signal-to-Noise Ratio
TC	Telecommand
TFVN	Transfer Frame Version Number
TID	Total Ionising Dose
TM	Telemetry
VHDL	VHSIC Hardware Description Language
VHSIC	Very High Speed Integrated Circuit
XOR	Exclusive-OR

Nomenclature

Greek Letters:

α	Normalisation factor
β	Offset
λ_n	Log likelihood ratio
σ	Variance

Matrices and Vectors:

$\mathbf{G}$	Generator matrix
$\mathbf{H}$	Parity check matrix
$\mathbf{I}$	Identity matrix
$\mathbf{L}$	Probability matrix of horizontal step
m	Message vector
p	Check node vector
r	Redundancy vector
x	Codeword vector
$\widehat{x}$	Estimated codeword vector
z	Syndrome vector
$\mathbf{Z}$	Probability matrix of vertical step

Subscripts and Superscripts:

m	Column in matrix
n	Row in matrix
T	Transpose

Units:

bit/s	bits per second
dB	Decibel
Hz	Hertz
km	Kilometer
kg	Kilogram

Variables:

a	Amplitude
C	Capacity
E_b	Bit energy
E_L	Loss of bit energy
E_x	Coded bit energy
f_b	Baseband frequency
f_m	Modulated frequency
f_x	Encoded frequency
j	Column weight
k	Row weight
K	Information length
L_c	Channel reliability
M	Redundancy length

M_n	Set of checks in column n
N	Code length
$n(t)$	Noise signal
N_m	Set of bits that participate in check p_m
N_o	Noise spectral density
P	Average power
p_n^0	Probability from demodulator that variable is 0
q_{mn}^0	Message passing probability that variable is 0
R	Information rate
$r(t)$	Received signal
s	Cyclical shift
$s(t)$	Modulated transmitted signal
W	Bandwidth
w_c	Column weight
w_r	Row weight

Chapter 1

Introduction

1.1 Background

During February in 1999 South Africa's first scientific satellite was launched. SunSat was a micro satellite built by post-graduate engineering students at Stellenbosch University. SunSat operated successfully in space for two years, reaching all its mission objectives. A second satellite, SumbandilaSat, was commissioned and built in collaboration between Stellenbosch University and SunSpace. SumbandilaSat was launched on 17 September 2009. These satellites had similar error control strategies, implementing error detection without error correction.

A third satellite project, known as IS-HS II, is currently being developed in collaboration between Stellenbosch University and the Faculty of Bioscience Engineering at the Katholieke Universiteit Leuven. An innovative approach has been proposed to gather data "within the field". The idea is to forward in-situ measured data via satellite to a remote central server. These measurements include humidity, temperature and other environmental parameters. Measurements can be taken in a vineyard or forest and forwarded via a low earth orbit (LEO) satellite. At the same time the LEO satellite observes this area using earth observation techniques. This combined simultaneous information gathering and transmission holds good potential for agricultural research. In this thesis a modem link strategy is designed to improve the link budget of the new generation of satellites designed by Stellenbosch University. This modem link strategy includes a low density parity check (LDPC) code as a forward error correction code.

LDPC codes, also known as Gallager codes, was discovered by Gallager in 1962 [1]. For more than thirty years these codes were largely ignored, due to the lack of simple decoding techniques. MacKay rediscovered these codes in 1995 and used iterative decoding to show large channel coding gains with an efficient iterative decoding algorithm [2]. In the last few years, LDPC codes received a lot of attention, and error performances approaching the Shannon limit were demonstrated [3]-[4]. This thesis investigates the possibility of applying LDPC coding strategies to the IS-HS II project and other similar projects.

1.2 Motivation

LEO satellite communication presents a unique environment which inherently differs from most other communication channels. It has characteristics such as limited time windows, drastically varying link margins and varying geographic ground station distribution. Most protocols were designed for wired connection orientated networks. These protocols perform poorly when used on satellite channels with significant loss of packets [5], mostly due to noise introduced on the satellite link. Most of these error control strategies mainly implement error detection schemes. The absence of forward error correction codes over satellite links results in unreliable communication. This decreases the throughput to give unacceptably low channel utilisation results. Packet loss due to a link error wrongly interpreted by the protocol as a congestion related error, further decreases the performance of satellite communication. Through the implementation of forward error correction, large coding gains can be realised over the communication link.

Application specific design is required to optimise the throughput of the communication system. In this thesis a protocol strategy implementing a mission specific forward error correction (FEC) code is proposed. A protocol strategy is designed for the IS-HS II project to facilitate the implementation of a mission specific error control strategy, consisting of error detection and correction.

With power limitations on the space segment and remote ground stations, hardware utilisation efficiency need to be optimised to reduce power consumption. The belief propagation (BP) iterative decoder can be implemented with relatively low complexity to give error performances comparable to a maximum likelihood (ML) decoder [6]. Some approximations can be made in the iterative steps of the BP decoder to further decrease its complexity, while maintaining acceptable error performances [7]. However, it is important to realise that this type of approach has much wider application possibilities, particularly in terrestrial wireless mobile communications in fringe areas.

1.3 Research objectives

The main objective of this thesis is to design a modem link strategy to optimise the volumetric data throughput of an LEO satellite. Within the modem link strategy, a protocol strategy and error control strategy need to be designed. The protocol strategy needs to meet the following requirements:

- Allow the implementation of a mission specific error control strategy.

- Interdependence between layers of the open systems interconnection (OSI) model. This allows different layers of the protocol strategy to be implemented on various hardware devices, with hardware interfaces between layers.

The error control strategy involves error detection and error correction. Forward error correction schemes need to be investigated, and an FEC code must be chosen that meets the following requirements:

- Good error performance for variable link margins.

- Implementation needs to optimise hardware utilisation efficiency, to reduce power consumption on the space segment.

The whole communication system has to be implementable with limited resources on the on-board computer (OBC) of the space segment, and additional space qualified processors.

1.4 Contributions and summary of results

The following are the main contributions of the thesis:

- A protocol strategy was designed to facilitate a mission specific error control code.

- A packet processing scheme was designed to avoid the necessity of packet inspection of higher layer protocols in the data link layer.

- Forward error correction schemes were investigated for application in LEO satellite systems.

- It was shown that LDPC codes provide good error performance with low processing requirements.

- Different variants of the belief propagation decoding algorithm were explained and implemented in MATLAB.

- Min-sum approximation algorithms for the codes were implemented in MATLAB.

- Simulation results were presented that compare the performances and characteristics of the different decoding algorithms.

- A synthesizable MATLAB script was written in AccelDSP and VHSIC hardware description language (VHDL) code was generated for an LDPC encoder and decoder to be implemented on a Xilinx field programmable gate array (FPGA).

It was proved that the performances of the BP and min-sum decoders show significant coding gains over uncoded systems. It is found that for block lengths larger than approximately 200, LDPC codes outperform other popular coding schemes such as convolutional codes and Reed-Solomon codes. Using simplified approximations for the BP decoder, and with the appropriate selection of algorithm parameters, the normalised min-sum approximation decoder shows similar performance to the BP decoder. The normalised min-sum decoder shows performances comparable to that of the BP decoder with less associated processing complexity. The expected losses associated with hardware implementations are evaluated and provided. These include soft decision quantisation and fixed point quantisation errors. It is proved that the error performance of the fixed point implementation is good, while its processing requirements are within acceptable margins.

The generation of hardware description language (HDL) code in AccelDSP for design functions allows for a shorter design cycle from simulation to the final hardware implementation. Functionality of the system can be tested without the need to design HDL testbenches for all modules. The IS-HS II satellite project is still in a conceptual phase of development. This implementation is part of a structured process to practically implement short length LDPC codes in FPGA hardware for actual flight application. The proposed implementation of LDPC coding holds much promise to improve the volumetric data throughput of general communication systems with variable and relatively low quality communication links.

1.5 Outline of thesis

In this section, an overview of the main topics considered in the remaining chapters is presented. In Chapter 2 a theoretical framework of forward error correction (FEC) is presented, while focussing on LDPC codes. The main concepts of LEO satellite communication are explained, and a background of the LEO satellite project is given. The main findings are presented after conducting a literature study of various FEC codes.

The design of different LDPC decoders are shown in Chapter 3. Detailed descriptions of the BP and min-sum approximation algorithms are given. Two strategies to improve the performance of the standard min-sum decoder are introduced. A protocol strategy is presented to enable the implementation of a mission specific error control strategy.

In Chapter 4 the performance results of the LDPC decoders are given. It is shown that the BP and normalised min-sum decoders can realise large coding gains, comparing well to other codes, even for short block lengths. The effect of various parameters related to hardware implementations on the performance of the codes, are investigated. An implementation of the normalised min-sum decoder and an encoder for an LDPC code is presented. This is done in AccelDSP, to generate register transfer level (RTL) code for FPGA implementation.

Finally in Chapter 5 the research findings are discussed and summarised. The thesis is concluded by referring to future work.

Chapter 2

Literature review and theoretical framework

2.1 Introduction

LEO satellite communication provides a unique set of challenges with small communication time windows, varying link margins and power constraints. In this chapter an introduction to LEO satellite communication is given. Key concepts are explained which is essential in the understanding of the implementation of an error control strategy for satellite communication. A literature review is presented for many of the coding schemes. Concepts of linear block codes will be explained in detail. LDPC codes are introduced as an error control strategy for implementation in LEO satellite communication systems. They provide excellent performance by using an iterative message passing decoder on sparse graph codes. A history and literature review of LDPC codes are given, and some of the basic concepts of LDPC codes are introduced.

The IS-HS II is an LEO micro satellite project. The background of the IS-HS II project is given and concepts are introduced for the design of a protocol strategy for this project. The system architecture of the space segment, and design goals of the ground stations are presented. These concepts will be used in Chapter 3 to design the protocol and error control strategies.

The chapter is concluded with a discussion of the literature covered, and conclusions are made concerning the implementation of error control strategies for LEO satellite applications.

2.2 Key concepts

2.2.1 LEO satellite communication link

LEO satellite communication creates an environment in which the communication time is very limited. The polar orbit of an LEO satellite is described in [8]. Sun-synchronous LEO satellites have the property of passing a ground station each day within two time windows. One of these time windows will be during daytime, and one during the night, twelve hours apart. Each of these time windows, approximately 3 hours long, will consist of two or three passes within the footprint of the satellite, where communication can be realised. These satellites have the property that they pass over every region of the earth. About 52 passes can be expected in a two week period, after which the satellite will pass over the same point. Therefore, a two week simulation will give an accurate simulation of the satellite orbit. When designing an LEO satellite communication link, the following properties need to be considered, namely:

1. The communication time is very limited and statistically, the probability of a direct overhead pass is small. This means that most passes have a low elevation angle, with high propagation losses and low antenna gain, resulting in a poor link budget.

2. LEO satellites have circular orbits and therefore, constant altitudes between 600-1400km [9]. This implies large propagation delays in communication.

3. These micro satellites are relatively small, and with area and power constraints the antenna gains on the space segments are usually small. A major factor in cost reduction on the ground segment is to avoid satellite tracking and steerable antennas. All these factors add to a very poor link budget.

The implementation of forward error correction to improve the link quality as proposed in this thesis, can be applied to other communication scenarios with similar properties. Providing telecommunication services to rural areas and regions on the borders of terrestrial wireless areas can also be improved or realised through the implementation of the proposed error control strategies.

2.2.2 IS-HS II satellite project

The IS-HS II satellite project is an LEO micro satellite designed in cooperation between the Engineering Faculty of Stellenbosch University and the Bioscience Engineering Faculty at the Katholieke Universiteit Leuven. In this thesis, a protocol strategy is presented for this project. Design and hardware considerations were made for implementation on the IS-HS II. This satellite will be similar to its predecessor, SumbandilaSat (ZASAT-002), an 80kg LEO satellite expected to orbit at an altitude of approximately 600km. A short description is given of the satellite space segment and ground stations.

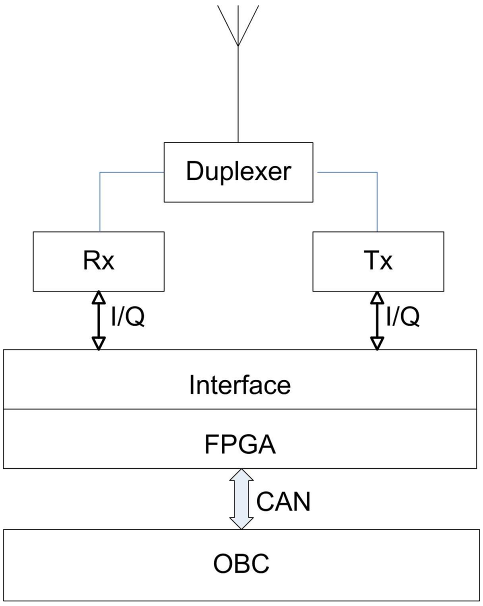


Figure 2.1: Architecture of space segment

Space segment

The IS-HS II project is an LEO satellite project implementing beam steering on the space segment [10]. The satellite will do on-board processing. The architecture of the space segment of the IS-HS II project can be seen in Figure 2.1. The in-phase/quadrature-phase (I/Q) signals between the interface and the transmitter and receiver will be at the radio frequency (RF). The interface will consist of all the analog devices to downconvert the signal onto baseband frequency and to do analog to digital conversion. This baseband frequency will be slightly higher than the effective data rate. This is due to the channel coding, as will be discussed in Section 2.2.3. Modulation and demodulation can be implemented on the FPGA, or on an external modem. When an additional modem is used for modulation and demodulation, it will form part of the interface block, as shown in Figure 2.1. A concept design of the data link layer implementation on the FPGA will be presented in this thesis. A controller-area network (CAN) [11] bus will be used as the interface between the FPGA and the OBC. The OBC will use a QNX operating system [12] and be responsible for networking on the transportation and application layers, as discussed in Section 2.2.7, as well as satellite systems information and application specific information for possible implementation of beam forming. Implementing FEC codes in the OBC might be considered as a possibility, but generally very limited resources are available on the OBC. Designing an OBC to be space qualified, the increase in memory and processing speed also reduces the reliability of the components. For this reason, the implementation of FEC codes on a space qualified FPGA is preferred.

Ground stations

The purpose of the IS-HS II project, inter alia, is to build low cost ground stations for developing countries. This data will be used in research to aid agricultural studies. One of the most important factors in building low cost ground stations is to avoid the use of satellite tracking antennas. This leads to a lower gain antenna and a smaller link margin. Another important requirement for the ground stations is self sustainable energy. Using solar panels implies that a limited power budget will be available, compounding the problem.

2.2.3 Error control coding

Error control coding forms part of the field of information theory, a branch of applied mathematics. The field of information theory had its origins in the paper by Claude Shannon in 1948 entitled *"A Mathematical Theory of communication"* [13]. By adding controlled redundancy to transmitted information, errors introduced on the channel can be detected and corrected in the receiver [14]. Prior to this discovery, it was generally understood that error-free communication cannot be realised, due to noise introduced on the communication channel. Shannon proved that the reliability of communication is not only dependent on the noise, but reliable communication can be obtained within the capacity of a channel, by adding an appropriate channel code. The channel capacity can be defined as the maximum mutual information between the input and the output. This capacity (C), defines the rate (R), at which reliable communication with an arbitrary small error probability can be obtained.

Channel capacity

Shannon's channel coding theorem gives the theoretical bounds within which reliable communication can be realised. As discussed in [15], the channel coding theorem can be formulated as:

Theorem 1 *Provided that the coded rate of transmission R is less than the channel capacity C, for any given probability of error ϵ specified, there is an error correction code of length n_0 such that there exist codes of length N exceeding n_0 for which the decoded probability of error is less than ϵ.*

From this theorem it is clear that the Shannon limit gives a rate R, in bits per second, at which reliable communication can be achieved. The concept of the code rate and specific codes will be discussed in the next section. Looking at a practical example as highlighted in [16], with additive white Gaussian noise, interference on an ideal band-limited channel of bandwidth W has a capacity C given by

$$C = W \log_2 \left(1 + \frac{P}{WN_0}\right) \quad \text{bits/s} \tag{2.2.1}$$

where P is the average transmitted power and N_o is the power spectral density of the additive noise. When you consider this example, the channel coding theorem can be explained as follows. If you have an information rate R from any source on this channel which is smaller than C as specified in Equation 2.2.1, it will be theoretically possible to achieve error-free transmission through appropriate coding. C is the limit for error-free transmission over this channel. If you have a source with an R larger than C, error-free transmission will not be possible, regardless of the implementation. For this reason, no modulation schemes and coding methods will do better than the theoretical limit given by Shannon. This limit can be used when designing, and to determine the efficiency of an implementation. In satellite systems it should be noted that the systems are generally power limited systems, and not band limited systems. It should be noted that Theorem 1 gives the most comprehensible explanation of Shannon's capacity theorem in the context of this thesis.

Forward error correction

Forward error correction is implemented by adding redundancy to a message and using that information at the receiver to correct errors introduced by the channel. When evaluating a wireless channel for noise characteristics, the bit error ratio (BER) is used as a metric to determine the reliability of communication.

The rate R of an error correction code is equal to the information bits K divided by the code length N. We have $R = \frac{K}{N}$, where the redundancy M in the code can be calculated as $M = N - K$. Throughout this thesis this will be referred to as an (N, K) code.

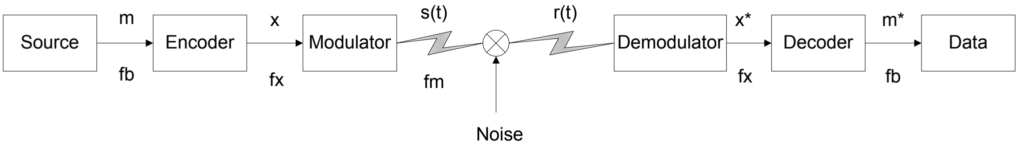


Figure 2.2: Block diagram of a communication channel using forward error correction.

A few concepts need to be defined to explain the measure in which we determine the performance of error correcting codes. Coding gain is a measure used to compare the signal-to-noise ratio (SNR) required by the transmitter of a coded system to that required by an uncoded system to obtain a specified BER. Coding gain is given as a decibel (dB) gain or loss. In Figure 2.2 a communication system implementing forward error correction is shown. It can be seen that f_b is the baseband frequency from the source. In this figure, an uncoded system can be viewed as a system without the encoder and decoder, thus the baseband frequency f_b will be sent to the modulator to be transmitted on the modulated signal f_m. This represents the end to end data rate of the communication system. For the coded system shown, the baseband data rate f_b will be changed to a higher rate f_x after encoding, where

$$f_x = f_b \times R^{-1}. \tag{2.2.2}$$

To ensure that the same amount of power is used for the coded and uncoded system, the energy per bit E_b will be smaller for the coded system. The energy per bit for the coded system will be denoted E_x, and calculated as $E_x = R * E_b$. This means that we have a loss of energy per bit of

$$E_L = 10 \log_2 R \qquad (dB). \tag{2.2.3}$$

The addition of redundancy in a coded system reduces the net energy of each bit averaged over the message, but the capabilities of correcting errors effectively adds a coding gain. It is obvious that adding redundancy to a system in which no errors occur on the link will reduce the effective throughput, but in a system with high error probabilities FEC can enable reliable communication. Calculating the difference in required power needed by the antenna to obtain the same BER will show a coding gain or loss.

It is desirable to design codes where the capability of the codes to correct corrupted errors at the receiver reduces the total power required by the antenna to achieve a specified BER. The performance of coded and uncoded systems with different code rates can be compared by using the E_b/N_o metric. A detailed discussion, and calculation of the E_b/N_o metric is given in Section 3.5.

It should be noted that when a single bit is corrupted it will effectively corrupt a whole frame, and therefore the frame error rate (FER) is an important measure when evaluating the overall reliability of a specific communication system. In systems where memory is limited, a corrupted frame will effectively stop the transmission until the error is corrected through retransmission, which implies major loss of throughput on channels with large delays such as satellite links.

2.2.4 Linear block codes

Block codes are fixed length channel codes. Each block is N bits long, representing K information bits, and M redundant bits. Linear block codes are a special class of block codes with linear dependencies between codewords. Linear block codes can be described by their parity check matrix $\mathbf{H}$, and generator matrix $\mathbf{G}$. Note that only binary codes will be considered throughout this thesis. An (N,K) linear block code is a code with 2^K codewords as defined in [15]. For linear block codes, the sum of any two codewords will produce another codeword. There exists an one to one mapping between a message m and a codeword x.

A linear block code needs to satisfy the condition that all linear combinations of the rows of **G** must produce a codeword x, and the following equation must be satisfied

$$\mathbf{G} \times \mathbf{H^T} = 0. \tag{2.2.4}$$

Any codeword x multiplied by the parity check matrix must give an all-zero vector, thus $x \times H^T = 0$.

Graphical representation of linear block codes

Graphical representations of linear block codes will be explained by referring to the (7,4) Hamming code. A message of four bits $[m_1\ m_2\ m_3\ m_4]$ is encoded to give a codeword $[m_1\ m_2\ m_3\ m_4\ r_5\ r_6\ r_7]$, where the bits denoted as r represent the redundant bits, added by the encoder. This is a single error correction code, and a graphical representation is given in Figure 2.3.

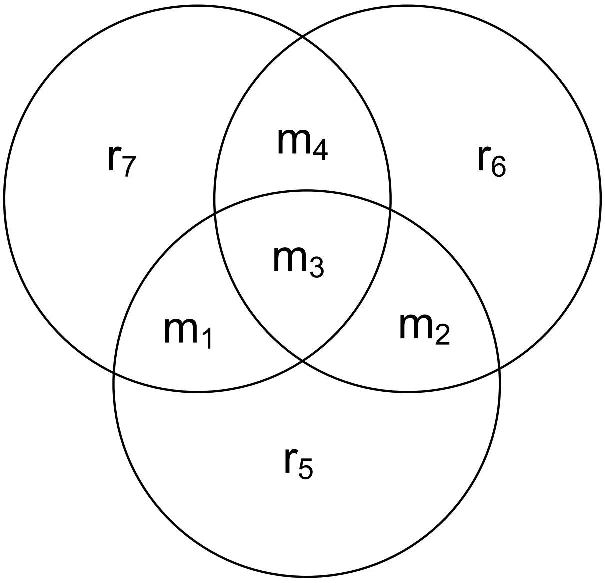


Figure 2.3: Graphical representation of (7,4) Hamming code showing the dependencies between message bits and redundant bits.

In Figure 2.3 the bits and dependencies are shown as three circles. The redundant bits $[r_5, r_6, r_7]$ are calculated to ensure that the parity of each circle is even. At the receiver a single error can be corrected. Any bit flipped by the noisy channel will ensure that one, two or all three of the circles will have uneven parities. At the receiver we calculate which single bit should be flipped back to ensure that all the circles have even parities. If a single error occurred, this method will correct a single erroneous bit. Looking at Figure 2.3 we can see that in order to change the parity of one circle, we need to flip the redundant bit in the associated circle, therefore r_5, r_6 or r_7. In the case that we have two circles with uneven parities, we flip the bit which influences both circles, therefore m_1, m_2 or m_4. When all circles have uneven parities at the receiver, we flip bit m_3.

In Figure 2.4, a graphical representation of the parity check matrix of the $(7, 4)$ Hamming code is shown. This graph is known as a Tanner graph. A Tanner graph gives a graphical representation of a linear block code. The Tanner graph is a bipartite graph, describing the parity check matrix $\mathbf{H}$ of a linear block code by using two sets [15]. The first set contains the variable nodes (represented by circles in Figure 2.4), where each node gives a description of a column in $\mathbf{H}$. The second set contains the check nodes (represented by rectangles in Figure 2.4), where each node gives a description of a row in $\mathbf{H}$. Every connection in the graph is known as an edge. An edge can only be a connection between the different sets in a graph for bipartite graphs. Whenever there is a $'1'$ in $\mathbf{H}$, it produces an edge in the graph. If $\mathbf{H}_{m,n} = 1$, there is an edge between the m^{th} bit node and the n^{th} check node.

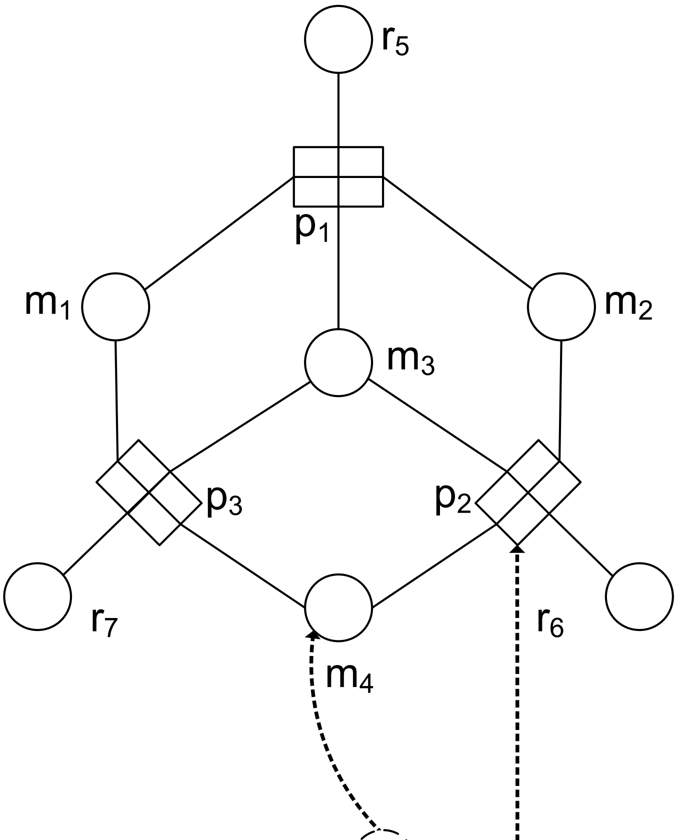
Throughout this thesis, the notation $\mathbf{H}_{m,n}$ will refer to the element in the m^{th} row and n^{th} column of $\mathbf{H}$. $\mathbf{H}_{3,n}$ will refer to the 3^{rd} row of $\mathbf{H}$, and $\mathbf{H}_{m,3}$ will refer to the 3^{rd} column of $\mathbf{H}$.

An (N, K) code has a constraint length of $M = N - K$, and produces M parity checks for the code. The $(7, 4)$ Hamming code example in Figure 2.4 has $M = 3$ constraints, and therefore we have three check nodes in the graph and three rows in its parity check matrix $\mathbf{H}$. Each row in $\mathbf{H}$ gives a description of a check node. In Figure 2.4 the check node p_2 is highlighted, the second row of $\mathbf{H}$ gives a description of p_2. It is shown that p_2 is connected to each bit node, for which it has $\mathbf{H}_{2,n} = 1$. It can be verified in the graph and parity check matrix that p_2 is connected to $[m_2\ m_3\ m_4\ r_6]$.

A variable node is described by a column in $\mathbf{H}$. In Figure 2.4 m_4 is highlighted. It is shown that $\mathbf{H}_{m,4}$ gives a complete description of the variable node m_4, and can be verified in the graphical representation. In $\mathbf{H}_{m,4}$ the elements $\mathbf{H}_{2,4} = 1$ and $\mathbf{H}_{3,4} = 1$, thus m_4 is connected to $[p_2, p_3]$.

An edge is a connection between a variable node and a check node. In Figure 2.4 $\mathbf{H}_{2,4} = 1$, this represents an edge between the variable node m_4 and a check node p_2.

Each check node forces the sum of all connected bit nodes to be even. It should be noted that the only bits sent over the communication link, are the variable nodes. The encoder enforces the constraints required by the check nodes or parity checks by the addition of the redundant bits. This is shown



$$H = \begin{array}{c}\left[\begin{array}{ccccccc} 1 & 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 1 & 0 & 0 & 1 \end{array}\right] \begin{array}{l} \Leftarrow p_1 \\ \Leftarrow p_2 \\ \Leftarrow p_3 \end{array} \\ \begin{array}{ccccccc} \Uparrow & \Uparrow & \Uparrow & \Uparrow & \Uparrow & \Uparrow & \Uparrow \\ m_1 & m_2 & m_3 & m_4 & r_5 & r_6 & r_7 \end{array}\end{array}$$

Figure 2.4: Graphical representation of the parity check matrix.

by an example. To calculate a codeword x for the $(7,4)$ Hamming code, the message m is multiplied by the generator matrix $\mathbf{G}$. In Equation 2.2.5 the codeword x is calculated for the message $m = [1\ 0\ 0\ 1]$.

$$x = [1\ 0\ 0\ 1] \times \begin{pmatrix} 1 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 & 1 & 1 \end{pmatrix} \tag{2.2.5}$$

where $x = m \times \mathbf{G}$, $\mathbf{G}$ represents the generator matrix for the (7,4) Hamming code, and $x = [1\ 0\ 0\ 1\ 1\ 1\ 0]$. The encoder determines a linear combination of every row in $\mathbf{G}$, where the value of the message is equal to one. Equation 2.2.6 verifies this statement by showing the calculated codeword x as the result of Galois field GF(2) addition of the first and fourth rows in $\mathbf{G}$.

$$\begin{aligned} x &= [1\ 0\ 0\ 0\ 1\ 0\ 1] + [0\ 0\ 0\ 1\ 0\ 1\ 1] \\ x &= [1\ 0\ 0\ 1\ 1\ 1\ 0] \end{aligned} \tag{2.2.6}$$

For larger codes, the Tanner graph is reordered to group the variable and check nodes together. The Tanner graph of the (7,4) Hamming code shown in Figure 2.4 is reordered into the standard representation and shown in Figure 2.5.

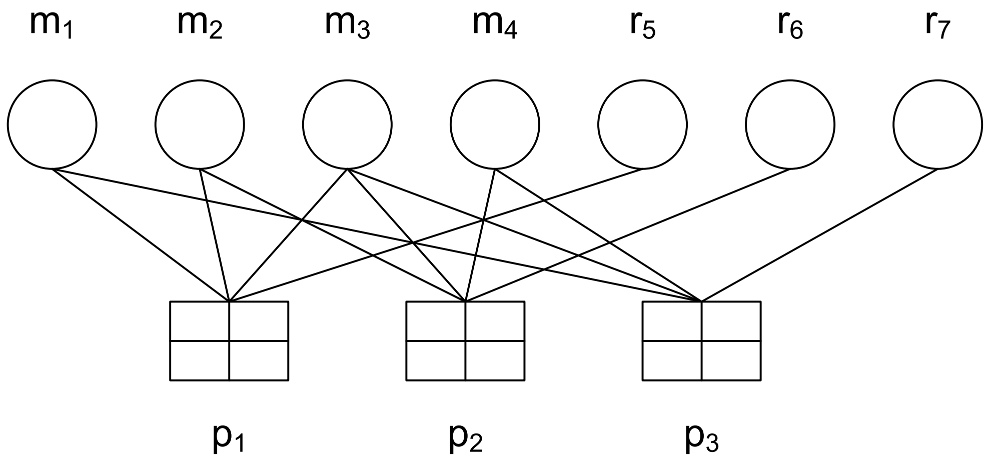


Figure 2.5: Tanner graph of the (7,4) Hamming code.

2.2.5 Coding schemes

2.2.5.1 Hamming codes

The first major error correcting code was discovered by Richard Hamming in 1950. Hamming collaborated with Shannon at Bell Telephone Laboratories, and because of his contributions to practical correction codes, he is considered by many as the founder of the subject of error control coding [14]. The Hamming code is capable of correcting all single errors, or detecting any combination of one or two errors [17]. Hamming codes can be implemented efficiently by using syndrome decoding. Each non-zero syndrome z, corresponds to a digital signature to identify the position of the erroneous bit in the codeword. A Hamming code can be expressed as an (N, K) code, where $N = 2^a - 1$, and $K = 2^a - a - 1$. These conditions hold for all values of a single integer a, where $a \geq 2$. Examples are the most basic (7,4) code, as discussed in Section 2.2.4, and the (31,26) code.

Extended Hamming codes are widely used in RAM implementations. By adding an additional parity check bit to a Hamming code, a single error correction double error detection (SECDED) code is created. An example of such a code is the (8,4) code.

2.2.5.2 BCH codes

Bose Chaudhuri Hocquenghem (BCH) codes are cyclic block codes which can have a binary or non-binary alphabet. BCH codes can be used to give a variety of codes rates, with good error performance. These codes increase significantly in computational complexity as the code lengths increase. The code length of a BCH code is $N = 2^a - 1$, where $a \geq 3$. In Table 2.1 an example is given of the error correcting capabilities of a binary BCH code, where $a = 6$.

Table 2.1: Error correcting capability t, for an (N,K) BCH code with $N = 63$

K	t	R
57	1	0.90
51	2	0.81
45	3	0.71
39	4	0.62
36	5	0.57
30	6	0.48
24	7	0.38
18	10	0.29
16	11	0.25
10	13	0.16
7	15	0.11

In Table 2.1 the block length $N = 63$, K represents the number of information bits, t represents the number of correctable errors and R represents the code rate. From this table it can be observed that BCH codes can be used for a variety of code rates, offering good error correcting capabilities. As discussed in [17], BCH codes performs best for code rates where $0.33 < R < 0.75$. BCH codes are popular codes, and included in many satellite and space standards, such as the telecommand (TC) protocols by the European Cooperation for Space Standardisation (ECSS) [18].

2.2.5.3 Reed-Muller codes

Reed-Muller codes followed the Hamming code in the early fifties and are able to correct a multiple of errors in a single code word. Even though this code was mostly replaced by BCH codes in the late sixties due to the improved error performance of the BCH code. Reed-Muller codes have an extremely fast maximum likelihood decoding algorithm which is superior to the decoding complexity of BCH codes. Reed-Muller codes were used on the *Mariner* deep space probes in 1969.

2.2.5.4 Reed-Solomon codes

Reed-Solomon codes were discovered in 1960, by I. Reed and G. Solomon [19]. Even though these codes were discovered before BCH codes, it can be considered as a special class of non-binary BCH code. The decoders of Reed-Solomon and BCH codes are similar. A Reed-Solomon code can be seen as a $q^m -$ ary BCH code with a code length of $q^m - 1$. Reed-Solomon codes are extremely good in correcting burst errors. These codes are maximum distance separable (MDS) codes. MDS codes have unique properties that are useful in shortening and puncturing codes. In a shortened code codewords are omitted from the code by decreasing K and N in an (N, K) code, and maintaining the redundancy M. In a punctured code redundancy bits are omitted in the code, reducing N and M for an (N, K) code. These modified codes are useful for the implementation in a variety of applications [14]. Reed-Solomon codes are widely used in storage, such as the compact disk (CD), and are used in concatenated codes, which find many applications in space and satellite communications.

2.2.5.5 Convolutional codes

Convolutional codes are codes which are generated by passing an information sequence through a linear-state shift register. They differ from block codes, where a bit stream can be shifted into a convolutional encoder, and seen as a continuous stream at the output. Convolutional codes were discovered in 1955 by Elias [20]. In 1979 Viterbi discovered his algorithm for decoding convolutional codes, and showed that it is an ML decoder for convolutional codes [21]. For a linear convolutional encoder, the N-bit output sequence is a linear combination of the K-bit input sequence, plus a linear combination of the previous $k-1$ bits, where k represents the constraint length of the code. The code rate is $R = K/N$. In convolutional codes, N and K does not represent the code length of a codeword. Convolutional codes are usually described as a tree diagram, a trellis diagram and a state diagram. Convolutional codes have extremely simple encoders requiring little computational overhead. The Viterbi decoding algorithm is very elegant, and the decoding complexity is not a function of the length of the codeword. The algorithm uses the constraints introduced in the encoding to eliminate paths which cannot lead to an ML decision. At each state, or for each new bit entering the algorithm, the decoder calculates the next most likely path. This ensures that unlikely paths are eliminated early in the algorithm, reducing the complexity. Choosing the most likely path is similar to choosing the most probable codeword. Convolutional codes are used extensively in satellite and space communications.

2.2.5.6 Concatenated codes

Concatenated codes are created by using more than one error control code in a communication system. When two codes are used in a concatenated code, the data from the first encoder is used as the input to the second encoder, creating a sequential system as shown in Figure 2.6. The information bits are encoded with the Reed-Solomon code, or outer code. This encoded Reed-Solomon data is used as input for the convolutional encoder, or inner code. The output of the convolutional encoder is modulated and sent over the noisy satellite channel. The received signal is demodulated and decoded by the convolutional decoder, or inner code. This decoded data is sent to the Reed-Solomon decoder, and lastly decoded by the outer code.

The concatenated Reed-Solomon and convolutional code is a powerful and popular scheme, and used in standards by the National Aeronautics and Space Administration (NASA) and the European Space Agency (ESA). Usually, a symbol interleaver is used between the inner and outer codes. Interleavers increase the performance of the concatenated codes. Examples include NASA's *Galileo* mission to Jupiter with a block interleaver that holds $n = 2$ Reed-Solomon code words, and ESA's *Giotto* mission to Halley's comet with an interleaver that holds $n = 8$ Reed-Solomon codewords [14]. This standard is a half rate convolutional code with constraint length of 7, combined with a (255,223) Reed-Solomon code [22].

The advantage of using a concatenated scheme, is that two relatively low complexity (small size) codes can be combined to provide a system with good error performance. The convolutional codes are good codes in applications where moderate reliability is required on poor channels. Reed-Solomon codes are strong codes for providing high reliability on channels which are moderately noisy. Reed-Solomon codes also performs well when correcting burst

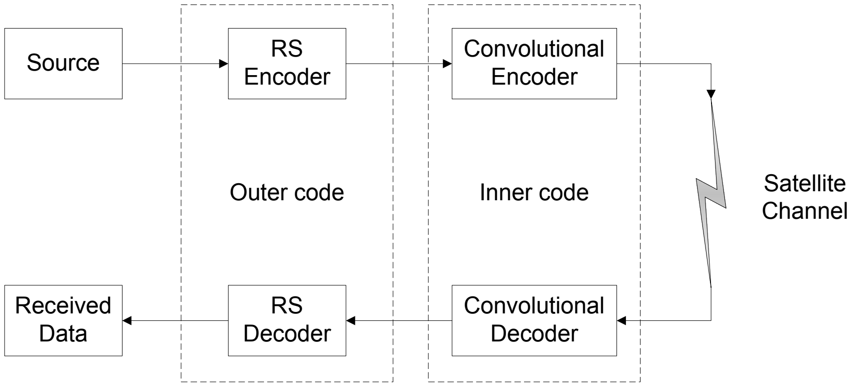


Figure 2.6: Concatenated code with a Reed-Solomon code as outer code, and a convolutional code as inner code.

errors. Combining these two codes results in a strong code as the two codes compliment each other by combining their strengths, giving a concatenated code performing well in channels with poor reliability and in situations of moderately noisy conditions. Another advantage is that the Viterbi algorithm occasionally gives out a burst error when choosing the wrong state, while the outer Reed-Solomon code is extremely good at correcting these burst errors.

2.2.5.7 Turbo codes

Turbo codes are block codes with large block sizes N. Turbo codes were discovered in 1993 by Berrou, Glaviux and Tshitimajshima [23]. The performance of block codes increase as the block size increase. The problem is that the decoder usually increases exponentially in complexity as the block length increases. Turbo codes, also known as parallel concatenated codes, allow for practical implementable block codes with large block sizes. The encoder uses multiple (usually two) systematic block codes working on the same data set by implementing interleavers. The most common encoder implementation consists of two recursive convolutional encoders. These encoders have a low complexity to realise in hardware and can be implemented by using simple logical operations, similar to that used in convolutional codes. The functionality of a Turbo code will be explained by referring to Figure 2.7.

It can be seen that the input message m is fed into the two convolutional encoders and also sent without manipulation over the wireless link. In this example a 1/3 rate code is realised. Note that a half rate code can be created by puncturing the outputs of the decoders to alternate between $p1$ and $p2$. For each bit m, two parity bits $p1$ and $p2$ are calculated by $Encoder1$ and $Encoder2$. Note that there is an interleaver between the input message m

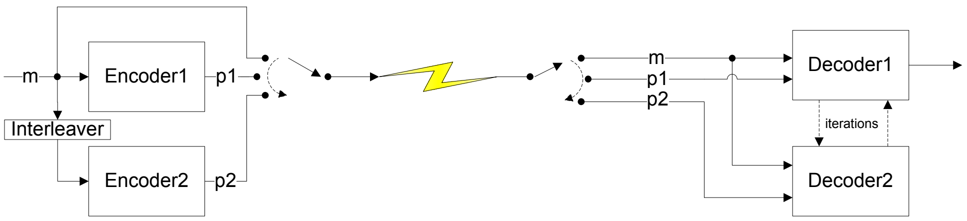


Figure 2.7: Turbo code implementation.

and *Encoder2*. The performance of the code depends heavily on the design of the interleaver. The interleaver implements a one-to-one mapping of bits in the message to the interleaved message used by *Encoder2*. This means that they work on the same set of data, or data block. The encoded message is sent over the wireless link. At the receiver the message bits are given to both decoders, while the parities are only given to the relevant decoders. Both of these decoders can be implemented with low complexity. They are both soft output decoders. The implementation of local message passing between these two decoders enables the implementation of large block codes, providing good error performances. The two decoders take turns to work on the decoding by producing their results as extrinsic information (output), to be used by the other decoder as prior information (input). Iterating two relatively simple decoding techniques, produces a strong code. Usually decoding continues for a fixed number of iterations, after which the output is given by either *Decoder1* or *Decoder2*. The interleaver is also present in message passing between the decoders. More detailed examples and discussions can be found in [24], [15] and [6].

A stopping criterion for iterating until decoding is finished, can be implemented for Turbo codes. This can be done by choosing the most probable nodes according to the decoders. If two valid paths in the decoders are found in each trellis, the probability of changing those valid codewords is small enough to stop the decoder [6]. This allows for distinguishing between undetected and detected errors. Information about detected errors can be very useful in a communication system. This holds many advantages, including simulation and design and classification of codes. Another advantage of using a stopping criterion is that fewer iterations are used, requiring less processing resources. It should be noted that using a stopping criterion in hardware implemented systems can sometimes add complexity to the design by not having a constant throughput.

The design of an interleaver of a Turbo code is a determining factor for the error performance of the code. Using rectangular interleavers, similar to those used in Reed-Solomon codes, lead to a degraded error performance of the Turbo code. This implies that structured interleavers, which can be easily implemented, don't produce Turbo codes with good performance. Low-weight codewords are the reason for the loss of error performance. When using a random interleaver, the probability of having low-weight codewords in both decoders becomes much smaller. A disadvantage of Turbo codes is that even for random construction of interleavers an error floor is introduced for E_b/N_o values smaller than approximately 10^{-5}. Designing interleavers for Turbo codes, which further decreases the probability of low-weight codewords, are much more difficult and can't completely eliminate low-weight codewords [6].

2.2.6 LDPC codes

Also known as Gallager codes, LDPC codes were discovered by Gallager in 1962 [1]. For more than thirty years these codes were largely ignored, due to the lack of simple decoding techniques. MacKay rediscovered these codes in 1995 and used iterative decoding to show channel coding gains with an efficient iterative decoding algorithm [2]. These decoders have similarities to the iterative message passing decoders used in Turbo codes. In the last few years LDPC codes received a lot of attention ([25], [26], [27], [3], [4]), showing error performances approaching the Shannon limit [3].

LDPC codes are a class of linear block codes, with the characteristic of having a sparse parity check matrix **H**. The sparseness of **H** can be exploited to give simple iterative decoders for codes of any length, as will be shown in Section 3.2. There are two classes of LDPC codes: regular codes and irregular codes [15]. Irregular codes have a random construction of **H**. Random construction of **H** produces codes which perform like the random codes used by Shannon in the proof of his channel coding theorem [15]. These irregular codes show very good error performance approaching the Shannon limit as N increases [3]. It was shown by Chung *et al.* [3] using a code with a rate of 0.5 and $N = 10^7$, that an error performance can be achieved within 0.04 dB of the Shannon limit on an additive white Gaussian noise (AWGN) channel.

The row weight w_r and column weight w_c is an important measure of describing regular LDPC codes, where the weight of a vector is defined as the number of non-zero elements in the vector. Regular LDPC codes are codes in which the row weight w_r and column weight w_c are constant throughout **H**. When the weight of a code is maintained, the Hamming distance between codewords increase as N increases. The Hamming distance between two codewords is defined as the number of positions at which the codewords differ. This is what inherently causes LDPC codes to be good codes, with a very small probability of introducing a decoding error. A decoding error can be defined as the decoder in Figure 2.2 choosing the wrong codeword x from the received codeword x^*. Decoder failures happen when the received data differs from all valid codewords by a specified distance. Decoder failures can be used to detect errors and function as an error detection scheme. LDPC codes have excellent distance properties, which make the probability of an undetected error very small.

The biggest advantage of LDPC codes is due to the simplicity of sparse matrix multiplication. This imposes a decoding algorithm with low complexity. This property was first used by MacKay to give an efficient iterative decoder for LDPC codes [2]. Efficient decoders can be obtained through the use of parallel hardware implementations and by using techniques such as message passing [27], [25]. A disadvantage of LDPC codes is that the generator matrix **G** is not necessarily a sparse matrix and the complexity of encoding can be $O(N^2)$.

Decoder

Efficient decoders can be realised for LDPC codes by implementing iterative message passing decoders. An LDPC code can be completely described by its parity check matrix **H**. A good graphical representation of an LDPC code, or parity check matrix, is the Tanner graph, as discussed in Section 2.2.4. Within a Tanner graph two sets of nodes are defined. The first set is the variable nodes, represented by the individual bits of a valid codeword x, with a length of N bits. The second set is represented by the check nodes. This set represents all the parity constraints that need to be met to provide a valid codeword x. A soft decision belief propagation decoder is one which sends probabilities between the variable nodes and check nodes. This is known as message passing, where local messages are used with low complexity to find the solution of a complex global problem. A maximum likelihood decoder of a random-like code such as an LDPC code is one which maximises $P(x|x^*, \mathbf{H} \times x = 0)$. However, all codewords need to be stored for such an implementation, which implies a search over 2^K codewords for decoding, growing exponentially as K increases. The belief propagation decoder uses iterative message passing between check and bit nodes to find the codeword x^*, satisfying the condition $\mathbf{H} \times x^* = z$, where the syndrome $z = 0$ [6].

Encoder

Encoding can be implemented by simply multiplying the data bits with the generator matrix. The generator matrix **G** of an LDPC code is not necessarily a sparse matrix. This implies that the complexity of encoding can scale quadratically with the block size N, when encoding is implemented by multiplication. A few design methods will be discussed which decrease the complexity of encoding an LDPC code. It is shown in [28] that by doing some pre-processing, a method can be implemented to dramatically decrease the encoding complexity for any matrix **H**. A second method is to implement staircase codes. By introducing a staircase structure in **H**, encoding can be done in linear time compared to N [6]. Unlike the iterative decoder, implementation of the encoder by simple multiplication only requires a single computation. Considering relatively short length LDPC codes for hardware implementation in this thesis, the computational requirement of the encoder falls within acceptable margins.

Regular quasi-cyclic (QC) LDPC codes are an important subclass of LDPC codes, addressing some of the practical shortcomings of randomly generated LDPC codes [29]. It has been shown in [30] that for N smaller than 10 000, QC LDPC codes have error performances comparable to randomly generated LDPC codes. Due to their cyclic property, QC codes are well defined in terms of their structure for the parity check matrix. The parity check matrix **H** of a QC code can be represented as follows:

$$\mathbf{H} = \begin{bmatrix} \mathbf{I}_{s(1,1)} & \mathbf{I}_{s(1,2)} & \cdots & \mathbf{I}_{s(1,b)} \\ \mathbf{I}_{s(2,1)} & \mathbf{I}_{s(2,2)} & \cdots & \mathbf{I}_{s(2,b)} \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{I}_{s(a,1)} & \mathbf{I}_{s(a,2)} & \cdots & \mathbf{I}_{s(a,b)} \end{bmatrix} \tag{2.2.7}$$

where every I_s is a $k \times k$ identity matrix cyclically shifted by variable s. A cyclic shift with offset s implies that each row is shifted to the right s times. Equation 2.2.8 gives an example of the identity matrix I_s cyclically shifted, where the offset $s(a, b) = 2$.

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix} \rightarrow \begin{bmatrix} 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \end{bmatrix} \tag{2.2.8}$$

Note that k must be a prime number as shown in [30]. The parity check matrix **H** is a size $ka \times kb$ matrix. Most QC codes are described by the notation (N,a,b), due to the use of the identity matrix in H, $w_c = a$ and $w_r = b$. This corresponds to a code of rate $R \geq 1 - \frac{a}{b}$, where the code rate can be greater than this lower bound where linear dependencies exist between rows in H. Within this structure every bit will participate in w_c checks and every check is calculated by w_r associated bits. The Tanner graph produces a graphical representation of the relationship between check nodes and the bit nodes [31]. A cycle in a Tanner graph is a sequence of related check and bit nodes beginning and ending in the same point. Any cycle of length 4 needs to be avoided. An example of a cycle of length 4 is given in Chapter 3.2.5. Randomly generated codes make use of their sparse property to avoid this situation. For codes with smaller lengths, as considered in this study, the systematic generation of codes can be used to ensure that cycles of length 4 can be avoided. The girth of a code is the length of the smallest cycle in the Tanner graph. Designing codes with a large Hamming distance and girth is desirable.

The encoding of QC LDPC codes have lower complexity than the encoding of any other type of LDPC code. The encoding complexity for randomly generated LDPC codes is $O(N^2)$, but it has been shown that for QC codes a parallel processing implementation can reduce it to $O(N)$, and for a serial implementation it is linearly proportional to the number of the parity check bits [26], [30].

Reversible LDPC codes are another class of codes with practical encoders. By designing reversible LDPC codes, the message passing iterative decoder can be used for encoding and decoding. Designing a circuit that represents the factor graph of the Tanner graph, the same circuit can be used for encoding and decoding. In applications where circuit area is limited, such an implementation holds much promise. It has been shown in [32] that their proposed codes based on the Jacobi method for encoding provide reversible LDPC codes that compare well to the performance of randomly constructed QC LDPC codes.

2.2.7 Protocol design

To integrate a project specific error control strategy, a protocol strategy to facilitate this implementation needs to be designed. The protocol strategy was designed for implementation on the IS-HS II satellite project, as discussed in Section 2.2.2. Four layers of the standard OSI model will be used to give a description of the network protocol design. All layers have the same description as those of the standard OSI model, with the exception of defining two sublayers within the standard OSI data link layer, as will be discussed. The physical layer will consist of all RF equipment, digital-to-analogue, analogue-to-digital conversion and detection.

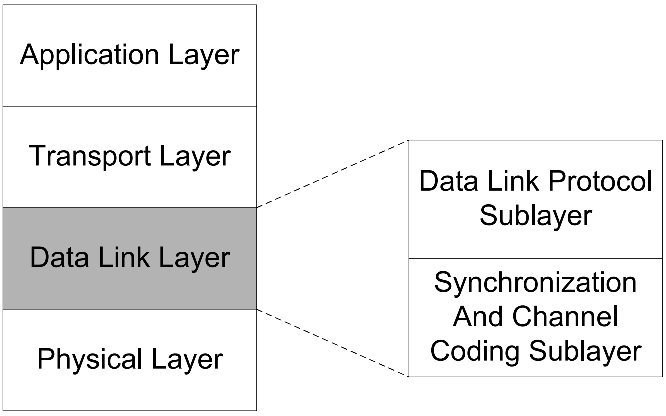

Application Layer
Transport Layer
Data Link Layer
Physical Layer
Data Link Protocol Sublayer
Synchronization And Channel Coding Sublayer

Figure 2.8: Layered protocol model

The data link layer will use an implementation based on standards established by the ECSS [22], [33]. The ECSS is a body governed by the ESA, national space agencies and European industry associations. Two sublayers will be defined within the data link layer. As shown in Figure 2.8, the bottom sublayer interconnecting with the physical layer, will be the synchronisation and channel coding sublayer. This sublayer will be responsible for synchronisation, error control coding and providing acceptable bit transition densities. The data link protocol sublayer will function on top of the synchronisation and channel coding sublayer. This sublayer will interconnect between the synchronisation and channel coding sublayer and the transport layer. The telemetry (TM) transfer frame protocol designed for space data links will be implemented on this sublayer [33]. Using this implementation, a mission specific error control strategy can be implemented to optimise the overall link throughput, as will be discussed in Section 3.2.

An automatic repeat-request (ARQ) strategy will be implemented on the transportation layer. The satellite software system communicates in the application layer. The application layer will consist of all databases and means to ensure useful data is downloaded to the central server or base station. The transport and application layers will be implemented on the OBC.

2.3 Conclusions and finding

Concepts of LEO satellite communication is presented, and the system architecture of the IS-HS II satellite system is given. Looking at this project, it is shown that LEO satellites are power limited systems, with varying link margins and large propagation delays. Error control coding is introduced as a mechanism to increase reliability and throughput of poor quality links. The overall throughput of LEO satellites can be optimised with the implementation of error control strategies. Linear block codes use a mapping scheme to convert information to be transmitted over the noisy channel into codewords, by adding redundancy to the code blocks. The Tanner graph is explained as a graphical representation of linear block codes, to be used as a powerful tool in the design of codes, encoders and decoders. An overview is given of some of the popular coding schemes. Conventional codes such as the Hamming code, Reed-Muller, Reed-Solomon and BCH codes have been used extensively over the last few decades. Their inability to scale to codes with larger block sizes, or complexity for convolutional codes, have made them lack the error performance of sparse graph codes. Concatenated codes provide a relatively strong code by combining more than one simple code. The Reed-Solomon and convolutional concatenated code is shown as viable options, due to many implementations in space missions. In recent years sparse graph codes have shown superior error performance, approaching the Shannon limit and outperforming the concatenated and conventional codes. Using an iterative message passing decoder, random-like sparse graph codes result in practical implementable codes with extremely good performance.

Among sparse graph codes two families of codes that show extremely good performance for error correction are Turbo codes and LDPC codes. Turbo codes have simple encoders, implementing two recursive concatenated codes with an interleaver between them. The design of the interleaver is an important part of the design to produce good Turbo codes. Turbo codes show good performance within the waterfall region, for poor links to moderately noisy channels. Turbo codes show excellent performance for decoded error performance down to error probabilities of around 10^{-5}. In this region an error floor is introduced. Due to its simple encoders, Turbo codes are excellent for deep space communication where limited resources are available at the sender. LDPC codes rarely show error floors, and when ensuring that the parity check matrix has few columns or no columns of weight 2, these error floors can be eliminated. LDPC codes are versatile and it is easy to create codes with excellent performance for many code rates and code lengths. Unlike the design complexity of interleavers for Turbo codes, the simplicity of designing LDPC codes makes them freely available. A big advantage of LDPC codes are their capability to distinguish between undetected and detected errors. Increasing the code length of sparse graph codes increases the error performance of the codes, but with message passing decoders, the complexity of such decoders doesn't grow exponentially as expected. LDPC codes provide excellent practical codes for the implementation on LEO satellite communications systems, and can take advantage of parallel processing in hardware implemented systems to provide systems with efficient hardware utilisation. This will be expanded upon in the next chapter.

Chapter 3

Design and implementation

3.1 Introduction

In the previous chapter an overview of error correction schemes was presented. This chapter will focus on the implementation and design considerations for LDPC codes. Theoretical concepts and implementation of the sum-product, log-likelihood and min-sum approximation decoders will be explained. A detailed description of the sum-product algorithm will be given and used as a framework for all the iterative message-passing decoders. Pseudo code is given for the min-sum approximation decoder, followed by some remarks on hardware implementation of the decoders. The normalised and offset min-sum decoders are presented to show performance improvements compared to the standard min-sum decoder. Preliminary results are given in this chapter to explain some of the design considerations, while the main results of the decoders will be presented in Chapter 4.

To enable the implementation of forward error correction on a communication system, the specific strategy should be supported by the protocol and be compatible with hardware requirements. The above leads to a protocol design for LEO satellite communication. Concepts and design considerations for implementing the channel coding and synchronisation sublayer will be introduced and explained. A description of the TM space packet protocol is provided. A protocol implementation and packet processing scheme is designed to avoid the necessity of higher layer protocols to do packet inspection of the data link layer. The chapter will be concluded by explaining the simulator, data capturing methods and satellite channel characteristics.

3.2 LDPC code implementations

The basic implementation of the encoding and decoding of an LDPC code is described by referring to the matrix multiplications given in Figure 3.1. In this illustration a half rate code is used. The code is described by its parity check matrix $\mathbf{H}$, where $\mathbf{G} \times \mathbf{H}^T = 0$. For the encoding process, a codeword x is calculated by multiplying the message with the generator matrix, giving $x = m \times \mathbf{G}$. In the encoder the message of size $1 \times K$ is multiplied by the generator matrix of size $(K \times N)$. This maps the message into a codeword x of size $1 \times N$. The $(4, 2)$ code has a code rate of 0.5 and it can be seen that a redundancy M of two bits are added for each message. For half rate codes, the data rate f_x will be double that of f_b. Note that only binary codes will be considered in this thesis. The codeword x is modulated on to the RF frequencies, and sent over the noisy satellite link. In Figure 3.1, $s(t)$ represents the modulated signal transmitted by the sender and $r(t)$ represents the received signal with the noise from the satellite channel. At the receiver the received signal $r(t)$ is demodulated, and the demodulated data (or codeword) x^* sent to the decoder. In the decoder the syndrome can be computed as $z = x^* \times \mathbf{H}^T$. The condition set in (2.2.4) implies that x^* is a valid codeword, if and only if, the syndrome z is equal to a zero vector. The value of x^* can be described as the codeword x with the noise added by the channel n, thus, $x^* = x + n$. The syndrome only represents the noise added by the channel and is not dependent on x. This is shown in the following equation:

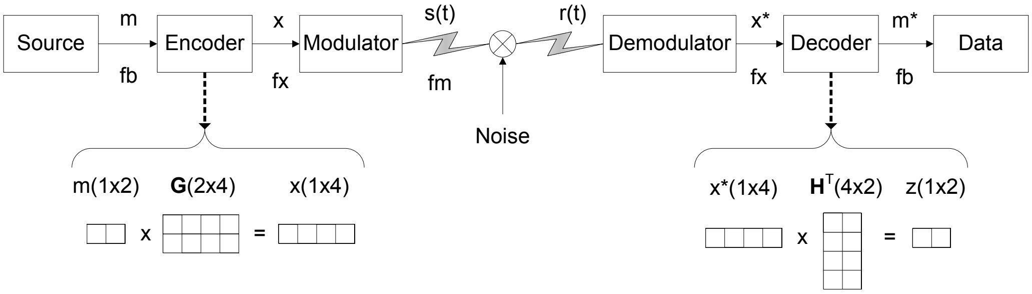


Figure 3.1: Block diagram of a communication channel implementing a (4,2) linear block code.

$$z = (x + n) \times \mathbf{H}^T \tag{3.2.1}$$

where the condition set in (2.2.4) implies that $x \times \mathbf{H}^T = 0$. Therefore, Equation 3.2.1 can be simplified to give:

$$z = n \times \mathbf{H}^T \tag{3.2.2}$$

The received codeword x^* can be mapped back to the original message m by extracting the data from the codeword. This is done by implementing a systematic generator matrix $\mathbf{G}$. A systematic generator matrix is one in which the identity matrix can be identified among the rows of $\mathbf{G}$. For this implementation, we define $\mathbf{G} = [\mathbf{P} \quad \mathbf{I}_k]$. The first K bits of the codeword x represent the parity bits of the message, and the last K bits represent the message m.

The generator matrix $\mathbf{G}$ of an LDPC code is not necessarily a sparse matrix. The complexity of encoding can, therefore, scale quadratically with the block size N, when encoding is implemented by simple multiplication. Some design methods are available to create an encoder that is less complicated than multiplying with a matrix which is not sparse. It is shown in [28] that by doing some pre-processing, a method can be implemented to dramatically decrease the encoding complexity for any matrix $\mathbf{H}$. A second method is to implement staircase codes. By introducing a staircase structure in $\mathbf{H}$, encoding can be done in linear time in terms of N [6]. QC codes also decrease the complexity, as discussed in Section 2.2.6. The encoder is implemented by taking the input bit stream and multiplying it with the generator matrix, as shown in Figure 3.1. Unlike the iterative decoder, implementing the encoder by multiplication only requires a single computation. Considering the relatively short length LDPC codes for hardware implementation, the computational requirement of the encoder falls within acceptable margins for implementation.

When using an iterative decoder, the fact that the encoder requires more processing does not pose a problem. The encoding involves a single multiplication, and therefore the requirements of throughput on a transmitting device are not too severe.

The decoder is chosen to be a sum-product decoder, also known as an iterative probabilistic decoder. LDPC codes are described in terms of their code structure in the parity check matrix $\mathbf{H}$. This structure dramatically influences the design and complexity of the decoder. A regular LDPC code will be implemented. The Tanner graph, as explained in Section 2.2.4, gives a graphical representation of linear block codes. This graphical representation shows the relationships between the variable nodes and check nodes, as described by the parity check matrix $\mathbf{H}$. Each '1' in the parity check matrix represents an edge on the Tanner graph, indicating a connection between a check and variable node. The min-sum decoder utilises a message passing algorithm by making use of local messages between nodes requiring only simple processing, to solve a complex global problem. The aim of the decoder is to find the solution of x which maximises the probability of $x \times \mathbf{H}^T = z$, where x is defined as the received code vector and z is the syndrome. The decoder should find the most probable value of vector x to satisfy the condition. The codeword viewpoint will be used throughout this thesis, stating that the value of z should be zero for the correct codeword to be found.

Regular codes introducing a regular code structure in $\mathbf{H}$ allow for efficient hardware implementations of the decoder, while comparing well with the performance of randomly constructed codes for short to medium length codes [30]. LDPC codes show good performances and good distance properties for column weights of $w_c \geq 3$ [15]. The sum-product algorithm, explained in Section 3.2.1, also makes poor progress in codes with high column weights [6]. Considering these constraints, all half rate codes in this thesis are codes with $w_c = 3$ and $w_r = 6$.

In Section 2.2.4 the Tanner graph was introduced and an example given for a Hamming code. All codes used in this thesis will have a Tanner graph similar to the section shown in Figure 3.2.

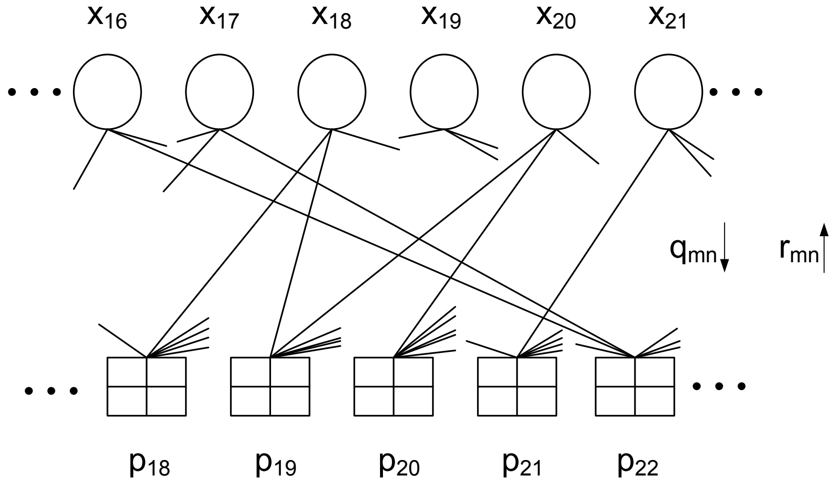


Figure 3.2: Section of a Tanner graph for a (96,48) LDPC code with $w_c = 3$ and $w_r = 6$.

It can be seen that each variable node in Figure 3.2, denoted as $(x_{16}, x_{17}, ..., x_{21})$, is connected to three check nodes. This is due to the column weight of $\mathbf{H}$ being equal to three. Similarly each check node, denoted as $(p_{18}, p_{19}, ..., p_{22})$, is connected to six variable nodes, where $w_r = 6$. The direction of messages q_{mn} and r_{mn} is also shown in this figure and will be explained in Chapter 3.2.1. Due to the long length of the code, only a small segment is shown and short lines indicate connections to nodes not included in the section displayed. The (96,48) code has 96 variable nodes $(x_1, x_2, ..., x_{96})$, and 48 parity check nodes $(p_1, p_2, ..., p_{48})$. The notation used in [15] will be adopted to describe the sets, used in the explanation of the different decoders. M_n represents the set of checks in which the variable node x_n participates. This corresponds to all the non-zero elements in column n of the parity check matrix. As an example we look at the variable node x_{18} in Figure 3.2, described by set M_{18}. In the eighteenth column of $\mathbf{H}$, there are non-zero elements in $\mathbf{H}_{18,18}$, $\mathbf{H}_{19,18}$ and $\mathbf{H}_{27,18}$. From this we can write the set as $M_{18} = \{18, 19, 27\}$. For a regular code, the number of elements in M_n will always be equal to w_c. $M_{(n)/r}$ involves all the checks of column n, except the check in row r. In this example the set of x_{18}

involving all the checks in column 18, except the check in row 19, will give the set $M_{(18)/19} = \{18, 27\}$.

The set of bits that participate in check p_m will be defined by the set N_m, where the number of elements in the set will be equal to w_r. This set represents all the non-zero elements in the m^{th} row of $\mathbf{H}$. As an example we look at the variable node p_{19} in Figure 3.2. In this example $N_{19} = \{18, 20, 35, 52, 54, 68\}$. Similar to the above notation, $N_{19/20} = \{18, 35, 52, 54, 68\}$.

All variable nodes are the bits received by the receiver and passed to the decoder. The check nodes should not be seen as bits physically sent over the communication channel, but rather as conditions set in the encoder (calculating the codeword).

All decoders considered, implement an iterative message passing algorithm as discussed in Section 2.2.6. A top-level flow diagram of the iterative message passing algorithm is given in Figure 3.3. At the start of decoding, the codeword x^* is read with the length of the codeword as N bits. The codeword is now used to initialise the variables used in the horizontal and vertical step. Note that horizontal and vertical refer to the calculation of the elements of the parity check matrix $\mathbf{H}$, as will be explained in Section 3.2.1. In the horizontal step, the check nodes will be updated. These updated check nodes will be used to compute new values for the bit nodes. The stopping criterion will use the variables computed by the vertical and horizontal steps, to calculate an estimated codeword. The syndrome is computed from the estimated codeword, and if the syndrome z is zero, the decoder will declare that the information received is the correct data sent by the transmitter and will return the message bits m^*, which are extracted from the codeword x^*. If the syndrome z is not zero, the codeword has errors. This indicates that another iteration should be done to fix the errors in the codeword. At this point, the number of iterations used by the algorithm should be checked. If the number of iterations used by the algorithm is smaller than the maximum number of iterations specified for the algorithm, the current iteration number should be incremented before the next horizontal step is performed. If the current iteration is larger than the maximum allowable iterations for the algorithm, a decoder failure should be declared. This means that more errors were introduced on the channel, than can be corrected by the decoder within the specified number of maximum iterations.

This is a basic flow that will be implemented by all the decoders. The concepts of the different algorithms will be explained more comprehensively in Section 3.2.1. The log-likelihood ratio and min-sum approximation algorithms will be explained by stating their differences from the sum-product algorithm.

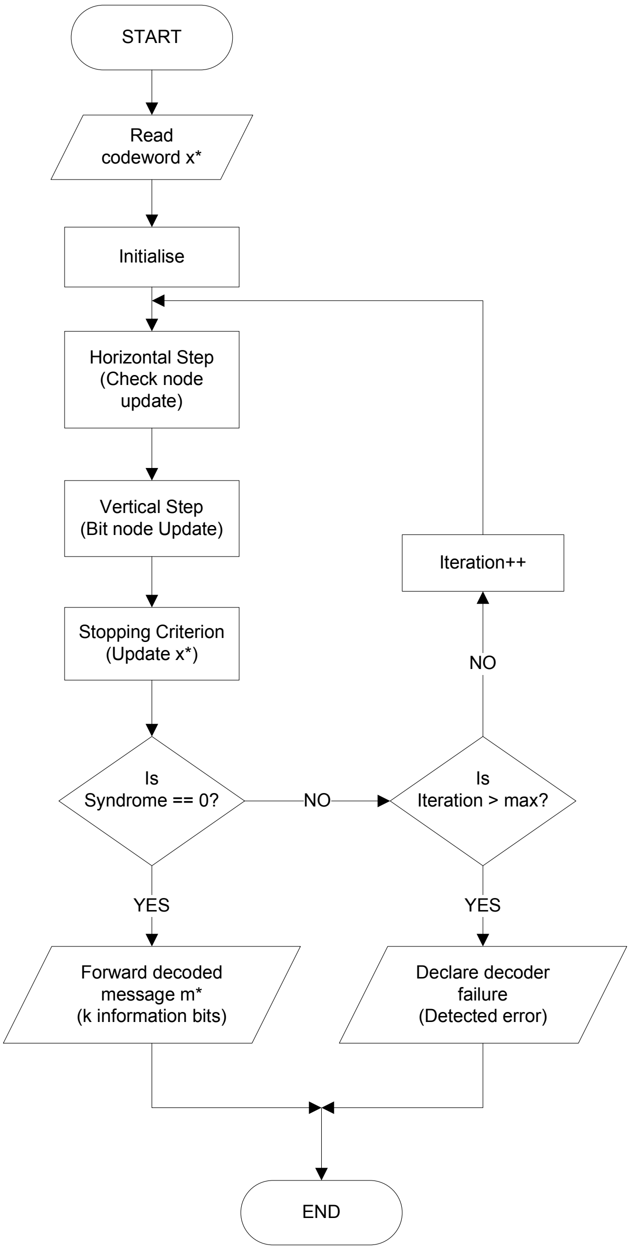


Figure 3.3: Top-level design flow of the iterative message passing decoder.

3.2.1 Sum-product algorithm

The sum-product algorithm, or belief propagation algorithm, is based on the parity check matrix, where $\mathbf{H}_{m,n}$ will represent the elements of the parity check matrix.

There are two types of messages passed in the decoder: messages passed from the variable nodes to the check nodes, denoted as q_{mn}, and messages passed from the check nodes to the variable nodes, denoted as r_{mn}. The direction in which these messages are sent, is shown in Figure 3.2. Every message will consist of two probabilities. In the case of messages sent from the variable nodes to the check nodes, q^0_{mn} and q^1_{mn} will be sent. q^0_{mn} represents the probability that the variable is a 0, and q^1_{mn} the probability that the variable is a 1, where $q^0_{mn} + q^1_{mn} = 1$. The probabilities received from a soft decision decoder for a node will be p^0_n and p^1_n, representing the probability for a 0 and a 1 respectively. The sum-product algorithm will now be explained in four steps.

A regular code with column weight $j = 3$ and row weight $k = 6$ will be assumed.

1. **Initialisation:** The variables q_{mn} are assigned to the probabilities of the codeword received from the channel. Thus, we have $q^1_{mn} = p^1_n$ and $q^0_{mn} = p^0_n$, where $\mathbf{H}_{m,n} = 1$.

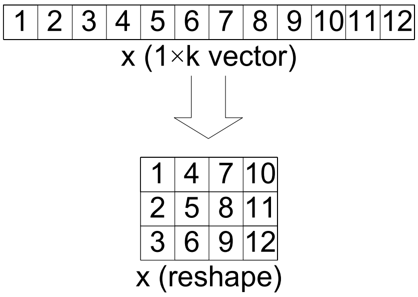
2. **Horizontal step:** The probabilities for q_{mn} are used to determine the probabilities for r_{mn}, where r^0_{mn} is the probability that the m^{th} check is satisfied when the n^{th} variable has the value 0. Similarly, the probability for r^1_{mn} is calculated. Every check node calculates the parity of k variable nodes.

3. **Vertical step:** The calculated probabilities for r_{mn} are sent to variable nodes to update the probabilities of q_{mn}. This is done by calculating the combined probability of the j checks involved, (excluding the probability of the m^{th} node computed), in each variable node, and combining it with the probability p_n. For each variable node, the q_{mn} probabilities are updated for each of the check nodes associated with that variable node.

4. **Stopping Criterion:** In this step we compute the posterior pseudo-probabilities. For each variable node, the probabilities for p_n is computed, given as p^0_n and p^1_n. This is done by multiplying the prior probability of p_n with the probabilities given by its associated check nodes q_{mn}. Here the variable node's value is set to the highest probability,

creating the new vector $\widehat{x}$ from the received vector x^{*}. If the condition $\widehat{x} \times \mathbf{H^T} = 0$ is met, then $\widehat{x}$ is given as the decoded codeword, otherwise we start again at step 2, with the updated variable nodes.

In Figure 3.5, a graphical representation of the decoding process is shown. A (204,102) code was used, with a parity check matrix $\mathbf{H}$ designed by D.J.C. MacKay [34]. In Figure 3.5 the codeword x is presented in a graphical way to illustrate different stages of the coding system. Note that x is a vector and displayed as a matrix for viewing purposes only. This is illustrated in Figure 3.4.



1	2	3	4	5	6	7	8	9	10	11	12

1	4	7	10
2	5	8	11
3	6	9	12

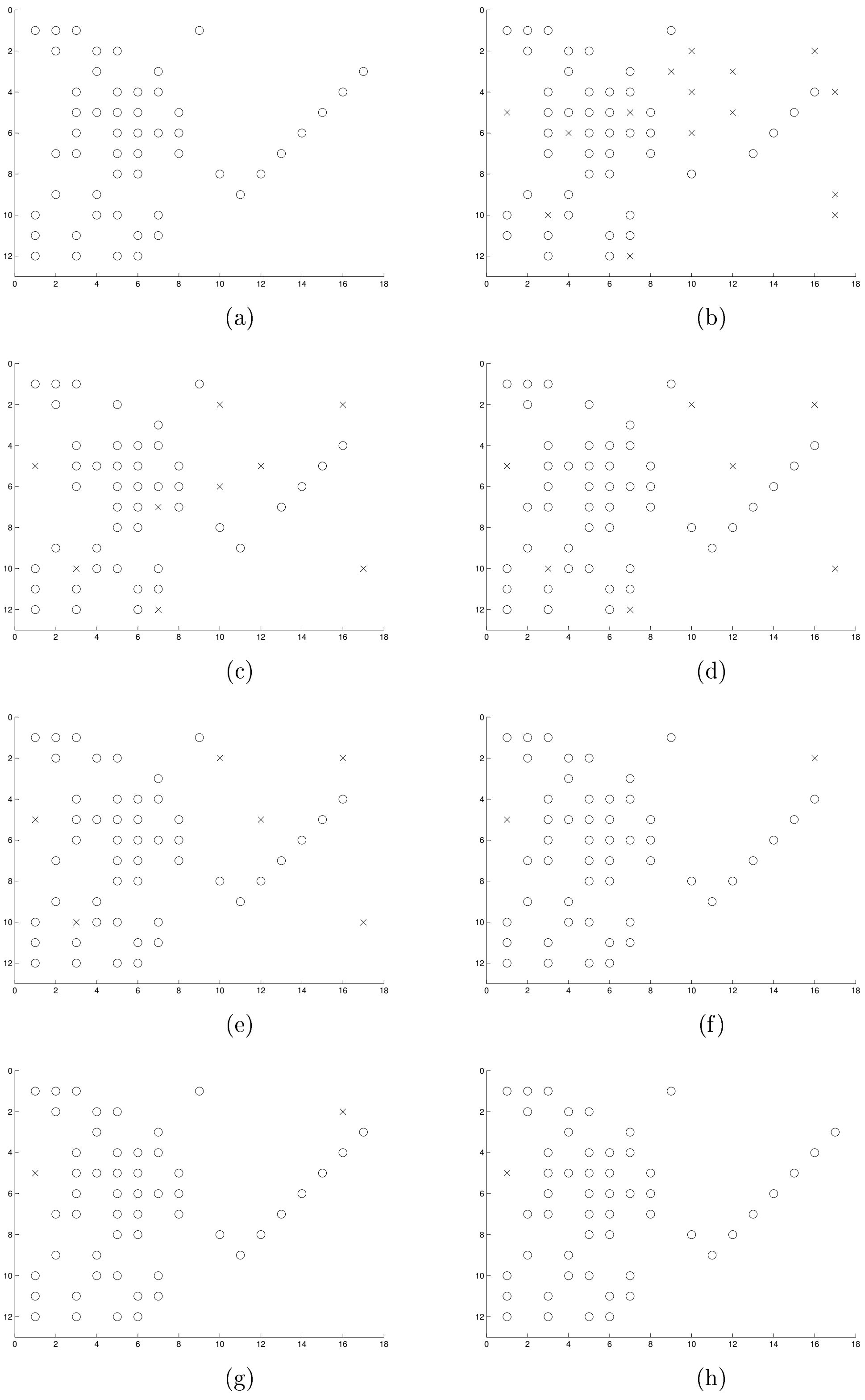
Figure 3.4: Reshape the codeword for visual representation.

Figure 3.5(a) represents x as computed by $x = m \times \mathbf{G}$. All ones in the codeword is represented by a circle. Because $\mathbf{G} = [\mathbf{P} \quad \mathbf{I}_k]$ the parities are represented by the left hand side of the matrix and the message m on the right. The codeword is corrupted by 22 errors, which is illustrated in Figure 3.5(b). A value of one corrupted by the channel is left out in the figure, and a zero corrupted to represent a one is illustrated by a cross. Figure 3.5(c) to Figure 3.5(h) show the value of $\widehat{x}$ after the first six iterations. The code is able to correctly decode the codeword after 7 iterations.

3.2.2 Log likelihood ratio algorithm

The BP log likelihood ratio (LLR) algorithm is similar to the one explained in the previous section, but will be expressed in terms of probabilistic ratios. The advantage of using the LLR algorithm is that the normalisation that needs to be computed in the vertical and horizontal steps of the standard BP algorithm is avoided. The log likelihood ratios from the channel, which represent the intrinsic information, can be calculated as:

$$\lambda_n = \log\left(\frac{P(x_n = 1)}{P(x_n = 0)}\right). \tag{3.2.3}$$



(a) (b)

(c) (d)

(e) (f)

(g) (h)

Figure 3.5: Illustration of the BP decoder for a (204,102) code. (a) Codeword generated by sender. (b) Received codeword x^* with 27 errors introduced by the channel. (c) $\widehat{x}$ after first iteration. (d) - (h) $\widehat{x}$ for second to sixth iteration. The codeword was successfully decoded after 7 iterations.

The channel reliability can be computed as $L_c = 2\sqrt{E_x}/\sigma^2$. For the LLR decoder we express the variable representing the check nodes as $\mathbf{L}_{m,n}$, and the variable nodes as $\mathbf{Z}_{m,n}$. During the initialisation, we set $\lambda_n = L_c x_n^*$. We use the same notation as defined in Section 3.2.1. As shown in [15], the horizontal step can be simplified to:

$$\mathbf{L}_{m,n} = -2\tanh^{-1}\left[\prod_{n' \in N_{(m)/n}} \tanh\left(\frac{\mathbf{Z}_{m,n'}}{2}\right)\right]. \tag{3.2.4}$$

The vertical step update can be calculated as

$$\mathbf{Z}_{m,n} = \lambda_n + \sum_{m' \in M_{(n)/m}} \mathbf{L}_{m',n}. \tag{3.2.5}$$

The stopping criterion is determined by calculating the updated variable node probability as:

$$\lambda_n = L_c R_n + \sum_{m \in M_{(n)}} \mathbf{L}_{m,n}. \tag{3.2.6}$$

Using this result we set $\widehat{x}_n = 1$ if $\lambda_n > 0$, and otherwise set $\widehat{x}_n = 0$. If the condition $\widehat{x} \times \mathbf{H} = 0$ is met, we send $\widehat{x}$ as the decoded codeword.

Both representations of the BP algorithm produce exactly the same results. In Figure 3.6, results are shown for LDPC codes with four different code lengths, using the BP decoding algorithm. A maximum number of 100 iterations was used for decoding. The parity check matrices were designed by D.J.C. MacKay [34]. The performance of the code improves as N increases. Figure 3.6 shows that at a BER of 10^{-5}, the (96,48), (204,102), (408,204) and (1008,504) codes show coding gains of 4.98 dB, 5.46 dB, 6.12 dB and 7.02 dB respectively.

3.2.3 Min-sum approximation decoder

BP decoders can be simplified by using the min-sum approximation algorithm. Calculations for updating the check nodes are simplified by using the following approximation: the horizontal step of the LLR BP algorithm, presented in Section 3.2.2, can be viewed as determining the last two states in a Markov model [6]. The min-sum approximation can now be used to simplify (3.2.4) to

$$\mathbf{L}_{m,n} \approx \prod_{n' \in N_{(m)/n}} \operatorname{sgn}(\mathbf{Z}_{m,n'}) \times \min_{n' \in N_{(m)/n}} |\mathbf{Z}_{m,n'}|. \tag{3.2.7}$$

Using this equation for updating the check node in the horizontal step, we can reduce the decoding algorithm's complexity. This approximation only requires the computation of a minimum value, determining the sign, and doing addition. This is a desirable result for the implementation of the BP algorithm in hardware [35].

```
input  : Codeword $x^*$ (N-bit vector)
output: Message $m$ (K-bit vector), success-STATUS
Initialise;
for $n \leftarrow 1$ to length of codeword $x^*$ do
    foreach element $m$, where $H_{mn} = 1$ do
        $Z_{mn} = x^*(n)$
    end
end
while iteration < iterationmax and Success == 0 do
    Horizontal step;
    for each row (m) in $\mathbf{H}$ do
        Calculate the |min|, second |min| and $\prod$ signs for non-zero
        elements in $\mathbf{Z}_{mn}$;
        if $\mathbf{Z}_{mn} == |\text{min}|$ then ;
        $\mathbf{L}_{row,n}$ = second $|\text{min}| \times \prod \text{sign} \times \text{sign}(\mathbf{Z}_{mn})$  else ;
        $\mathbf{L}_{row,n} = |\text{min}| \times \prod \text{sign} \times \text{sign}(\mathbf{Z}_{mn})$;
    end
    Vertical step;
    foreach column of $\mathbf{H}$ do
        $sum(column) = \sum \mathbf{L}_{m,column}$;
        $\mathbf{Z}_{mn} = x(column) + sum(column) - \mathbf{L}_{m,column}$;
    end
    Stopping criterion;
    $\mathbf{x}(n) = x^*(n) + sum(n)$ ;
    if $\mathbf{x}(n) > 0$ then $\mathbf{x}(n) = 0$ else $\mathbf{x}(n) = 0$ ;
    Syndrome = $\mathbf{x}^* \times \mathbf{H}^{\mathbf{T}}$ (in GF(2)) ;
    if Syndrome == 0 then
        Success = 1
    end
    iteration ++
end
```

Algorithm 1: Min-sum approximation algorithm

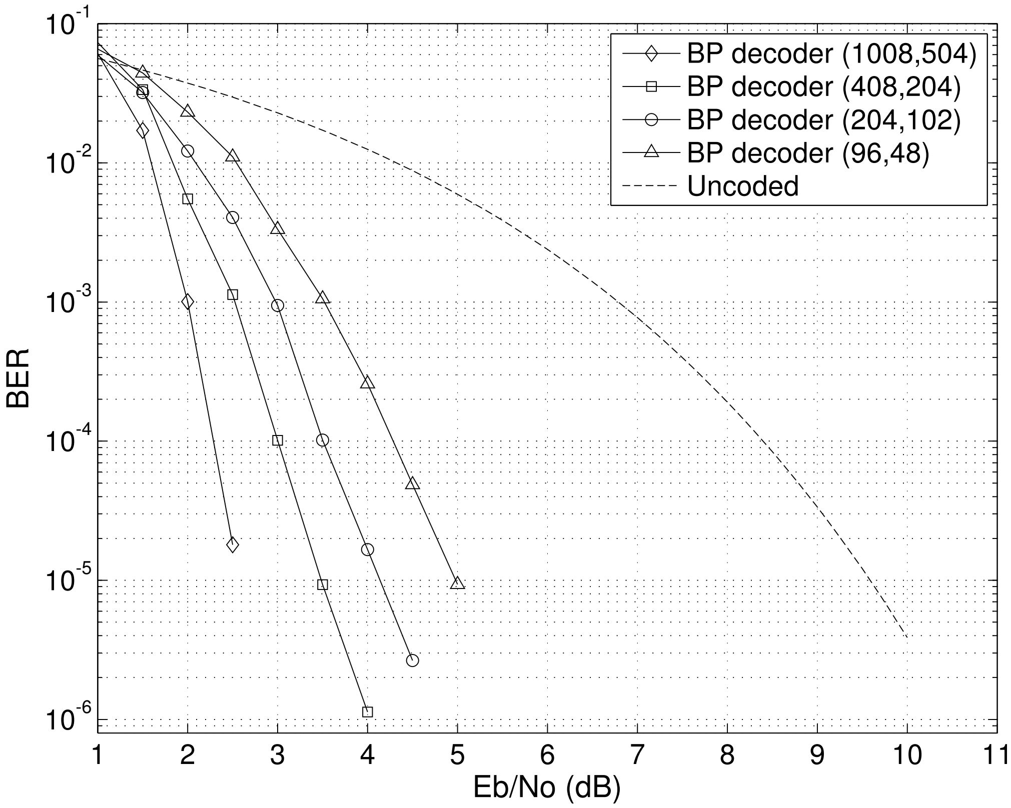


Figure 3.6: Performance results for the BP algorithm on different code lengths.

In Algorithm 1 the pseudo code for the decoder is presented. This is similar to the actual implementation of the decoder in hardware and the function used in the simulator. The simulator itself will be explained in Section 3.5. The input to the decoder will be a N-bit vector, and the output a K-bit vector for an (N, K) code. The variable $Success$ will indicate whether a valid codeword was found, or an error detected. The basic instructions of the four steps of the algorithm as explained in Section 3.2.1, will be highlighted. It should be noted that the variable matrices $\mathbf{Z}_{mn}$ and $\mathbf{L}_{mn}$ are sparse matrices, only having non-zero elements in its elements corresponding to non-zero elements in the parity check matrix $\mathbf{H}$. In lines $2 - 6$, all the non-zero elements of each column are assigned the value of the received probability calculated from $x^*(n)$. Depending on the implemented demodulator, these values may be used directly from the received vector x^* or require some pre-processing. It should be noted that $\mathbf{L}_{mn}$ and $Success$ will be initialised to zero. The $while$-loop in line 7 will ensure that the message-passing decoder will iterate between the horizontal and vertical steps until a valid codeword is found or the maximum number of iterations reached. The horizontal step will update the variable $\mathbf{L}_{mn}$ given in (3.2.7). This is done in the for-loop in lines $9 - 14$. For each row, the minimum and second minimum will be determined for the absolute values of the elements in $\mathbf{Z}_{mn}$. The product of the signs of the elements is also determined. These values are used in the if-statement of lines $11-13$ to update the elements of the row in $\mathbf{Z}_{mn}$. These calculations avoid the need of computing the minimum and calculating the sign of five, or $(w_r - 1)$, elements for each non-zero element. With this implementation, a minimum, second minimum and sign is calculated once for w_r elements, and can be used to update all those elements. In the vertical step the sum of the w_c columns are determined for each row. These values are used with the updated values of $\mathbf{L}_{mn}$ and the initial values of x^*, to update $\mathbf{Z}_{mn}$. The stopping criterion is used to check if a valid codeword was found. The value of *sum* calculated in the vertical step, and initial values for the codeword x^* are used to calculate the most probable codeword x. This codeword is multiplied with $\mathbf{H}$ in GF(2) to calculate the syndrome. If the syndrome is zero, then the codeword was successfully decoded and the message bits are returned with the variable *Success*, set to 1.

3.2.4 Normalised min-sum decoder

The normalised min-sum decoder improves the performance of the min-sum approximation decoder by introducing a normalisation factor α in the check node update or horizontal step [36]. We have:

$$\mathbf{L}_{m,n} \Leftarrow \mathbf{L}_{m,n} \times \alpha \tag{3.2.8}$$

where α is a constant smaller than 1.

The performance can be increased by changing α for each iteration, but in this implementation a fixed value will be assigned to α to reduce the complexity of the decoder. The optimum value of α varies for each unique parity check matrix $\mathbf{H}$ and can be determined by the analytical technique called density evolution [7], or simply by simulation. By simulation it was shown that values ranging from 0.7-0.8 result in good performance for all the codes considered in the thesis. These results are shown in Section 4.2.

3.2.5 Offset min-sum decoder

The offset min-sum decoder is similar to the normalised min-sum decoder. In this decoder, we use β to reduce the value of $\mathbf{L}_{m,n}$ in the horizontal step, such that

$$\mathbf{L}_{m,n} \Leftarrow \text{sgn}(\mathbf{L}_{m,n}) \times \max(|\mathbf{L}_{m,n}| - \beta) \tag{3.2.9}$$

where the value of $\mathbf{L}_{m,n}$ is set to zero when the initial value of $\mathbf{L}_{m,n}$ is smaller than β [36].

The offset min-sum and normalised min-sum decoders give similar performance when estimating good values for α and β. In Section 4.2 the performance of the normalised min-sum approximation algorithm will be compared to the performance of the BP algorithm introduced in Section 3.2.1.

It is desirable to design codes with large girth. Girth is defined as the smallest cycle in the Tanner graph. In Figure 3.7 the dotted lines show a cycle of length 4 between variable nodes x_{17} and x_{18}, and check nodes p_{18} and p_{19}. Small cycles in the Tanner graph of a code reduces its performance by limiting the distribution of probabilities in the graph. All codes used in this thesis were tested to avoid cycles of length 4.

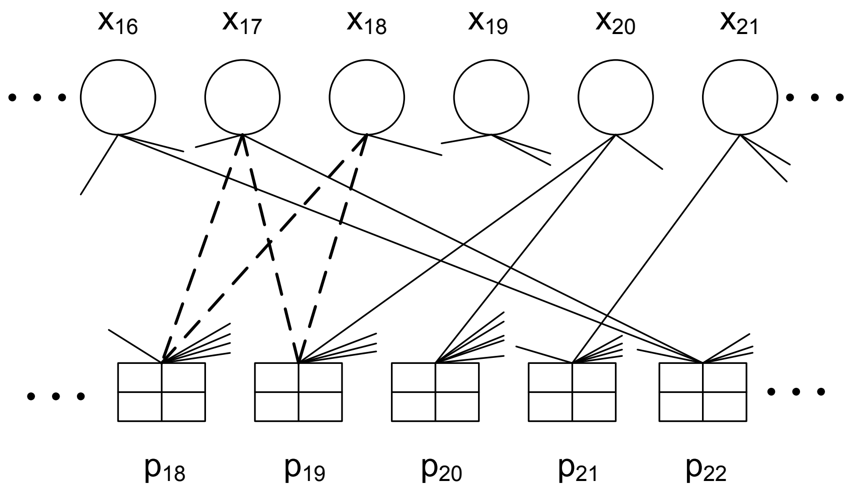


Figure 3.7: Tanner graph showing a cycle of length 4.

3.3 Protocol implementation

An overview of the protocol design and OSI model was presented in Section 2.2.7. The protocol strategy was designed for implementation on the IS-HS II satellite project and falls within the scope and hardware feasibility of this project, as discussed in Section 2.2.1.

3.3.1 Synchronisation and channel coding sublayer

The implementation of the synchronisation and channel coding sublayer, as discussed in Section 2.2.7, will now be explained according to Consultative Committee for Space Data Systems (CCSDS) specifications in [37]. Each frame within this sublayer will be referred to as a channel access data unit (CADU). During transmission, the TM transfer frame received by the data link protocol sublayer will be encoded into an LDPC codeblock. The LDPC code to be used cannot guarantee the minimum symbol transition density required for synchronisation in the demodulator of the receiver. A pseudo-randomiser will be implemented to ensure that the incoming signal is sufficiently random. This will be done by serially applying an exclusive-OR (XOR) of each bit in the codeblock with the pseudo-randomised sequence. The generation of the pseudo randomised sequence is given in [22]. Synchronisation of the fixed length LDPC codeblocks will be achieved by adding an attached synchronisation marker (ASM). Synchronisation is required for the correct decoding, processing of transfer frames and the synchronisation of the pseudo-randomizer. The ASM bit pattern is a 32 bit sequence shown in [22]. This bit pattern will not be randomised by the pseudo-randomiser. The encapsulation packet containing the ASM and randomised LDPC codeblock is called the CADU.

The synchronization and channel coding sublayer will be implemented as defined in the ECSS standard [22]. The main services provided by this sublayer are:

- provide required bit transition density
- frame synchronisation
- error detection
- error correction

A flow diagram of the synchronisation and channel coding sublayer is shown in Figure 3.8.

The synchronisation and channel coding sublayer will be implemented on a Xilinx Virtex-5 FPGA [38]. This sublayer operates between the TM protocol sublayer and the physical layer, as shown in Figure 2.8. The process operates as follows: on transmission, this sublayer will receive transfer frames from the TM protocol on the OBC. The transfer frames are then encoded with a cyclic redundancy check (CRC-16), added to the transfer frames. This will be sent to the FEC encoder. The FEC encoder encodes the data and sends the codeblocks to the pseudo-randomiser. The pseudo-randomiser will randomise the codeblocks and send the randomised codeblocks to the next block, which adds the ASM. This data are sent to the modulator, to be transmitted over the satellite link.

When data are received from the demodulator, the frame synchronisation block looks for the ASM, removes it, and sends the randomised codeblock to the pseudo-derandomiser. The derandomised codeblock is sent to the FEC decoder. After decoding, the decoded codeblock will be checked with the CRC code and the transfer frame sent to the TM protocol routine on the OBC.

Error detection

The purpose of the CRC code is to ensure that no transfer frames with errors are sent to the TM protocol. The CRC code to be implemented will have a redundancy length of 16. All transfer frames are appended with a 16 bit

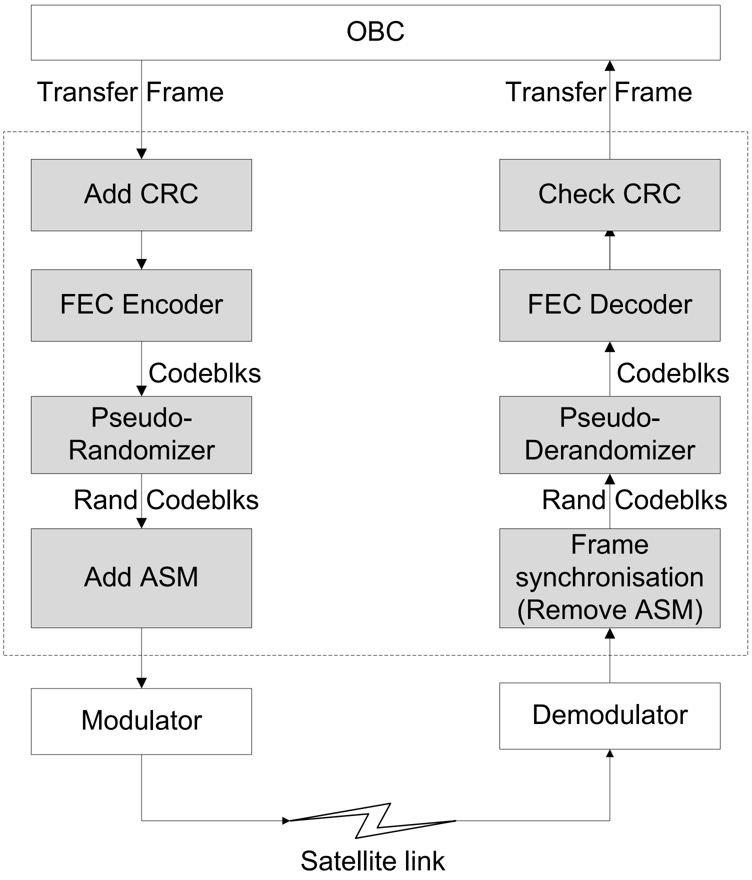


Figure 3.8: Channel coding and synchronization sublayer

CRC code. This will give a symmetric block code (N,N-16). The generator polynomial of the CRC code is given by:

$$G(X) = X^{16} + X^{12} + X^5 + 1 \tag{3.3.1}$$

The data link layer is responsible for dropping all erroneous packets and sending correct transfer frames to the TM protocol routine. The ARQ protocol is responsible for flow control and retransmission services, to provide end to end reliability.

Forward error correction

LDPC codes will be implemented as forward error correction to achieve an effective coding gain over the noisy channel.

Interleaving can also be implemented to increase the throughput of the system in the decoder. This allows the decoder to work separately on two different blocks of data by separating the check nodes from the variable nodes in the decoder [25]. This can increase the throughput of the decoder. The use of an interleaver will at least require double the memory usage compared to a system without interleaving, and increase the area required on an FPGA for parallel implementation. With the requirement of developing low cost ground stations and with limited resources on the space segment, interleaving will not be implemented in this design.

Pseudo-Randomiser

A pseudo-randomiser should be used to ensure that the message sent by the modulator have sufficient bit transitions. The randomiser is implemented as specified in [22]. Each bit in every codeword received from the FEC encoder is XORed with the corresponding bit in the pseudo-randomiser sequence. The polynomial used to generate the pseudo-random sequence is given by:

$$P(x) = x^8 + x^7 + x^5 + x^3 + 1. \tag{3.3.2}$$

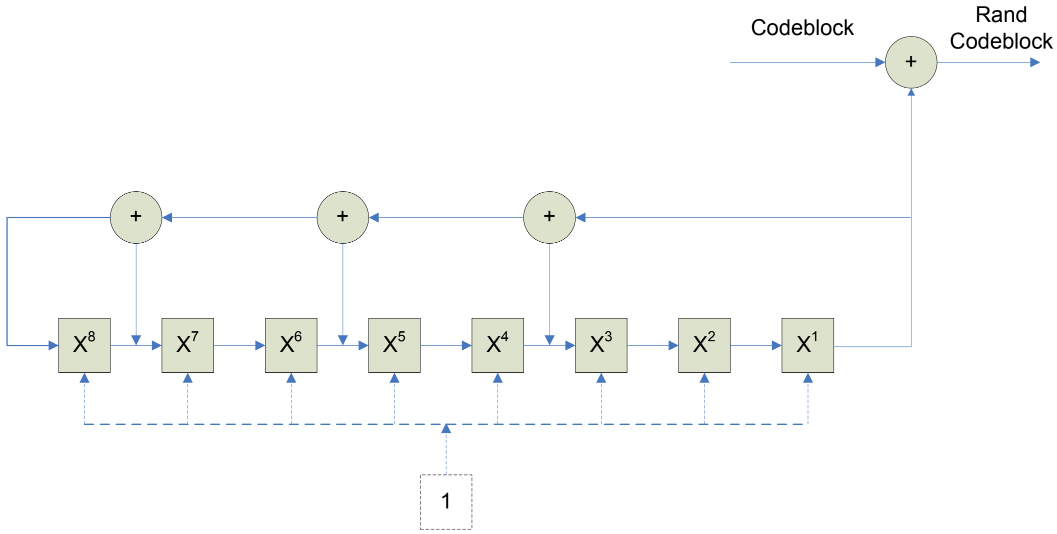


Figure 3.9: Pseudo-randomizer

In Figure 3.9 each circle represents a modulo 2 adder (XOR), and each square represents a single bit delay. At the start of each new codeblock, all the values in the shift register are reset to represent the value '1'. Every bit from the codeblock is XORed with the result from the pseudo-randomiser sequence. The randomised sequence repeats itself after 255 bits. Note that the ASM discussed in the next section will not be randomized by the pseudo-randomiser, because the sequence for the ASM is already optimised to ensure that there are enough bit transitions. At the receiver the same sequence is applied to the randomised codeblock. The register is also reset to the all 1's states before each codeblock.

Attached Synchronisation Marker

Codeblock synchronisation is essential for any decoding where block codes are used. Furthermore, the ASM is also used to synchronise the pseudo-randomiser, explained in the previous section. At the sender, the ASM is simply a specific bit pattern that is appended to each randomised codeblock. At the receiver, the ASM is detected and the recognised codeblock is sent to the de-randomiser, after which it will be decoded. In this implementation, the ASM will be detected in the channel domain. This implies that it will be detected before decoding starts, as required by the LDPC decoder.

The ASM is a four octet (32-bit) marker and is described in hexadecimal and binary by:

$$\begin{array}{lcccccccc} ASM & 1 & A & C & F & F & C & 1 & D \end{array} \tag{3.3.3}$$

$$\begin{array}{cccccccc} 0001 & 1010 & 1100 & 1111 & 1111 & 1100 & 0001 & 1101 \end{array} \tag{3.3.4}$$

3.3.2 Telemetry transfer frame protocol design

On the data link protocol sublayer the TM transfer frame protocol will be implemented. The TM space data link protocol provides functions to send fixed length protocol data units, called transfer frames, over a satellite link as specified in [22]. The most important service of the TM protocol is to enable the transfer of variable length packets received from the transport layer, implementing an ARQ protocol. Each frame contains a header with control information and a fixed length data field containing ARQ packets received from the transport layer. The TM protocol is responsible for packet processing, including fragmentation and assembly of packets. The TM protocol standard, designed by the ECSS, provides a lot of optional services to the user, and the implemented services thereof will be explained in this section. It is important to note that the data link layer does not provide end-to-end reliability, but all frames passed to the transport layer can be accepted as frames without error. The TM protocol also provides services through the use of master and virtual channels to enable multiplexing for systems with more than one source. In this application the TM protocol will only receive data from a single source, and therefore the virtual channels will not be implemented, but will be available for future implementations.

TM Frame Structure

The TM transfer frame format is shown in Figure 3.10. The various fields will be briefly explained. The primary header is a 6 octet field, containing information such as the spacecraft identifier and other control flags related to the TM frame structure. The transfer frame secondary header is an optional field used to carry fixed length mission specific data. The transfer frame data field will consist of the data received by the network layer on transmission. The transfer frame trailer is an optional field used for error detection when not present in the synchronization and channel coding sublayer.

This implementation will only consist of the transfer frame primary header and transfer frame data field. The optional transfer frame secondary header will not be used. The main purpose of the transfer frame secondary header is to carry additional data, enabling communication between higher layers of the protocol stack. The optional transfer frame trailer will also be omitted and its operational control field is used to specify different type reports. The error control field, used to do error detection, is already included in the synchronisation and channel coding sublayer, as specified in Section 3.3.1, and will not be implemented in the TM frame protocol.

Transfer Frame Primary Header	Transfer Frame Secondary Header (optional)			Transfer Frame Data Field	Transfer Frame Trailer (optional)	
	Transfer Frame Secondary Header ID		Transfer Frame Secondary Header Data Field		Operational Control Field (optional)	Frame Error Control Field (optional)
	Transfer Frame Secondary Header Version Number	Transfer Frame Secondary Header Length				
6 octets	2 bits	6 bits	up to 63 octets	variable	4 octets	2 octets

Figure 3.10: TM Frame

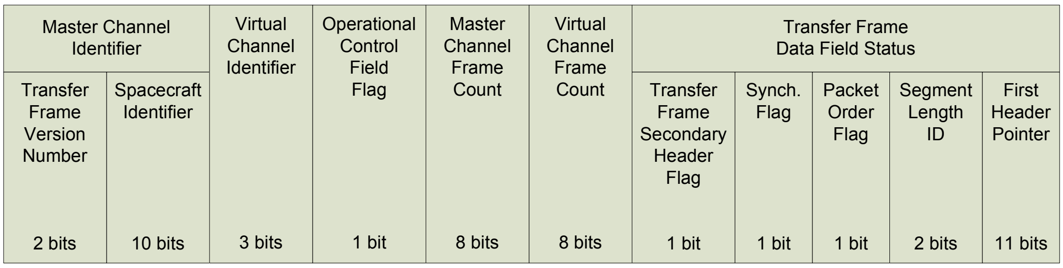
Transfer Frame Primary Header

The transfer frame's primary header is shown in Figure 3.11. The implementation and services of all the fields will now be discussed.

Master Channel Identifier

The master channel identifier consists of the transfer frame version number (TFVN) and spacecraft identifier (SCID). The TFVN shall always be set to '00'. This value was fixed by the CCSDS for all TM transfer frames.

The spacecraft identifier is a unique number assigned to the spacecraft. This number will be used as a source address. With permission of the CCSDS [33], any number can be used as a spacecraft identifier during pre-launch tests



Master Channel Identifier		Virtual Channel Identifier	Operational Control Field Flag	Master Channel Frame Count	Virtual Channel Frame Count	Transfer Frame Data Field Status				
Transfer Frame Version Number	Spacecraft Identifier					Transfer Frame Secondary Header Flag	Synch. Flag	Packet Order Flag	Segment Length ID	First Header Pointer
2 bits	10 bits	3 bits	1 bit	8 bits	8 bits	1 bit	1 bit	1 bit	2 bits	11 bits

Figure 3.11: Transfer Frame Primary Header

and simulations. For use in space missions, a unique identifier will be assigned by the CCSDS as defined in [39].

Virtual Channel Identifier

The virtual channel identifier will always be set to the value '001'. There is only one source for the IS-HS II implementation, and therefore no need to distinguish between different virtual channels.

Operational Control Field Flag

The operational control field flag will indicate whether the operational control field will be present. This field will always be absent in this particular implementation and therefore, the value will always be set to '0'.

Master Channel Frame Count

The master channel frame count will function as a running counter for the frames received on the channel. This is a counter running from 0 to 255, after which it will be reset to the value 0. This counter will be used to assemble ARQ packets for the transport layer. Any gap in a received ARQ packet will lead to the packet being dropped. Note that the running counter will not be reset to zero for each new ARQ packet.

Virtual Channel Frame Count

The virtual channel frame count will have the same value as the master channel frame count and will not be checked or used in the receiver.

Transfer Frame Data Field Status

The transfer frame secondary header flag will indicate whether a transfer frame secondary header field will be included in the frame. In this implementation a transfer frame secondary header will not be included, and the value will always be set to '0'.

When the synchronization flag is set to '0' it will indicate that the frame contains a new ARQ packet, or idle data is inserted. When the value of the synchronization flag is set to '1', it indicates that no new packets start in the frame, meaning that a fraction of an ARQ packet spans across the whole frame, as explained in the next section.

The packet order flag is always set to '0', and reserved for future use. The segment length identifier was used in earlier standards to support an optional feature that is not used any more. However, this field is still present in current specifications. Due to the fact that we don't want to do any packet inspection of higher layer protocols (ARQ) to indicate the length of packets, this last bit of the segment length identifier will be used as follows: the first header pointer will indicate if a new ARQ packet starts in the transfer frame. Due to the fact that ARQ packets will always be larger in size than the transfer frame, we know that only one new packet can start in a transfer frame. The last bit of the segment length identifier will be used to indicate whether the new packet is an ARQ packet or an idle packet used to fill up a transfer frame, as explained in the next section. The segment length identifier will be set to '11' if the new packet is an ARQ packet, and '10' if it is idle data used to fill a transfer frame.

The first header pointer will indicate the first octet of the packet beginning in the frame, if the synchronisation flag is set to '0'. If no new packets start in the frame, the value of the first header pointer will be set to '11111111111'. If it is required, for synchronisation, that an idle packet be sent, the value of the first header pointer will be set to '11111111110'. Idle packets will be dropped by the TM protocol.

Packet Processing

The packet processing functions will be explained by referring to Figure 3.12. At the transmitter, the frame construction and segmentation function will receive variable length ARQ packets. The frame header for each TM frame will be constructed as discussed. As described in Section 5.4.3.1 of the TM transfer frame specifications ECSS [33], "*The functions for processing the packets depend on knowledge of the position, size and meaning of certain fields in the standard packet headers*". In order to provide a modular system with interdependence between protocol layers, it is not desirable to do packet inspection of higher layer protocols at the data link layer. The following method will be implemented to avoid packet inspection:

Note that the packet length of the ARQ packet will be larger than the data field length of a TM frame. Using this property in our implementation, it is known that only one new packet will start in any TM frame. Figure 3.12 illustrates this graphically. Exceptions of smaller packets will be handled by the design of the encoder as discussed later in this section. In Figure 3.12, two ARQ packets are sent to the TM protocol to be transmitted. *Packet1* is larger than *Packet2* and will span across two whole TM frames as well as a part of the third frame. The first header pointer is part of the TM packet header and used to indicate the start of a new packet. The first header pointer in the first frame indicates that a new packet starts in the beginning of the data field. The value of the first header pointer is set to '11111111111' in the second frame, indicating that no new ARQ packets start in this frame. In the third frame, the first header pointer indicates the offset at which the new ARQ packet starts within the data field. The packet processing function will indicate that it is the start of a new ARQ packet by setting the segment length identifier to '11'. The fourth frame is similar to the second one with no new packets starting in the frame. In the last frame, the first header pointer indicates the end of the ARQ packet and indicates that idle data (fill bits) follow the packet by setting the segment length identifier to the value '10'. To cater for the exception that a packet is smaller than a TM frame, the value of the segment length identifier will be set to '00'. This will indicate that a higher layer packet smaller than a TM frame is present, and the first header pointer will indicate the end of the packet (beginning of idle data). This scenario is not shown in Figure 3.12, due to the fact that the ARQ packets will be very large in comparison with the implemented TM frames.

Because only one new packet or idle data (fill bits) will start in a transfer frame, the only information required by the receiver is whether it is a packet or idle data. This avoids the necessity for packet inspection of higher layer protocols.

The encoder of the system will be designed to check the remainder of a TM frame when starting a new ARQ packet in a TM frame. This will be done to avoid a scenario where a new packet will start and finish in a TM frame, following the trailer of another TM frame.

At the receiver the information in the primary header will be used to unpack the frames and build the ARQ packets. ARQ packets will be rebuilt by using the master channel sequence numbers. Any gaps in an ARQ packet will be detected. If a TM frame was received in error or not at all, the TM protocol will not be able to reconstruct the ARQ packet correctly and these erroneous or incomplete packets will be dropped. Information of invalid ARQ packets will be available at the data link layer, but will not be sent to the higher layer protocol in our application. The ARQ protocol's timers will be used to resend unacknowledged packets. Note that the ARQ protocol provides end-to-end reliability. The data link layer ensures that all ARQ packets sent to the higher layer is error free and complete. The synchronization and channel coding sublayer guarantees error free data and the TM protocol ensures that all ARQ packets are complete (without gaps).

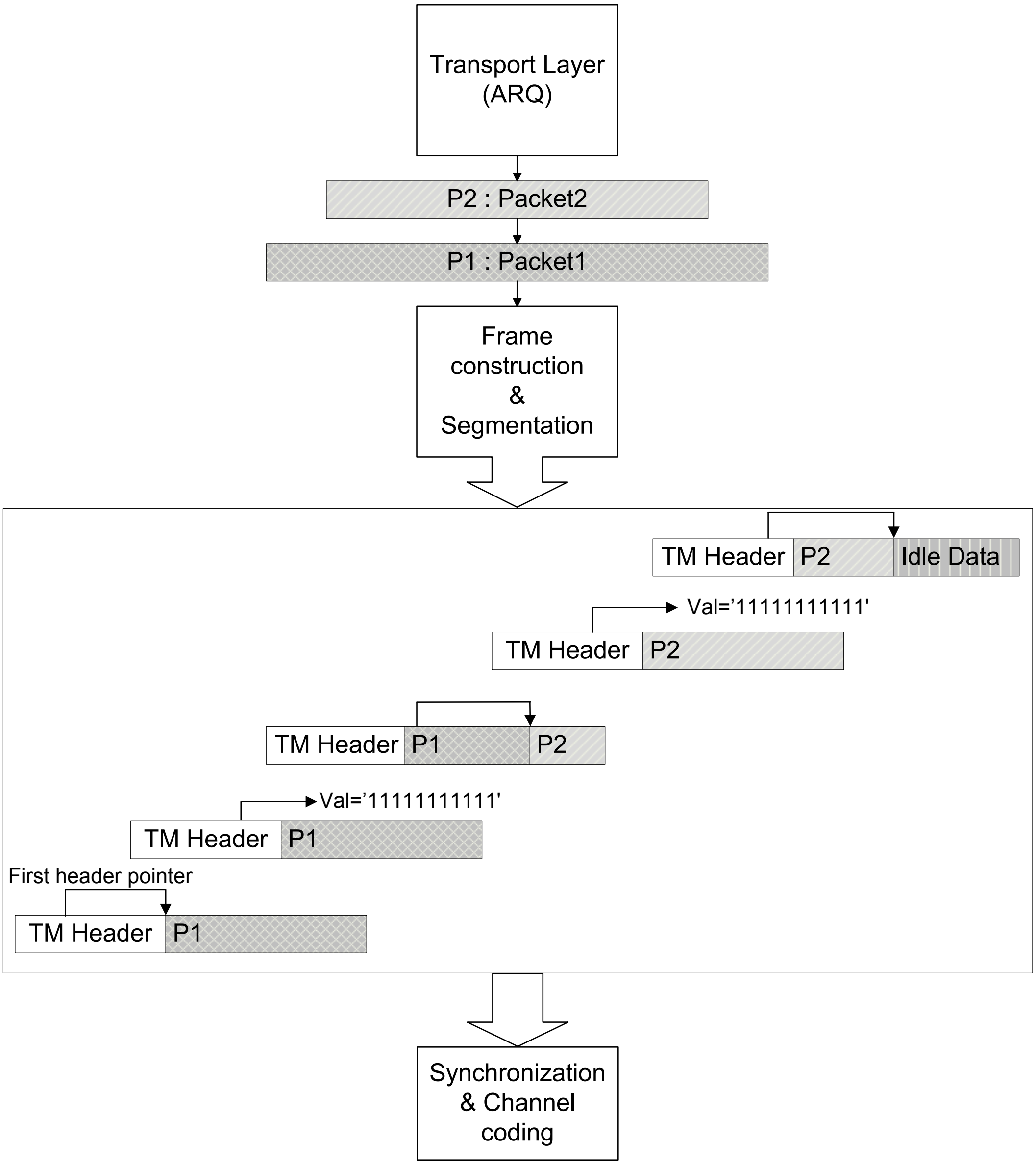


Figure 3.12: TM packet processing

3.4 Design cycle and design tools

The synchronisation and channel coding sublayer will be implemented on a Virtex-5 XC5VLX50-1FFG676 FPGA [40]. The design tools to be used are MATLAB®, AccelDSP™, System Generator for digital signal processing (DSP) and ISE Foundation™[1]. The basic functionality and data handling methods will be designed in Simulink®[2] with the use of System Generator for DSP. This

[1] AccelDSP, System Generator and Xilinx ISE are all Xilinx™products. http://xilinx.com

[2] MATLAB®and is Simulink®are Mathworks™products. http://mathworks.com

allows for a shorter design cycle by enabling the designer to test the functionality of the system with simple sinks and sources in Simulink, without the need to design testbenches in HDL for all modules. The attached synchronisation marker and pseudo randomiser will also be implemented in Simulink with System Generator for DSP. The FEC encoder and decoder are to be implemented through the use of AccelDSP. AccelDSP allows the designer to work with a reduced instruction set of MATLAB functions to design blocks used in System Generator, as shown in Figure 3.13. Synthesizable MATLAB code will be developed to describe the LDPC encoder and decoder. This will be converted to HDL code by AccelDSP and used as a custom functional block in Simulink. The complete data link layer will be developed in Simulink and converted to a VHDL implementation to be used on the FPGA through System Generator. The final step will be implemented in ISE Foundation, to configure the user constraint file and for final testing. The encoder and decoder will be defined in AccelDSP with some control logic to be developed in System Generator. The CRC coder and decoder will be implemented in System Generator.

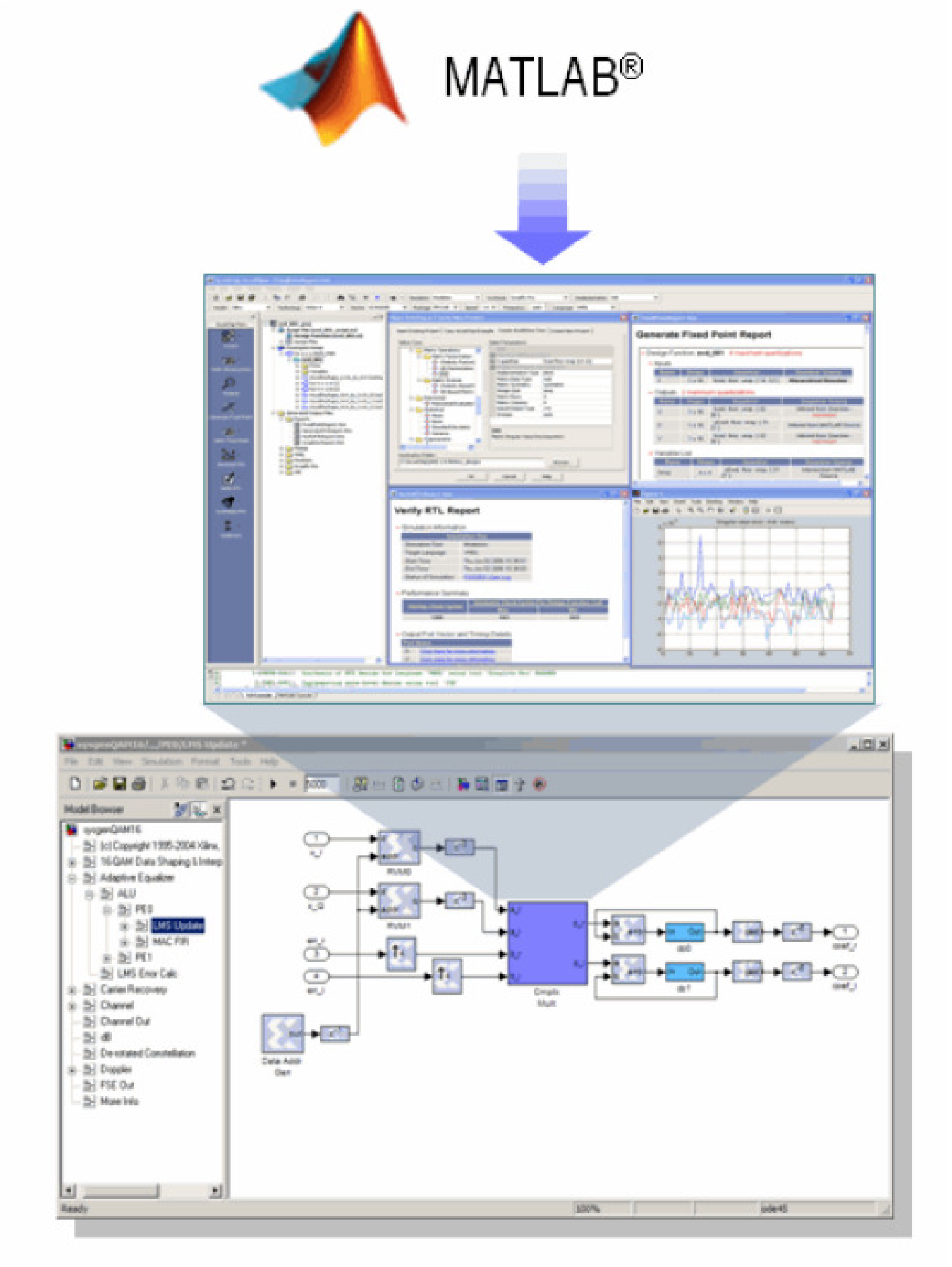


Figure 3.13: Design cycle

3.5 Simulator

The simulator was written in MATLAB to simulate a communication channel over a noisy wireless link. The simulator will be explained by referring to the flow diagram in Figure 3.14.

The parity check matrix **H** used, is given as input by the user to the simulator. Another input the user must provide is the decoder to be used. Decoder options for this implementation includes:

1. Belief propagation decoder
2. Log likelihood BP decoder
3. Min-sum approximation decoder
4. Normalised min-sum approximation decoder

By evaluating the size of **H**, the code rate can be determined. The user selects the range of E_b/N_o values that will be simulated. In Figure 3.14 $EbNo[0..max]$ represents the range of E_b/N_o values to be simulated, and $index$ refers to the current element within this set. The range doesn't need to start at 0 dB, but was chosen to simplify the understanding of the flow diagram. The simulation stopping criterion is also given as input to the simulator. $BLK_{min}[index]$ determines the number of block errors that must be reached for the corresponding $EbNo[index]$ value, before the simulation stops. The next step is to either read a systematic generator matrix **G**, or generate one. A systematic generator matrix can be calculated for a sparse **H**, by using Gaussian elimination if the first K rows of **H** are linearly independent. By truncating the linear dependent rows from **H**, a systematic generator matrix can be determined for a code of rate $(N - M)/N$, which has a slightly higher rate. For a parity check matrix **H** with no linear dependent rows, a systematic generator matrix **G** can be calculated for **H** or a dual code of **H** [15]. In this simulation a systematic generator matrix was determined by using a MATLAB script file written by T.K. Moon[3], using Gaussian elimination.

The use of regular codes are important, as its structure simplifies the the method of propagating through the matrix. Reading **H**, and indexing the non-zero elements are essential to write an efficient decoder. Two textfiles are created; H_Row represents a $K \times 6$ matrix, indicating the six column values in each row, where **H** has non-zero elements. H_Column represents a $N \times 3$ matrix, indicating the three non-zero elements in each row of **H**.

[3] Used with permission of author, T.K.Moon

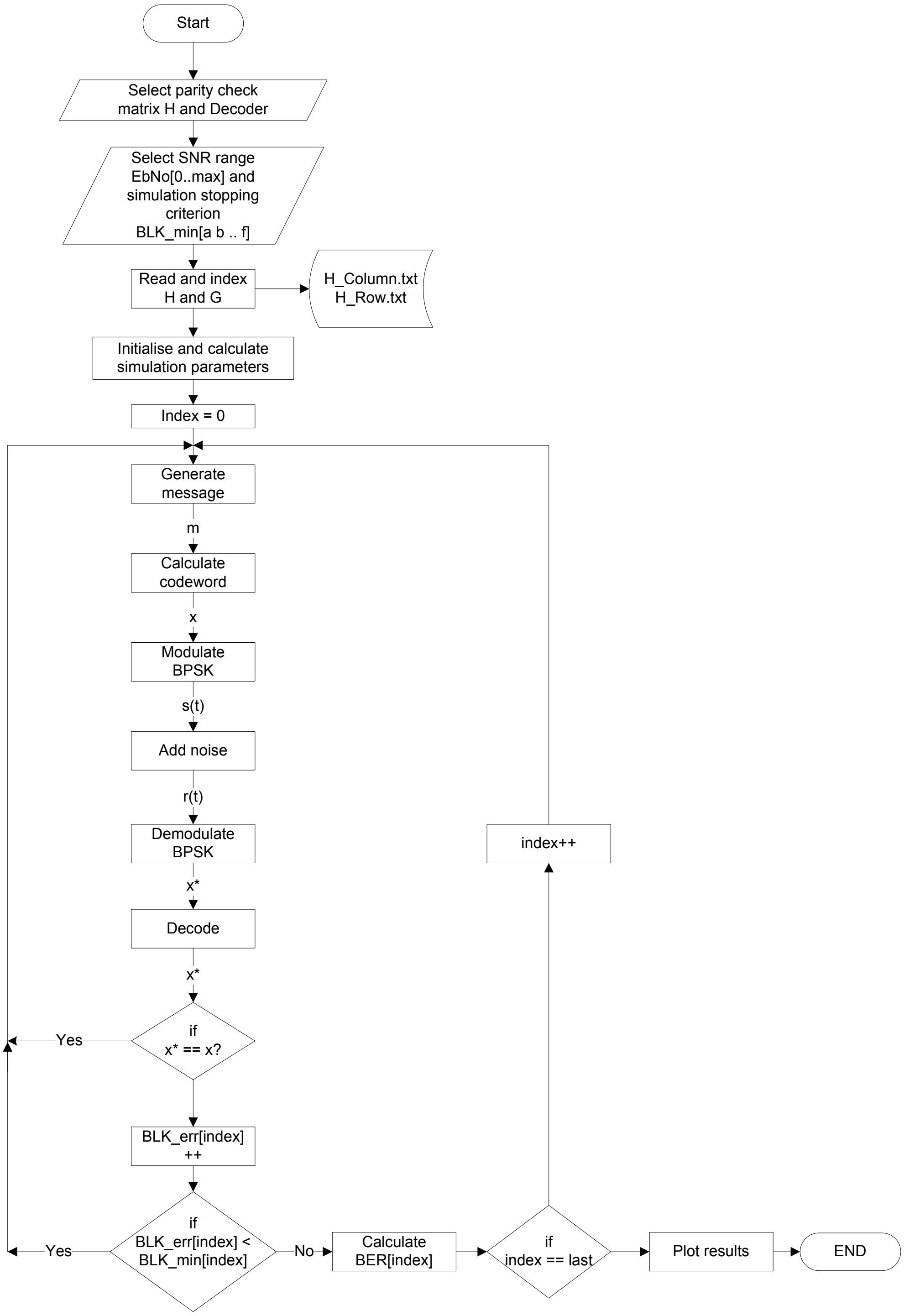


Figure 3.14: Flow diagram of simulator

In the next block, the simulation parameters are set up. This includes calculating the transmitted amplitudes for the binary phase shift keying (BPSK) modulation for each E_b/N_o value, taking the code rate into account. The simulation can be conducted by fixing the amplitude a of the modulated signal and calculating the noise σ, or fixing σ and calculating the amplitude a for a specific E_b/N_o value. In this simulation $\sigma = 1$. It was explained in Section 2.2.3 that the energy of each transmitted encoded bit is equal to the energy of a bit in an uncoded system multiplied by the code rate, giving

$$E_x = E_b \times R. \tag{3.5.1}$$

This can be written as $E_b = E_x/R$. For BPSK $E_b = a^2$ and $N_o = \sigma^2 \times 2$, where a represents the amplitude of the modulated signal. From this we have

$$\frac{E_b}{N_o} = \frac{a^2}{2 \times R \times \sigma^2} \tag{3.5.2}$$

where $\sigma = 1$. From (3.5.2), the amplitude of an encoded bit can be calculated for any E_b/N_o value.

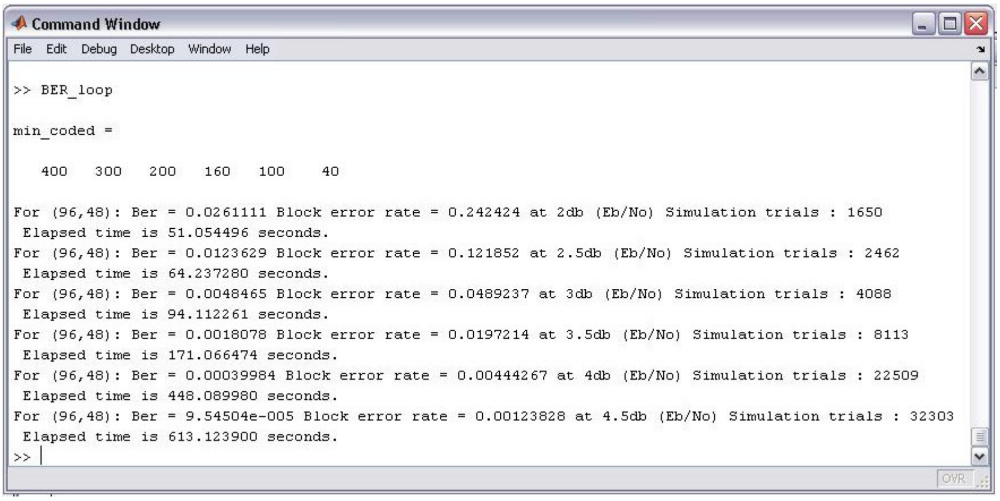
For each E_b/N_o value in the range specified, the amplitudes are calculated for BPSK modulation. In Figure 3.14, *index* is set to zero to start the simulation of the first E_b/N_o value.

A random binary message m of length K is generated. This message is encoded as explained in Section 3.2, giving a codeword x of length N. The codeword is BPSK modulated giving $s(t)$. White Gaussian noise is added to the transmitted signal to give the received signal $r(t) = s(t) + n(t)$. BPSK demodulation is performed and soft decisions are provided according to the specified quantisation. In the receiver an estimated message m^* is chosen and the value of m^* is compared to the initial message m. If the message was decoded correctly, the next message will be generated, as the simulation will only stop when reaching the specified number of block errors. If a block error is found, the BLK_err variable will be incremented. The new value of BLK_err will be compared to the minimum specified stopping criterion. If the minimum specified value has not been reached, another message will be generated, otherwise the bit error rate and block error rate for the specific E_b/N_o value will be calculated. Note that detected and undetected errors can also be determined. If there are more E_b/N_o values to be computed, the next simulation will be run. If this E_b/N_o value was the last of the set, the results for all E_b/N_o values are plotted, stored, and the simulation ends.

3.6 Data capturing and sources of error

The AWGN model was used in the simulations. This model provides a good estimation for satellite application. Using BPSK as modulation scheme, only real noise was added on the channel.

In Figure 3.15 a screenshot of the MATLAB workspace is shown to illustrate the data capturing procedure. In this simulation, the (96,48) LDPC code was used. The input *min_coded* (or *BLK_min* in Figure 3.14) can be seen to be $\{400, 300, 200, 160, 100, 40\}$. These values are used for the different E_b/N_o values, stopping when the corresponding number of block errors was found by simulation. It can be seen that the first simulation at 2 dB required 1650 trials to reach 400 block errors, while the last simulation at 4.5 dB required 32303 trials to reach 40 block errors. In order to ensure that the resultant BER plots are accurate, a 98% confidence interval was plotted on the BER plot for Monte Carlo simulations.



```
>> BER_loop

min_coded =

   400   300   200   160   100    40

For (96,48): Ber = 0.0261111 Block error rate = 0.242424 at 2db (Eb/No) Simulation trials : 1650
 Elapsed time is 51.054496 seconds.
For (96,48): Ber = 0.0123629 Block error rate = 0.121852 at 2.5db (Eb/No) Simulation trials : 2462
 Elapsed time is 64.237280 seconds.
For (96,48): Ber = 0.0048465 Block error rate = 0.0489237 at 3db (Eb/No) Simulation trials : 4088
 Elapsed time is 94.112261 seconds.
For (96,48): Ber = 0.0018078 Block error rate = 0.0197214 at 3.5db (Eb/No) Simulation trials : 8113
 Elapsed time is 171.066474 seconds.
For (96,48): Ber = 0.00039984 Block error rate = 0.00444267 at 4db (Eb/No) Simulation trials : 22509
 Elapsed time is 448.089980 seconds.
For (96,48): Ber = 9.54504e-005 Block error rate = 0.00123828 at 4.5db (Eb/No) Simulation trials : 32303
 Elapsed time is 613.123900 seconds.
>> |
```

Figure 3.15: Screenshot of Workspace in MATLAB for simulation of (96,48) LDPC code.

It can be seen from Figure 3.16 that in order to obtain good results for the high E_b/N_o values, many simulations have to be run. Many of the graphs in this thesis compare the results from a significant number of different codes, or different decoders. For this reason, the confidence intervals are not added. It should be noted that the intervals were used as a guideline to ensure that the simulations were sufficiently accurate. This example was only used to illustrate the concepts, and results given throughout the thesis have confidence intervals much smaller than portrayed in Figure 3.16.

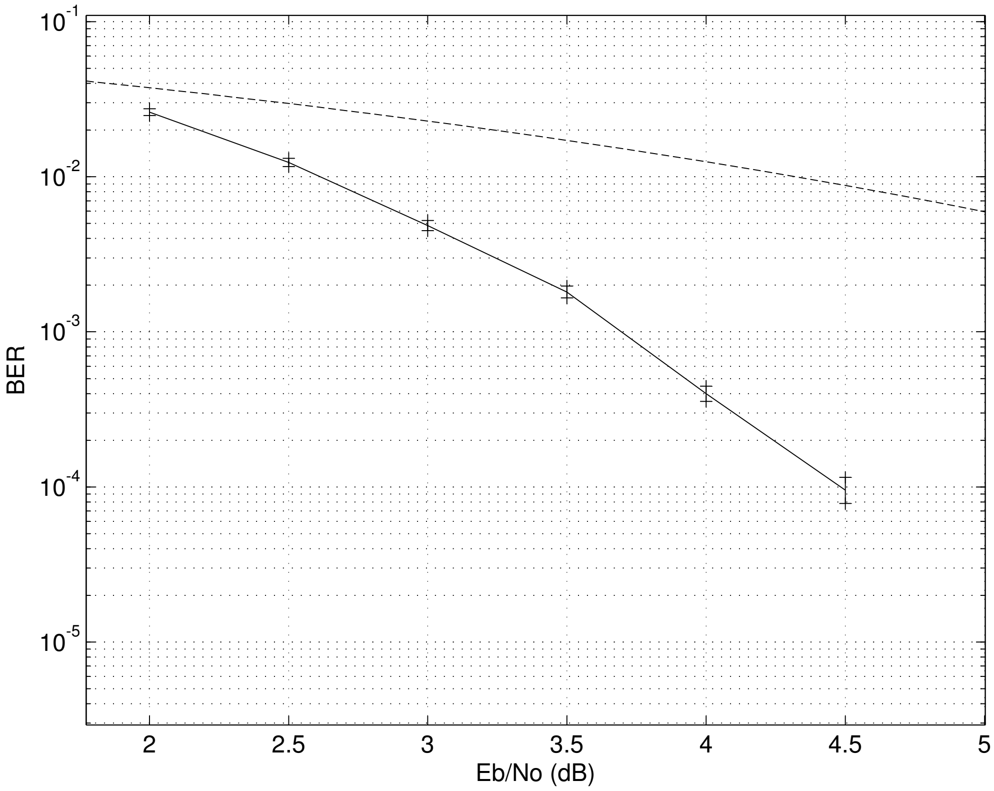


Figure 3.16: Performance of the (96,48) LDPC code showing a confidence interval of 98%. The dashed line represents an uncoded system.

Sources of error introduced in the simulation includes differences between the simulated channel and a real satellite channel, as well as quantisation. Some characteristics of a real satellite channel were, however, not simulated, including fading and Doppler shift inaccuracies. Quantisation includes both soft decision quantisation together with those related to hardware implementations and fixed point models. These will be explained in Chapter 4.

3.7 Conclusion

In this chapter it was shown that the proposed LDPC decoders hold much promise to be implemented within the TM synchronisation and channel coding sublayer. Some preliminary results for the BP decoder prove to be a great improvement over uncoded systems. The log likelihood BP decoder is shown to avoid many of the normalisation computations necessary in the standard BP decoder. The normalised min-sum decoder further reduces the computational complexity of both of these BP decoders. A detailed description of the min-sum approximation algorithm was given, while discussing some of its advantages in hardware implementations. Two strategies were shown to improve the performance of the standard min-sum approximation decoder.

Within the data link layer, a design was provided for a communication link functioning within the ECSS standards. A packet processing scheme was presented to avoid the necessity of packet inspection of higher layer protocols, giving a modular design.

The next chapter will focus on the performance results of the LDPC codes and discuss hardware implementation considerations.

Chapter 4

Results

4.1 Introduction

This chapter will focus on the results obtained by the different LDPC decoders introduced in Chapter 3. Error performances of the BP, min-sum and normalised min-sum decoders are presented and compared. Design choices are quantified by showing the performance effects of various parameters on the different decoders. These parameters include soft decision quantisation, maximum number of iterations and code length. The effect of the normalisation constant on the performance of the normalised min-sum decoder is shown for different values of α.

The second part of this chapter will cover the implementation of these codes in hardware and show fixed point results. The AccelDSP design cycle is presented and discussed for VHDL generation. Designs of the encoder and decoder for an LDPC code are presented. The effect of quantisation of the fixed point model is investigated. The chapter will be concluded with some remarks and findings on hardware implementations of LDPC codes.

4.2 Simulation results

In this section the performances of the different decoders will be compared to one another and to the uncoded system. Results were obtained by using the decoders, simulator and data capturing methods discussed in Chapter 3. The first result in Figure 4.1 shows the performance of the BP, min-sum and normalised min sum decoders. A (204,102) LDPC code was implemented with a maximum number of 100 iterations used in the decoding process, together with 3-bit soft decision decoding. Later in this section more detail will be given on the influence of the maximum number of iterations and soft decision quantisation on the performance of the codes. The normalised min-sum decoder was used with a normalisation constant set to $\alpha = 0.7$. It is shown that the BP algorithm provides the best coding performance, while the standard min-sum approximation algorithm gives results within 0.3 dB from the BP algorithm. As predicted, the normalised min-sum approximation, with $\alpha = 0.7$, outperforms the standard min-sum approximation algorithm and shows performances within 0.15 dB from the BP algorithm. At higher E_b/N_o values, the results of the min-sum approximation algorithms are even closer to that given by the BP algorithm.

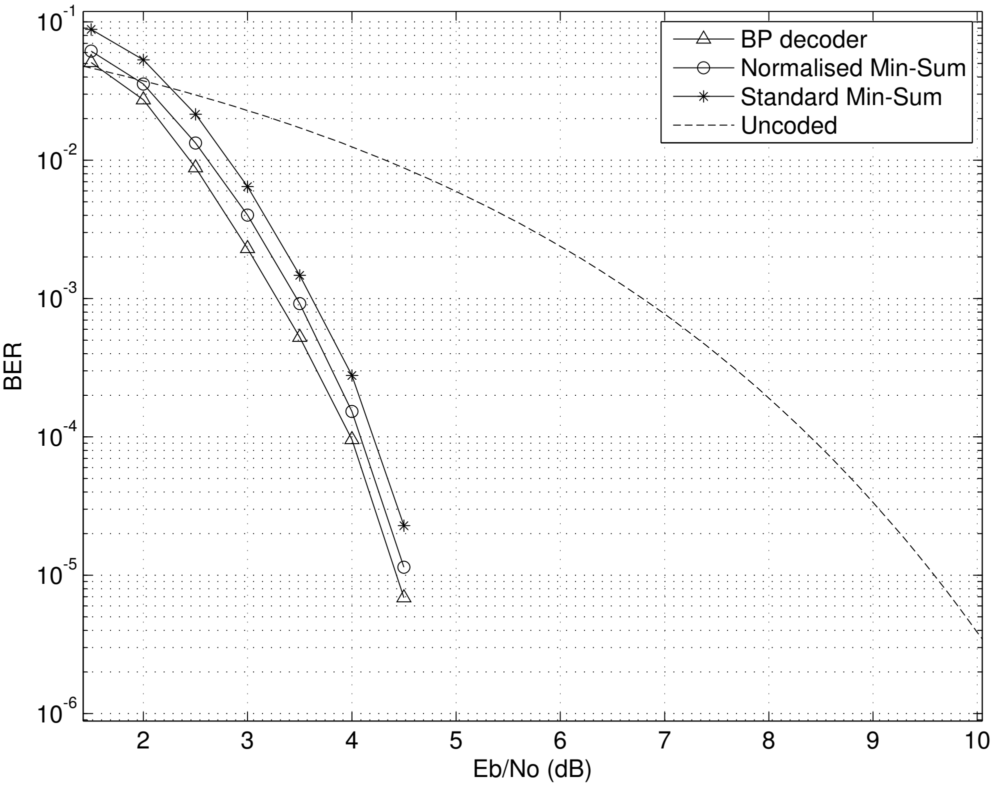


Figure 4.1: Performance of the BP, min-sum and normalised min-sum decoders. A (204,102) LDPC code was used with a maximum number of 100 iterations and 3-bit soft decision decoding.

It was shown in Section 3.2.4 that the normalisation constant α is used in the horizontal step of the normalised min-sum decoder. The check nodes $\mathbf{L}_{m,n}$ are multiplied by α. Figure 4.2 shows the performance of the normalised min-sum decoder with different values of α. A (204,102) LDPC code was utilised without quantisation of soft decisions. It can be seen that the values of 0.7 and 0.8 gives the best performances for the (204,102) code. The value of 0.7 was used in most of the LDPC codes considered in this thesis as it provides a good performance for all code lengths. With the value of 0.7, a consistent result is given over the whole E_b/N_o range considered. It should be noted that the value $\alpha = 1$ corresponds to the standard min-sum decoder without normalisation.

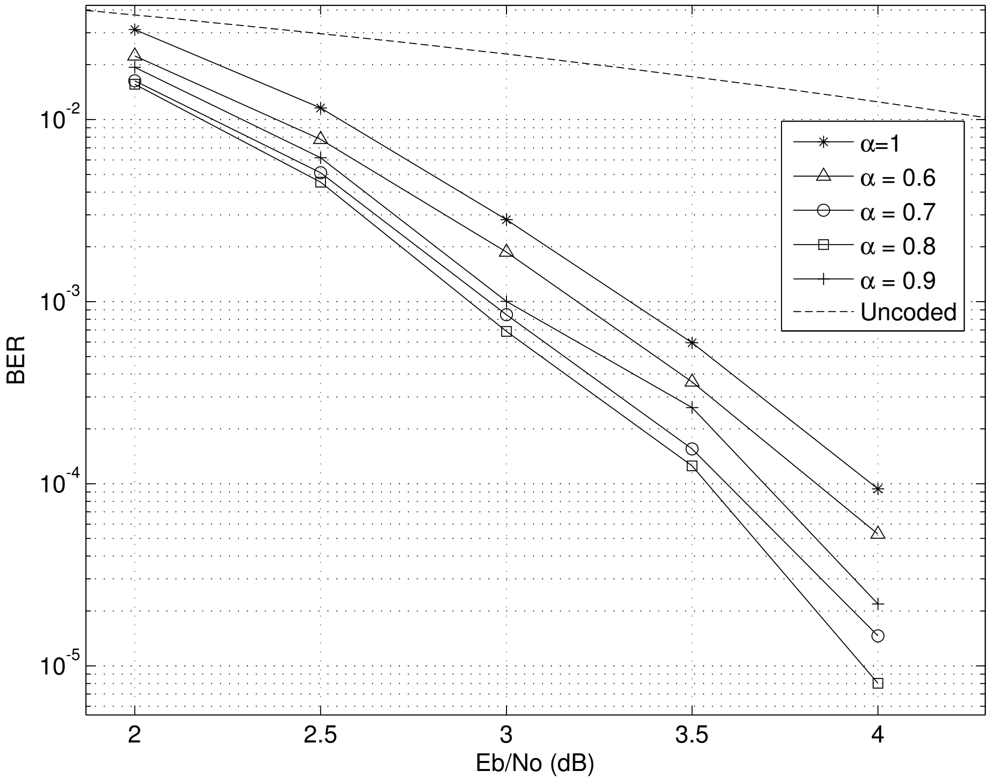


Figure 4.2: Performance of the normalised min-sum decoder for different values of α. A (204,102) LDPC code was used with a maximum number of 20 iterations and soft decision decoding with no quantisation.

In Figure 4.3 the results are shown for the normalised approximation algorithm, with different numbers of maximum iterations used by the algorithm. It can be seen that this algorithm has a fast conversion rate for the relatively low number of iterations used. Results for the maximum number of iterations set at 20, gives a performance within 0-0.05 dB of the algorithm with a maximum number of iterations of 100. When considering a low cost design, it is important to look at the number of iterations required by the algorithms to have good error performances. Reducing the maximum number of iterations will reduce the error performance, but will also decrease the latency of the decoder and reduce the processing requirements [35].

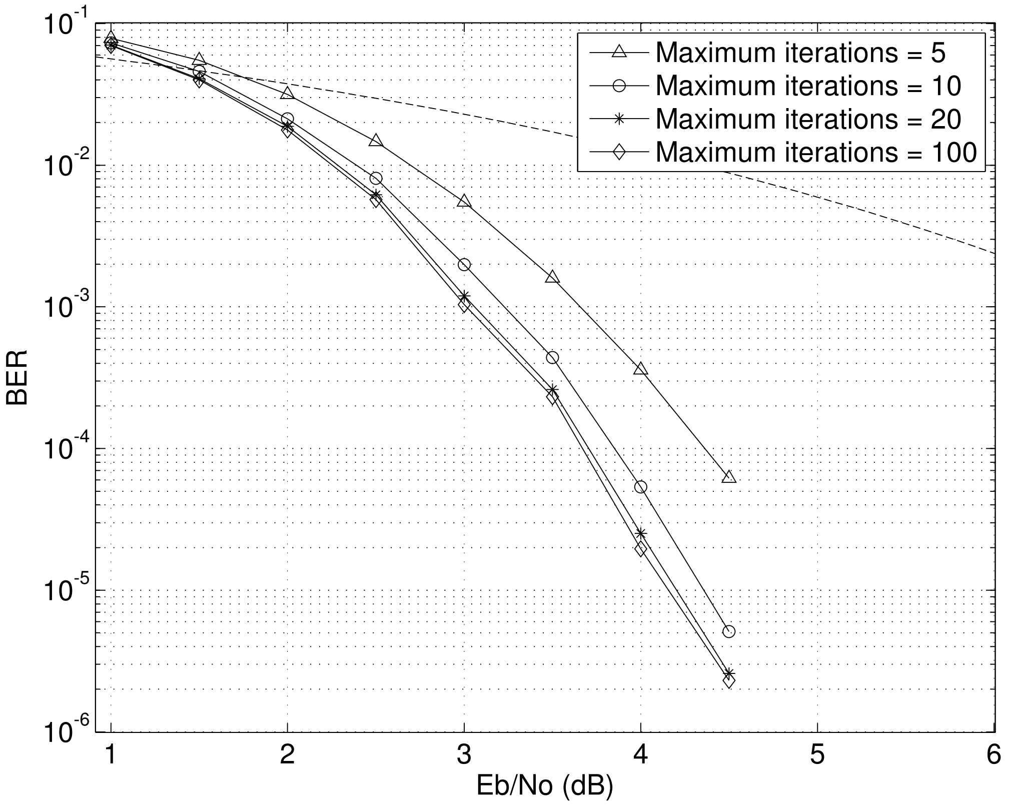


Figure 4.3: Performance of the normalised min-sum approximation algorithm on (204,102) code for different maximum iterations. A (204,102) LDPC code was used with a maximum number of 20 iterations and soft decision decoding with no quantisation.

The results for the different decoding algorithms with a maximum of 20 iterations are shown in Figure 4.4. For a smaller number of iterations, the normalised min-sum approximation algorithm shows good error performance. It should be noted that, unlike the BP algorithm, the performance of the normalised min-sum approximation algorithm compares favourably for a maximum number of iterations of 20 and 100 respectively. The normalised min-sum decoder converges quickly to the correct codeword. It can also be expected that the normalised min-sum approximation decoders outperform the BP algorithm at high E_b/N_o values for short length LDPC codes with a higher number of maximum iterations. This result as verified in Figure 4.9, will be discussed in detail later in this section.

Using this result, hardware implementations of the normalised min-sum decoder with a maximum number of 20 iterations, can be considered to give an efficient decoder with good error performance.

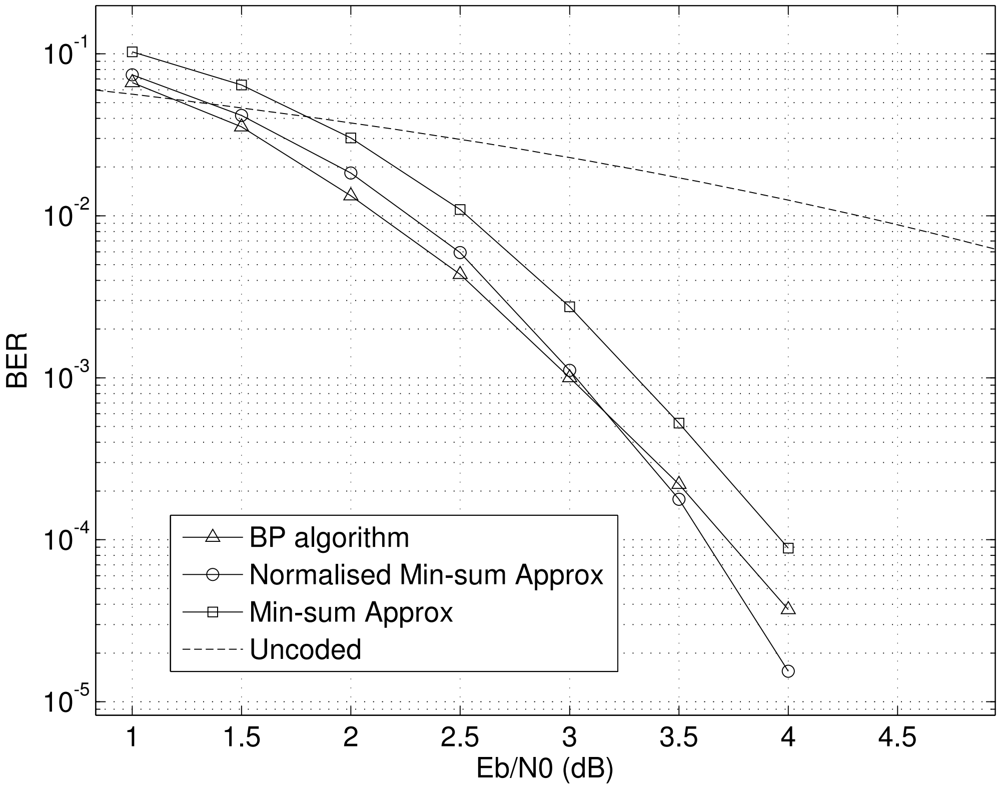


Figure 4.4: Performance of decoding algorithms on a (204,102) LDPC code with a maximum number of 20 iterations.

Figure 4.5 illustrates the average number of iterations used by the three decoders to find the correct codeword. These results used a maximum number of 100 iterations. For a maximum number of 100 iterations, the BP decoder outperforms the normalised and standard min-sum approximation decoders for the E_b/N_o range from 1-4 dB. This result is verified in Figure 4.1. The average number of iterations used by the BP decoder is also less than that of the other two decoders in the lower E_b/N_o range. The average number of iterations used by the normalised min-sum decoder at 4 dB, is less than that of the BP decoder. Even though it is not apparent from this graph, this result confirms the superior performance of the normalised min-sum decoder over the BP decoder for high E_b/N_o values when requiring fewer iterations.

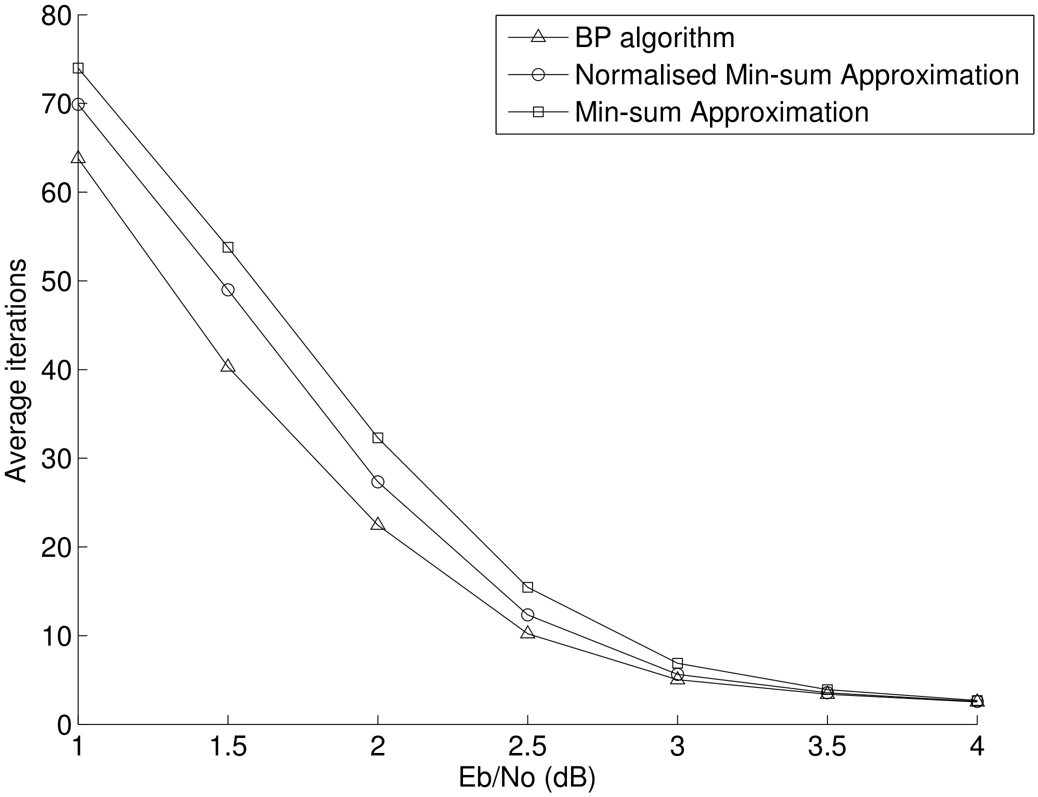


Figure 4.5: Average number of iterations used by the decoding algorithms on a (204,102) LDPC code.

Quantisation noise introduced into the system results in performance loss in the decoder. Soft decisions are sent to the decoder from the demodulator. In Figure 4.6 the performance for the BP decoder is shown for different levels of soft decision quantisation. The performance of the decoder with no quantisation performs approximately 1.8 dB better than the decoder implementing hard decision decoding. The decoder using 2-bit soft decision decoding already presents a large improvement in performance compared to the hard decision decoder. The 2-bit soft decision decoder has a performance loss of approximately 0.8 dB over the the decoder without quantisation. The 3-bit soft decision decoder reveals a performance loss of approximately 0.4 dB compared to the decoder using soft decisions without quantisation. The BP decoder can take any level of quantisation as input without alteration of the decoder.

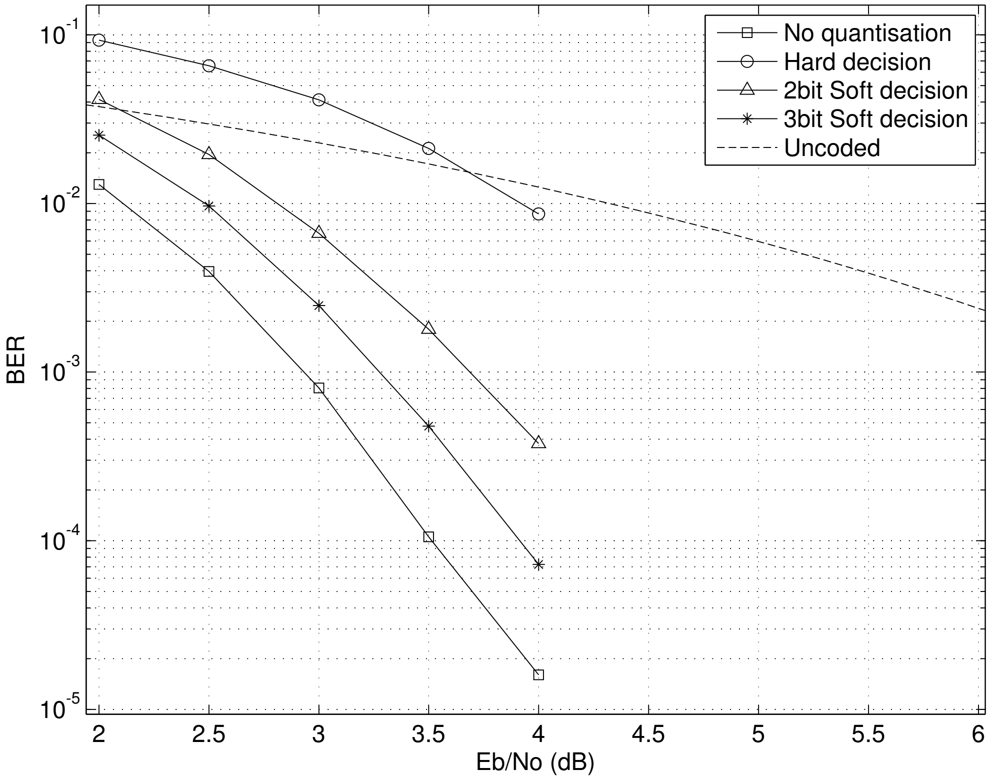


Figure 4.6: Performance of the (204,102) BP decoder illustrating the performance gain of soft decision quantised decoding over hard decision decoding. A maximum number of 100 iterations were implemented.

The block error rate of the decoder is also an important metric to evaluate the performance of a code. When implementing a forward error correction code in a communication system where it is critical that all packets have to be received without error, this implies that a packet with a single error will be dropped, and needs to be retransmitted. Considering the block error rate performance loss due to quantisation, it is important to ensure that random single errors introduced in various blocks don't decrease the overall throughput of the system beyond the expected loss shown for the BER plot in Figure 4.6. In Figure 4.7 the performance for the block error rates of the BP decoder is plotted. From this figure it can be seen that the performance loss in dB due to the quantisation levels is consistent to that evident in the BER plot of Figure 4.6.

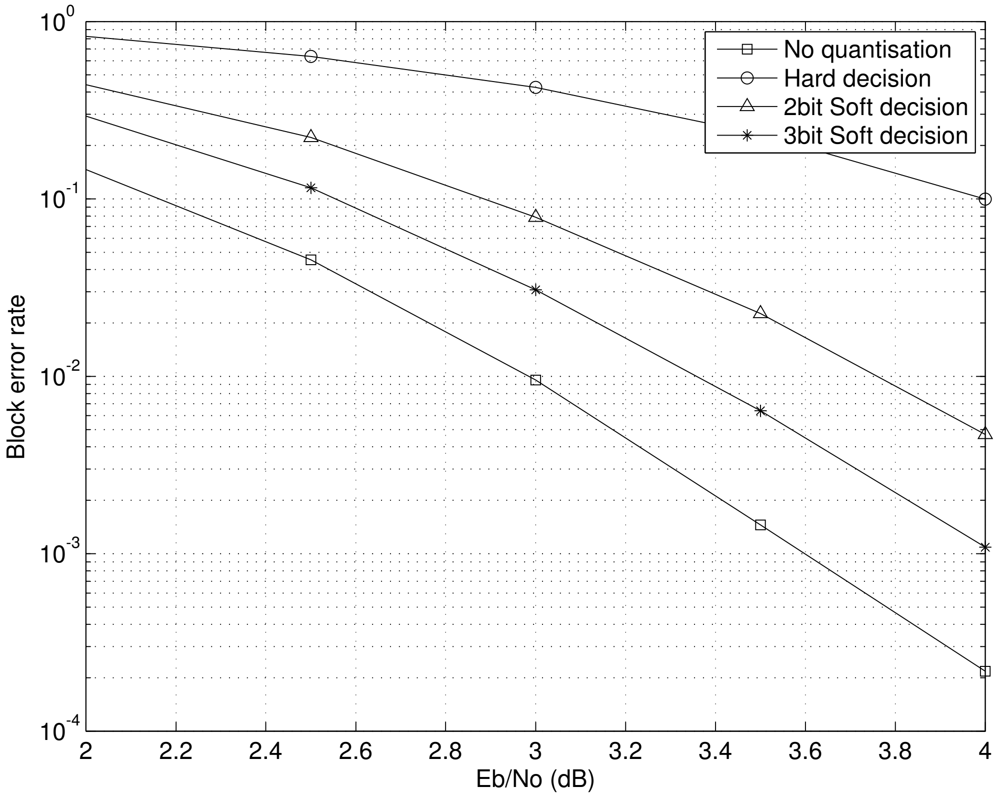


Figure 4.7: Performance of the (204,102) BP decoder illustrating the performance gain of soft decision quantised decoding over hard decision decoding. Maximum number of 100 iterations used in decoding. Performance given as block error rates.

In Figure 4.8 the performance of the normalised min-sum decoder is plotted for different levels of soft decision quantisation. A maximum number of 20 iterations was used in this simulation. The hard decision decoder shows a performance loss of approximately 1.7 dB compared to the decoder with no quantisation. The 2-bit soft decision has a performance loss of approximately 0.7 dB relative to that of the decoder without quantisation. The 3-bit soft decision decoder shows a performance loss of approximately 0.4 dB to the decoder without quantisation. These results are similar to those found for the BP decoder in Figure 4.6.

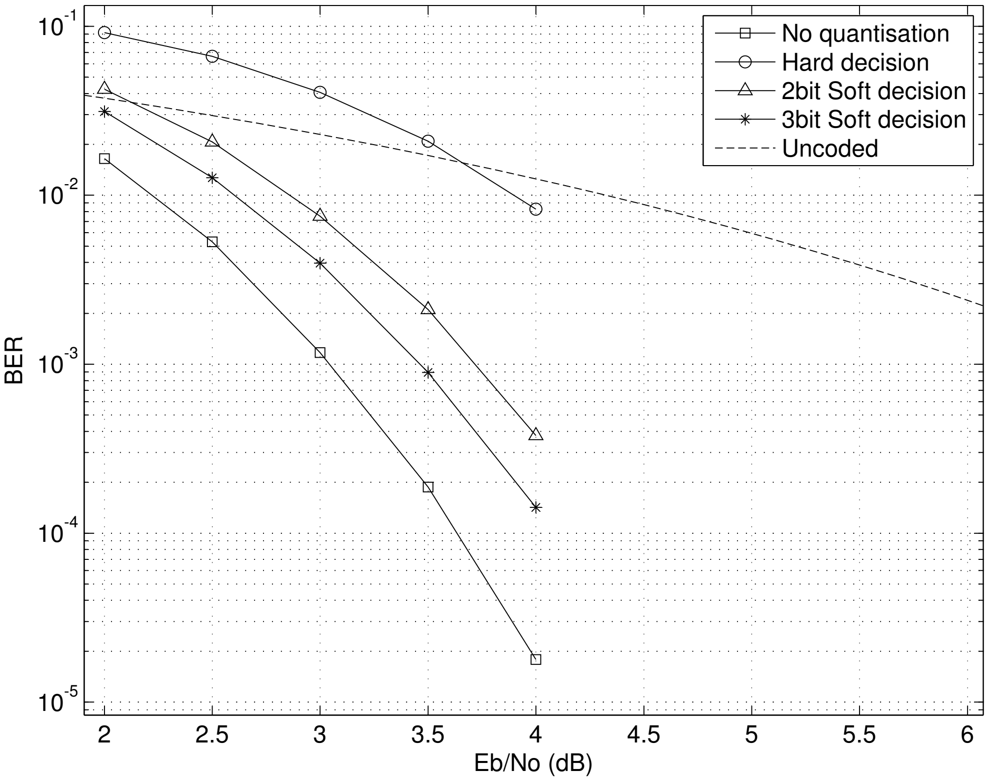


Figure 4.8: Performance of the normalised min-sum decoder for different levels of quantisation for soft decision decoding. A (204,102) LDPC code was applied with a maximum number of 20 iterations.

In Figure 4.9 the the BER and block error rates of the BP and normalised min-sum decoders are shown. For the decoders a (204,102) LDPC code with 3-bit soft decision decoding was utilised with 100 iterations. In this simulation more than 600 000 codewords were simulated. In the range of 2-4 dB the BP decoder outperforms the normalised min-sum decoder by 0-0.15 dB. At 4 dB the two decoders give similar performance. At higher E_b/N_o values, the normalised min-sum decoder performs better than the BP algorithm. This is due to the fact that the BP algorithm is not a maximum likelihood decoder. The performance loss of the BP algorithm is due to correlations among the messages passed between the variable nodes and check nodes. These correlations are found in short length LDPC codes. The approximation algorithm reduces the effect of these correlations, and can therefore outperform the BP algorithm for codes with short block lengths at high E_b/N_o values. These results have been verified in [36]. As the block lengths of the codes increase, the regular code's structure behaves more like those of random codes, eliminating the correlations in the code structure. The approximation min-sum algorithm is not expected to outperform the BP algorithm at high E_b/N_o values for code lengths larger than approximately 4 000.

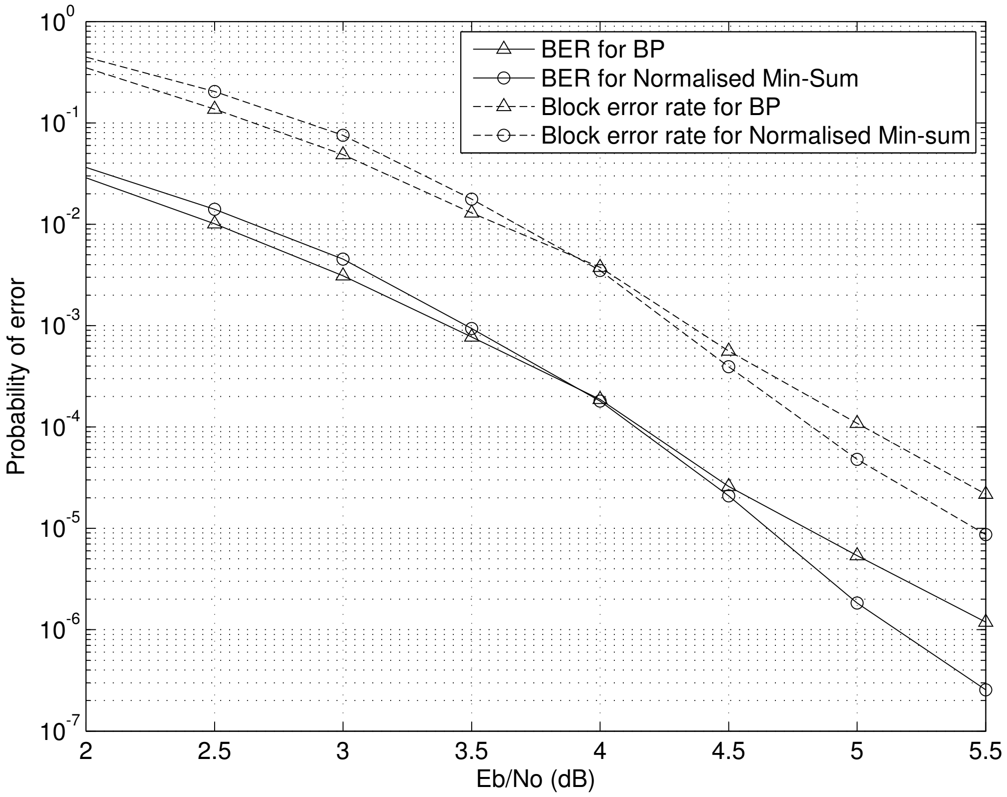


Figure 4.9: Performance of the BP and min-sum approximation decoders on a (204,102) LDPC code. Bit error rate and block error rate are shown for a maximum number of 100 iterations and 3-bit soft decision decoding.

Increasing the code length of the code increases the performance of the code. These results were discussed and presented in Section 3.2.2. Figure 4.10 verifies these results for the normalised min-sum decoder with a maximum number of 20 iterations and 3-bit soft decision decoding. It is clear that the (408,204) LDPC code exhibits a better error performance than that of the (204,102) LDPC code.

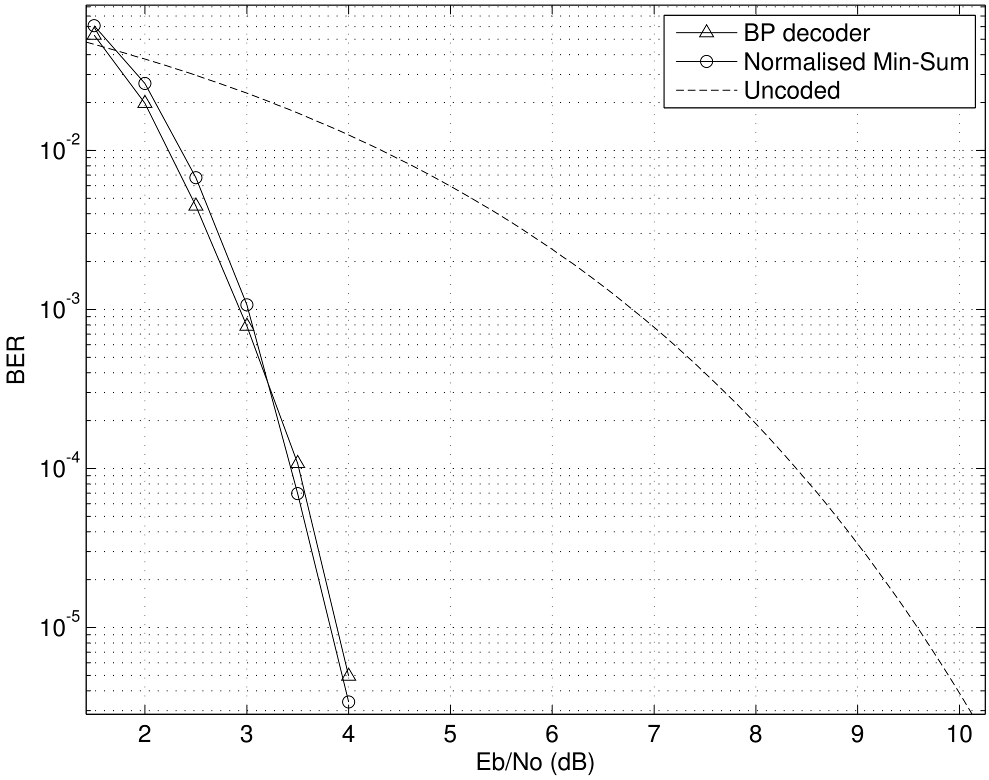


Figure 4.10: Performance of the BP and normalised min-sum approximation decoders on a (408,204) LDPC code. A maximum number of 20 iterations and 3-bit soft decision decoding were utilised.

The implemented short length LDPC codes are compared to well known Reed-Solomon and convolutional codes. Results for the LDPC codes were obtained by utilising a BP decoder with a maximum number of 100 iterations and no soft decision quantisation. These results are shown in Table 4.1. The (255,223) Reed-Solomon code requires 6.23 dB to achieve a BER of 10^{-5}

Table 4.1: Required E_b/N_o (dB) to obtain a BER of 10^{-5}

RS	Convolution	LDPC(96)	LDPC(204)	LDPC(408)	LDPC(1008)
6.23dB	4.15dB	4.98dB	4.14dB	3.49dB	2.57dB

[41]. With the normalised min-sum approximation decoder, the (96,48) and (1008,504) LDPC codes outperform the Reed-Solomon code with 1.25 dB and 3.66 dB respectively. The well known half rate convolutional code has a constraint length of 7, with constraints (171,133), specified in octal base [22]. This code requires 4.15 dB to achieve a BER of 10^{-5}. The smallest LDPC code considered, (96,48), requires 4.98 dB. This convolutional code outperforms the (96,48) LDPC code by 0.83 dB. The (204,102) LDPC code requires 4.15 dB, giving similar results to the convolutional code. All well designed LDPC codes with block lengths larger than approximately 204 should outperform this convolutional code as predicted by Figure 3.6. The (1008,504) LDPC code will outperform the convolutional code by 1.57 dB. A practical code which implements the normalised min-sum decoder with 3-bit soft decision decoding and using 20 iterations for decoding would give an error performance within 0.4 dB of the BP decoder without quantisation, as given in Table 4.1 for the (204,102) and (408,204) codes. These results are presented in Figure 4.8 - Figure 4.10.

A decoding error can be defined as the decoder in Figure 3.1 choosing the wrong codeword x from the received codeword x^*. Decoder failures happen when the received data differs from all valid codewords by a specified distance. Through smart implementation, with the excellent distance property of LDPC codes, the probability of a decoder failure becomes so small that LDPC codes can be employed as an error correction and error detection scheme. The frequency of undetected errors decreases rapidly by increasing the block length [42]. Probability of undetected errors for codes of block lengths larger than 200 is considered to be insignificantly small.

4.3 Hardware considerations

For implementation considerations the approximation algorithm has obvious advantages over the standard BP algorithm. For FPGA implementation the use of summation and compare functions is highly preferred over multiplication. A synthesizable fixed point implementation was designed using AccelDSP for the encoder and decoder. In this section, the implementation of the decoder and encoder is presented.

The Virtex-5 LX series is used as a design platform. The LX series is designed for high-performance general logic applications [38]. Within the Virtex-5 family, the Q series specialising in space applications can be used. This series provides radiation tolerant FPGA's with a guaranteed total ionising dose (TID) of up to 300 krads(Si), and provides single event latchup (SEL) immunity.

A characteristic of using the min-sum approximation algorithm is that it is universal for the AWGN channel [36]. This implies that we can feed the received values from the demodulator into the decoder without computing the probabilities or LLRs. Methods exist where channel information can be used to determine more accurate values for α and β, which will improve the performance of the code [36]. The practical implementation should consider the quantisation errors introduced into the system. Two types of quantisation noise will be discussed in this section, namely quantisation due to soft decision demodulation and quantisation noise introduced in moving from a floating point model to a fixed point model.

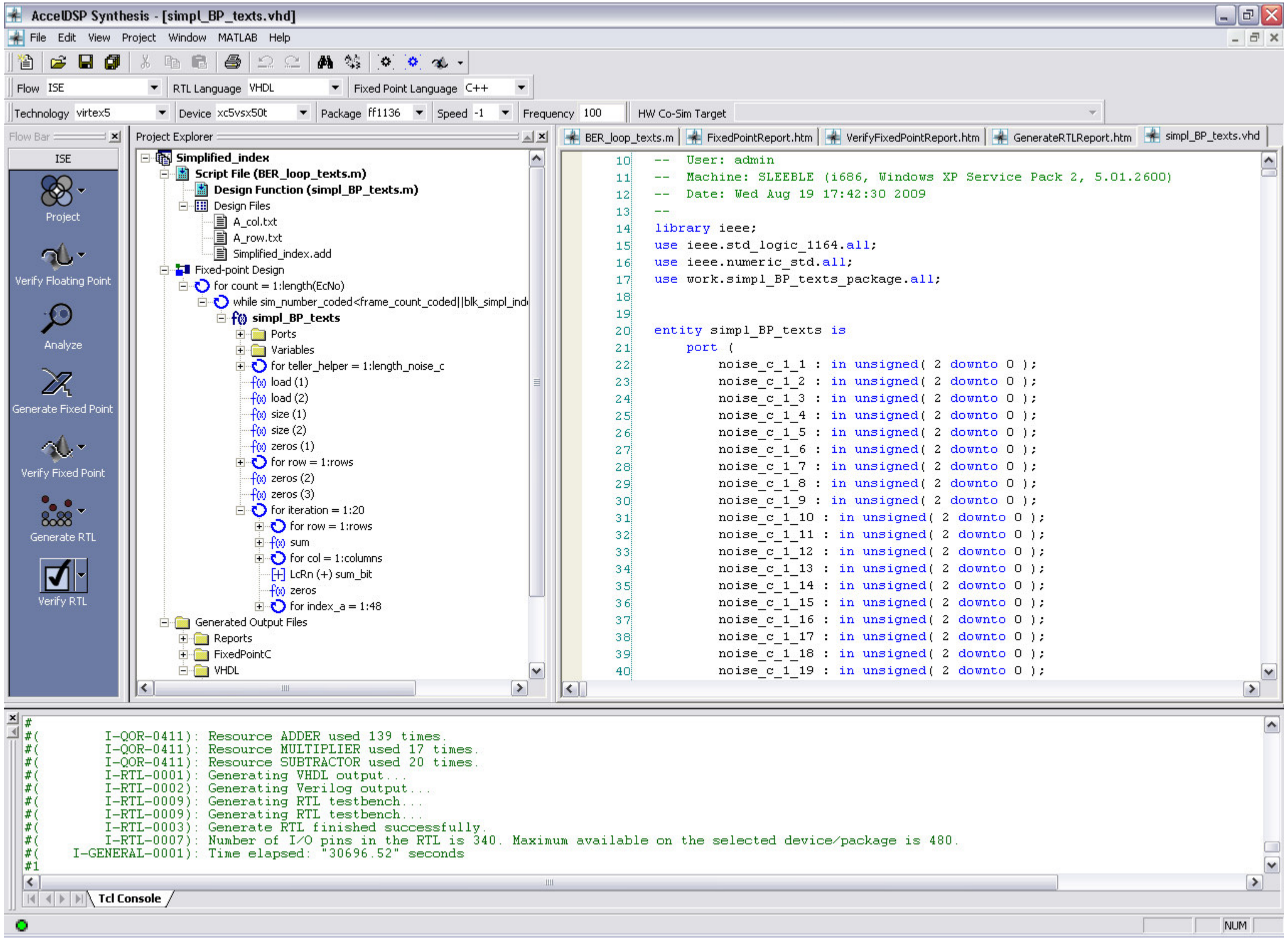
4.3.1 Decoder

A screenshot of the development environment in AccelDSP, after VHDL generation for the decoder, is shown in Figure 4.11. The design flow of RTL generation of the decoder function in AccelDSP will now be explained. The first step involves the verification of the floating point model. The floating point model is the MATLAB simulator explained in Section 3.5, shown as the script file *BER_loop_texts.m* in Figure 4.11. In this implementation a (96,48) LDPC code was used with a normalised min-sum decoder using 20 iterations and 3-bit soft decision quantisation. The results obtained from the floating point model are used to compare with those of the fixed point model. In the *Analyse* step the design function is set. This specifies the synthesizable MATLAB script to be implemented in hardware, *simpl_BP_text*, also known as the top level design function. The input to the design function is an unsigned fixed [3,0] vector representing the 96 bits of the codeword. The [3,0] notation indicates 3 bits to be used in the word length, and 0 of these bits represent the fractional part. The decoder uses 3-bit soft decision decoding as illustrated in Table 4.2.

Table 4.2: Input to the design function

Input value	Interpretation
0	Most confident 0
1	Second most confident 0
2	Third most confident 0
3	Least confident 0
4	Least confident 1
5	Third most confident 1
6	Second most confident 1
7	Most confident 1

The output of the function is an unsigned fixed [1 0] vector representing the 48 decoded bits. The necessary files required by the design function are also added in this step. This includes *A_col.txt*, *A_row.txt* and *Simplified_index.add.* The two text files specify the indices of the non-zero elements in the parity check matrix **H**, as explained in Section 3.5. The file



Flow ISE RTL Language VHDL Fixed Point Language C++

Technology virtex5 Device xc5vsx50t Package ff1136 Speed -1 Frequency 100 HW Co-Sim Target

```
--  User: admin
--  Machine: SLEEBLE (i686, Windows XP Service Pack 2, 5.01.2600)
--  Date: Wed Aug 19 17:42:30 2009
--
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
use work.simpl_BP_texts_package.all;


entity simpl_BP_texts is
    port (
        noise_c_1_1 : in unsigned( 2 downto 0 );
        noise_c_1_2 : in unsigned( 2 downto 0 );
        noise_c_1_3 : in unsigned( 2 downto 0 );
        noise_c_1_4 : in unsigned( 2 downto 0 );
        noise_c_1_5 : in unsigned( 2 downto 0 );
        noise_c_1_6 : in unsigned( 2 downto 0 );
        noise_c_1_7 : in unsigned( 2 downto 0 );
        noise_c_1_8 : in unsigned( 2 downto 0 );
        noise_c_1_9 : in unsigned( 2 downto 0 );
        noise_c_1_10 : in unsigned( 2 downto 0 );
        noise_c_1_11 : in unsigned( 2 downto 0 );
        noise_c_1_12 : in unsigned( 2 downto 0 );
        noise_c_1_13 : in unsigned( 2 downto 0 );
        noise_c_1_14 : in unsigned( 2 downto 0 );
        noise_c_1_15 : in unsigned( 2 downto 0 );
        noise_c_1_16 : in unsigned( 2 downto 0 );
        noise_c_1_17 : in unsigned( 2 downto 0 );
        noise_c_1_18 : in unsigned( 2 downto 0 );
        noise_c_1_19 : in unsigned( 2 downto 0 );
```

```
#
#(        I-QOR-0411): Resource ADDER used 139 times.
#(        I-QOR-0411): Resource MULTIPLIER used 17 times.
#(        I-QOR-0411): Resource SUBTRACTOR used 20 times.
#(        I-RTL-0001): Generating VHDL output...
#(        I-RTL-0002): Generating Verilog output...
#(        I-RTL-0009): Generating RTL testbench...
#(        I-RTL-0009): Generating RTL testbench...
#(        I-RTL-0003): Generate RTL finished successfully.
#(        I-RTL-0007): Number of I/O pins in the RTL is 340. Maximum available on the selected device/package is 480.
#(    I-GENERAL-0001): Time elapsed: "30696.52" seconds
#1
```

Figure 4.11: Screenshot of AccelDSP development environment after RTL generation of normalised min-sum decoder.

Simplified_index.add specifies the user defined variable quantisation for the fixed point model. The quantised fixed point values are chosen to be as small as possible, without decreasing the performance or accuracy of the system beyond unacceptable levels. The three most important variables that need to be quantised by the designer, are the variables representing the probabilities for the bit nodes and check nodes. The bit nodes are represented by the matrix variable **L**. The fixed point assignment for **L** is a signed [20 10] variable. The variable representing the probabilities of the check nodes is a signed [22 11] variable. Note that both of these matrices only represent values for the non-zero elements of the parity check matrix. Most of the quantisation of other variables are auto inferred from these directives.

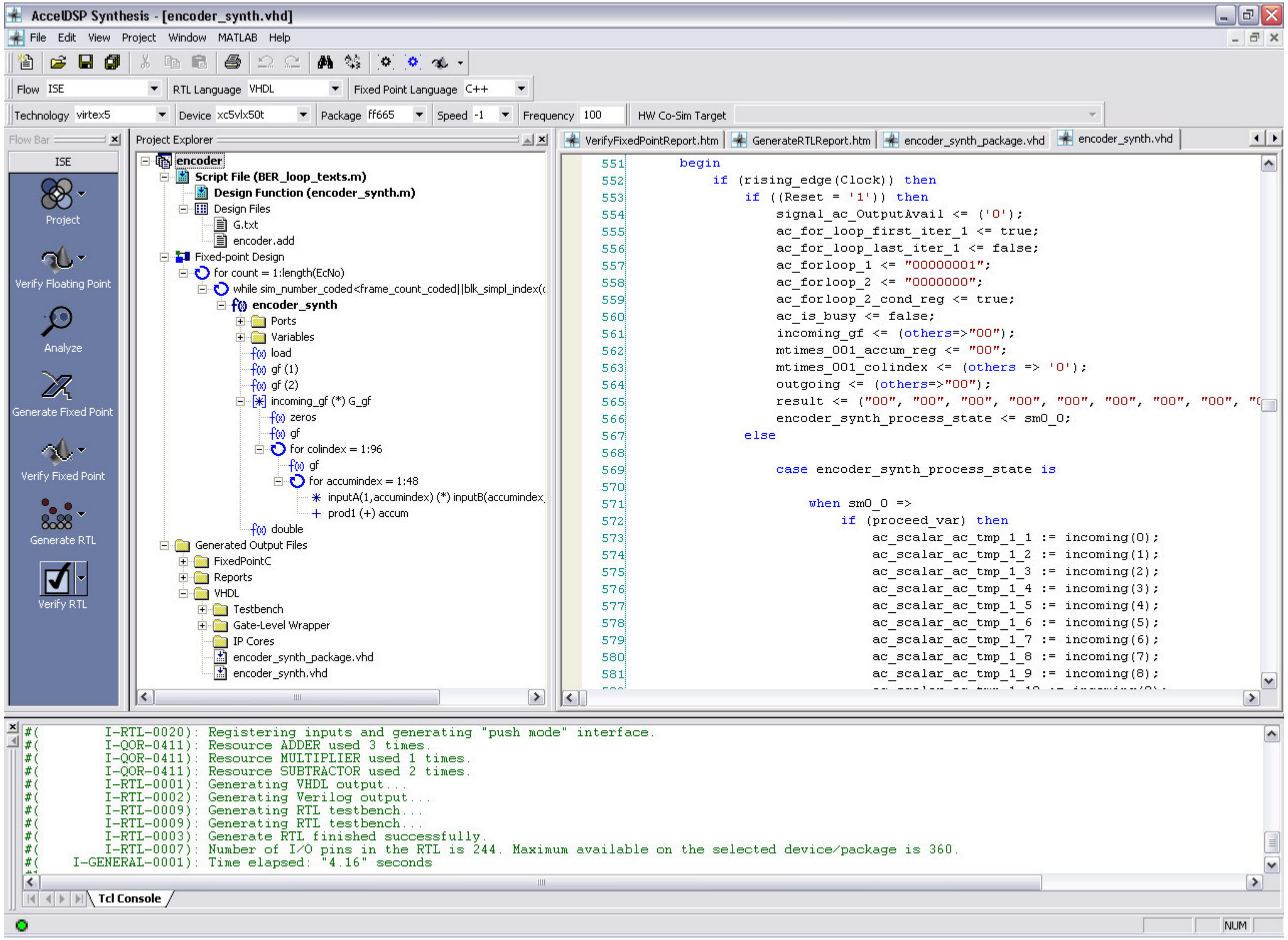
In the next step the fixed point model is generated. The fixed point function is generated in C++. The verification of the fixed point model provides the results using the quantised variables of the fixed point model. The results of the fixed point model are compared to those found in the floating point model. The fixed point report stated that no overflows were detected, and that some underflows were found. All the underflows were found in the calculation of variable **L**. This corresponds to the calculations done in lines $12 - 13$ of Algorithm 1 in Section 3. In all these underflows, the value of the underflow was smaller than 0.00061, and did not show performance degradation. In Algorithm 1 it can be verified that the variable **L** is added to other variables in the bit node update and therefore, does not impose dangers of propagating zero probabilities through the matrices in the unlikely scenario of being rounded to zero.

The results of the fixed point model indicated that the chosen quantised fixed point model doesn't show any performance degradation when compared to the floating point model. The next step is to use the fixed point model to generate the RTL code. In this design, VHDL was chosen, but a Verilog design can also be generated. Using the Virtex5 (xc5vsx50t) device, it was found that the device will require 120 117 hardware clock cycles to complete a function call. Using a 500MHz clock, this implementation results in a throughput of 4 162 codewords per second. This represents an end to end data rate of 199 kbit/s, or 400 kbit/s transmission rate as discussed in Section 2.2.3. The reason why the design function takes so many cycles to complete is because of the inability of the synthesizer to unroll the *for* loops due to memory limitations. This will be discussed in Section 4.3.2. The decoder function call implements the specified decoder using 20 iterations. In order to obtain a constant throughput in the implemented hardware decoder, the following change was made to the decoder, as explained in Algorithm 1. The decoder always completes 20 iterations. The stopping criterion was removed from the decoder. This implies that the decoder will always complete 20 iterations, after which the values of the information bits are calculated. In the synchronisation and channel coding sublayer a CRC code was implemented as the error detection scheme, and it is therefore not required of the LDPC decoder to provide information of detected errors.

4.3.2 Encoder

The encoder of the system is implemented by multiplying the information vector with the generator matrix as discussed in Section 3.2. This is a GF(2) operation. The same communication system used in the decoder can be applied by setting and modifying the encoder to be the top level design file. A screenshot of the development environment of the encoder is shown in Figure 4.12.

The top level design file is the MATLAB script *encoder_synth.m*, and the generator matrix is added and represented by the text file *G.txt*. Increasing the throughput of the system can be changed by changing the synthesizer's design goals. The trade-off is between area versus speed. Unrolling for loops increases the required area needed on the FPGA for implementation, but also increases the speed (or throughput). The encoder was generated using the same design goals for area versus speed as used in the decoder. The throughput of the system is 4 706 hardware clock cycles per function call. When receiving a vector of 48 bits on the input in parallel, codewords will be generated to produce a throughput of 65 139 codewords per second, using an estimated clock frequency of 306.8 MHz. A second design was implemented, unrolling the two for loops used in the design. The second method only requires 2 startup clock cycles, requiring only a single hardware cycle per design function call. This method can provide a throughput of 306 million codewords per second.



Flow ISE
RTL Language VHDL
Fixed Point Language C++
Technology virtex5
Device xc5vlx50t
Package ff665
Speed -1
Frequency 100
HW Co-Sim Target

```
encoder
  Script File (BER_loop_texts.m)
    Design Function (encoder_synth.m)
  Design Files
    G.txt
    encoder.add
  Fixed-point Design
    for count = 1:length(EcNo)
      while sim_number_coded<frame_count_coded||blk_simpl_index(
        encoder_synth
          Ports
          Variables
          load
          gf (1)
          gf (2)
          incoming_gf (*) G_gf
            zeros
            gf
            for colindex = 1:96
              gf
              for accumindex = 1:48
                inputA(1,accumindex) (*) inputB(accumindex
                prod1 (+) accum
          double
  Generated Output Files
    FixedPointC
    Reports
    VHDL
      Testbench
      Gate-Level Wrapper
      IP Cores
      encoder_synth_package.vhd
      encoder_synth.vhd
```

```
    begin
        if (rising_edge(Clock)) then
            if ((Reset = '1')) then
                signal_ac_OutputAvail <= ('0');
                ac_for_loop_first_iter_1 <= true;
                ac_for_loop_last_iter_1 <= false;
                ac_forloop_1 <= "00000001";
                ac_forloop_2 <= "0000000";
                ac_forloop_2_cond_reg <= true;
                ac_is_busy <= false;
                incoming_gf <= (others=>"00");
                mtimes_001_accum_reg <= "00";
                mtimes_001_colindex <= (others => '0');
                outgoing <= (others=>"00");
                result <= ("00", "00", "00", "00", "00", "00", "00", "00", "(
                encoder_synth_process_state <= sm0_0;
            else

                case encoder_synth_process_state is

                    when sm0_0 =>
                        if (proceed_var) then
                            ac_scalar_ac_tmp_1_1 := incoming(0);
                            ac_scalar_ac_tmp_1_2 := incoming(1);
                            ac_scalar_ac_tmp_1_3 := incoming(2);
                            ac_scalar_ac_tmp_1_4 := incoming(3);
                            ac_scalar_ac_tmp_1_5 := incoming(4);
                            ac_scalar_ac_tmp_1_6 := incoming(5);
                            ac_scalar_ac_tmp_1_7 := incoming(6);
                            ac_scalar_ac_tmp_1_8 := incoming(7);
                            ac_scalar_ac_tmp_1_9 := incoming(8);
```

```
#(          I-RTL-0020): Registering inputs and generating "push mode" interface.
#(          I-QOR-0411): Resource ADDER used 3 times.
#(          I-QOR-0411): Resource MULTIPLIER used 1 times.
#(          I-QOR-0411): Resource SUBTRACTOR used 2 times.
#(          I-RTL-0001): Generating VHDL output...
#(          I-RTL-0002): Generating Verilog output...
#(          I-RTL-0009): Generating RTL testbench...
#(          I-RTL-0009): Generating RTL testbench...
#(          I-RTL-0003): Generate RTL finished successfully.
#(          I-RTL-0007): Number of I/O pins in the RTL is 244. Maximum available on the selected device/package is 360.
#(      I-GENERAL-0001): Time elapsed: "4.16" seconds
```

Figure 4.12: Screenshot of AccelDSP development environment after RTL generation of encoder.

4.4 Summary

This chapter presented the performance results of the LDPC decoders. It was shown that the performances of the BP and min-sum decoders offer significant coding gains over uncoded systems. The BP decoder provides the best performance in the lower E_b/N_o range, where the normalised min-sum and standard min-sum show performances within 0.3 dB of the BP decoder. The normalised min-sum decoder outperforms the standard min-sum decoder. In the higher E_b/N_o range the normalised min-sum decoder outperforms the BP decoder. Parameters influencing the performance of the codes were simulated and presented. When implementing the normalised min-sum decoder, the value of the normalisation constant α influences the performance of the code. Results were given to illustrate the performance of the normalised min-sum decoder for different values of α. From simulations, it was determined that the normalised min-sun decoder performs optimally for $0.7 < \alpha < 0.8$, for the codes considered.

In many hardware implementations, a constant throughput is required. Decreasing the maximum number of iterations used by the decoder decreases the processing requirements for hardware implementations. The effect of decreasing the maximum number of iterations on the performance of the code was shown. The normalised min-sum decoder has a fast convergence rate, giving good performance for codes with a small maximum number of iterations. With 20 iterations, a performance within 0.05 dB of the decoder using 100 iterations is obtained. A comparison of the different decoders using a maximum number of 20 iterations was presented. The normalised min-sum decoder gives the best result for E_b/N_o values higher than 3 dB for the (204,102) LDPC code.

The effect of soft decision quantisation on the performance of the decoders was demonstrated. A BP decoder having no quantisation exhibits a performance increase of approximately 1.8 dB over a hard decision decoder. Using 2-bit and 3-bit soft decision decoders results in performance losses of approximately 0.8 dB and 0.4 dB over a decoder without quantisation. Results from the normalised min-sum decoder was also presented, showing similar performances.

Comparative results between LDPC codes, convolutional codes and Reed-Solomon codes were presented. These LDPC codes were relatively short length codes applying a BP decoder. It was shown that LDPC codes, with code lengths larger than approximately 200, outperform a well known convolutional code, and all codes considered outperform the Reed-Solomon code.

A normalised min-sum decoder and encoder for a (96,48) LDPC code was implemented in AccelDSP. The design cycle and development environment was explained. A decoder giving a constant throughput and using 20 iterations was implemented. Due to memory limitations the decoder could not be optimised for speed in this implementation. This limits the portability of the code and is mainly due to unused declared variables pipelined during HDL generation of the decoder. These variables are used in the probability matrices in the horizontal and vertical steps. All implementations were synthesized for a Virtex 5 FPGA.

Chapter 5

Conclusions, contributions and recommendations

5.1 Conclusions

In this thesis, different decoding schemes for short length LDPC codes were investigated. It was shown that for block lengths larger than approximately 200, LDPC codes outperform other popular coding schemes such as convolutional codes and Reed-Solomon codes. Using simplified approximations for the sum-product decoder, and with the appropriate selection of algorithm parameters, the normalised min-sum approximation decoder exhibits similar performance to the BP decoder. These results are obtained with much lower associated computational overhead, which is very important for a low complexity hardware implementation. It was found that the min-sum approximation decoder performs well with a normalisation constant of $\alpha = 0.7$. The normalised min-sum decoder eliminates the requirement for prior intrinsic channel estimation, which additionally simplifies the implementation.

The expected losses associated with hardware implementations were evaluated and provided. These include soft decision quantisation and fixed point quantisation errors. Using 2-bit and 3-bit soft decision decoders results in performance losses of approximately 0.8 dB and 0.4 dB over a decoder without quantisation. It is proved that the error performance of the fixed point implementation is good, while its processing requirements are within acceptable margins.

The performances of the BP and min-sum decoders offer significant coding gains over uncoded systems. The BP decoder provides the best performance in the lower E_b/N_o range, where the normalised min-sum and standard min-sum show performances within 0.3 dB of the BP decoder. In the higher E_b/N_o range the normalised min-sum decoder outperforms the BP decoder. The normalised min-sum decoder has a fast convergence rate, giving good performance for codes with a small maximum number of iterations. When reducing the maximum number of iterations used by the decoders from 100 to 20, the performance loss of the normalised min-sum decoder is less than that of the BP decoder. With 20 iterations the normalised min-sum decoder performs within 0.05 dB of the decoder using 100 iterations. Reducing the maximum number of iterations, increases the throughput of the decoder and decreases the latency. A fixed point implementation of a (96,48) LDPC code in AccelDSP showed good performance with a maximum number of 20 iterations and 3-bit soft decision quantisation. VHDL code was generated by a synthesis tool for the encoder and normalised min-sum decoder.

LDPC codes are versatile and present good error performances at many different code rates and block lengths. It was found that many parameters of the decoders and codes can be altered to allow the implementation of these codes in systems with varying memory and processing capabilities. A generic decoder was explained and implemented. LDPC codes of different lengths, rates and number of iterations can be implemented to meet the memory, bandwidth and latency requirements of a system. Results presented in this thesis can be used as a guideline by designers to predict the performance of LDPC codes in hardware implementations.

It is clear that this proposed implementation of LDPC coding holds much promise to improve the volumetric data throughput of not only ground-satellite communication systems, but for variable and relatively low quality communication links in general.

5.2 Contributions

The following are the main contributions of this thesis:

- A protocol strategy was designed to facilitate a mission specific error control code. An ARQ strategy is combined with a space communication protocol on the data link layer. The error control strategy is implemented on the synchronisation and channel coding sublayer.

- A packet processing scheme was designed to avoid the necessity of packet inspection of higher layer protocols in the data link layer.

- A significant range of forward error correction schemes were investigated for application in LEO satellite systems. From this investigation it was determined that sparse graph codes provide the best performances. From the class of sparse graph codes LDPC codes were chosen for implementation for its good performance in varying link margins and efficient message passing decoders.

- Different variants of the belief propagation decoding algorithm were explained and implemented in MATLAB.

- Min-sum approximation algorithms for the codes were implemented in MATLAB. This particular implemented algorithm has low processing requirements.

- Simulations were used to optimise performance of the min-sum approximation decoder by calibrating the normalisation constant.

- Simulation results were presented that compare the performances and characteristics of the different decoding algorithms for LDPC codes with different block lengths. These simulations were conducted for various channel and decoder conditions. These conditions included soft decision quantisation and maximum number of iterations used by the decoder.

- These results showed that LDPC codes provides good error performances.

- A fixed point model was generated and compared to the floating point model. The fixed point model show good performance.

- A synthesizable MATLAB script was written in AccelDSP and VHDL code generated for an LDPC encoder and decoder to be implemented on a Xilinx FPGA. The design cycle of the hardware implementations was explained and discussed.

5.3 Recommendations for future work

The work conducted in this thesis could form part of a structured process for current and future implementations of forward error correction in LEO satellite communication links. The protocol strategy and error control strategy designed in this thesis is currently being implemented into the IS-HS II satellite project. The design is currently being implemented for an aircraft based communication payload test. These implementations can be extended to other communication links with varying or poor link margins. The following suggestions of future research areas are:

- LDPC codes have excellent error detecting properties. As the code length increases the probability of undetected errors decrease. A further study can be conducted to determine the length at which the probability of undetected error is sufficiently small to be used as an error detecting scheme. With such an implementation the LDPC code can achieve error correction and detection, without the need for a CRC check.

- The performances of the considered LDPC codes were determined by simulation. Theoretical bounds for the performances of codes can be determined by using the analytical technique of density evolution.

- The modulation and detection of signals can be implemented on the FPGA. This will provide a single chip solution and avoid the necessity of an external modem. Such an implementation should be feasible with the DSP capabilities provided by new ranges of FPGAs.

- An adaptive coding and modulation (ACM) strategy can be implemented. Changing the code rate and modulation scheme to adapt to the channel conditions will increase the throughput of the system. For such a scheme channel information should be available at the transmitter. Currently a research project at the Stellenbosch University is being conducted for an on-line scheduler. Such a system could be combined with an ACM scheme.

- Research and implementation can be extended to include LDPC codes with different code rates.

- The synthesis tools of AccelDSP were not able to make use of the sparseness of the matrices, which limits the portability of the code to codes with larger block lengths. The generated VHDL code can be used as a starting point to implement the decoder directly into VHDL code. Another option would be to compile the project in future versions of the synthesis tool, which might support such functions.

List of References

[1] R. Gallager, “Low density parity check codes,” MIT Press, Cambridge Mass, 1963.

[2] D. MacKay, “Good error-correcting codes based on very sparse matrices,” *IEEE Transactions on Inf. Theory*, vol. 45, no. 2, pp. 399–431, Mar 1999.

[3] S. Chung, G. Forney, T. Richardson, and R. Urbanke, “On the design of low-density parity-check codes within 0.0045 dB of the Shannon limit,” *IEEE Communication Letters*, vol. 5, no. 2, pp. 58–60, Feb 2001.

[4] M. Francheschini, G. Ferrari, and R. Raheli, “Does the performance of LDPC codes depend on the channel?” Int. Symp. of Inf. Theory and its App., Oct 2004.

[5] N. Celandroni, “Comparison of FEC types with regard to the efficiency of TCP connections over AWGN satellite channels,” *IEEE Transactions on Wireless Communications*, vol. 5, no. 7, pp. 1735–1745, Jul 2006.

[6] D. J. C. MacKay, *Information Theory, Inference, and Learning Algorithms.* Cambridge University Press, 2003.

[7] J. Chen and M. P. C. Fossorier, “Density evolution for two improved BP-based decoding algorithms of LDPC codes,” *IEEE Communications Letters*, vol. 6, no. 5, pp. 208–210, May 2002.

[8] G. Maral and M. Bousquet, *Satellite communications systems*, third edition ed. John Wiley and Sons, LTD, 1998.

[9] I. Ali, P. G. Bonanni, N. Al-Dhahir, and J. E. Hershey, *Doppler applications in LEO satellite communication systems.* Springer, 2002.

[10] P. D. W. Aerts and G. A. E. Vandenbosch, “Conceptual study of analog baseband beam forming: Design and measurement of an eight-by-eight phased array,” *IEEE Transactions on Antennas and Propagation*, vol. 57, pp. 1667–1672, Jun 2009.

[11] “Controller area network (CAN) part 1: Data link layer and physical signalling,” International Organization for Standardization, ISO 11898-1, 2004.

[12] *QNX Neutrino RTOS*, QNX Software Systems International Corporation, June 2004.

[13] C. E. Shannon, “A mathematical theory of communication,” *Bell System Technical Journal*, vol. 27, 1948.

[14] S. B. Wicker, *Error Control Systems for Digital Communication and Storage.* Prentice Hall, 1995.

[15] T. K. Moon, *Error Correction Coding.* Hoboken, New Jersey: John Wiley & Sons, 2005.

[16] J. G. Proakis, *Digital Communications*, 4th ed. McGraw-Hill, 2001.

[17] B. Sklar, *Digital Communications, Fundamentals and Applications.* Prentice Hall, 1988.

[18] “Space data links - Telecommand protocols, synchronization and channel coding,” European Cooperation for Space Standardization, E-50-04A, Nov 2007.

[19] I. S. Reed and G. Solomon, “Polynomial codes over certain finite fields,” *Journal of the Society for Industrial and Applied Mathematics*, vol. 8, no. 2, pp. 300–304, 1960.

[20] P. Elias, “Coding for noisy channels,” *IRE Conv. Record*, pp. 37–47, 1955.

[21] A. Viterbi, “Convolutional codes and their performance in communication systems,” *IEEE Transactions on Communication Technology*, vol. 19, pp. 751–772, 1971.

[22] “Space data links - Telemetry synchronization and channel coding,” European Cooperation for Space Standardization, E-50-01A, Nov 2007.

[23] C. Berrou, A. Gaviuex, and P. Thitimajshima, “Near shannpn limit error-correcting coding and decoding: Turbo codes,” in *1993 IEEE International Conference on Communications*, Geneva, Switzerland, 1993, pp. 1064–1070.

[24] C. B. Schlegel and L. C. Pérez, *Trellis and Turbo Coding*, ser. IEEE Press Series on Digital and Mobile Communication, J. Anderson, Ed. John Wiley and Sons, INC, 2004.

[25] Y. Dai, Z. Yan, and N. Chen, “Optimal overlapped message passing decoding of quasi-cyclic LDPC codes,” *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 16, no. 5, pp. 565–578, May 2008.

[26] Z. Li, L. Chen, L. Zeng, S. Lin, and W. H. Fong, “Efficient encoding of quasi-cyclic low density parity-check codes,” *IEEE Transactions on Comminications*, vol. 54, no. 1, pp. 71–81, Jan 2006.

[27] Y. Chen and K. K. Parhi, “Overlapped message passing for quasi-cyclic low-density parity check codes,” *IEEE Transactions on Circuits and Systems*, vol. 51, no. 6, pp. 1106–1113, Jun 2004.

[28] T. J. Richardson and R. L. Urbanke, "Efficient encoding of low-density parity-check codes," *IEEE Transactions on Information Theory*, vol. 47, no. 2, February 2001.

[29] F. Olivier and R. Wolhuter, "A modem link strategy for optimization of LEO satellite throughput," no. 8. Wild coast sun, South Africa: Southern Africa Telecommunication Networks and Applications Conference, September 2008.

[30] R. M. Tanner, D. Sridhara, A. Sridharan, T. E. Fuja, and D. J. Costello, "LDPC block and convolutional codes based on circulant matrices," *IEEE Transactions on Information Theory*, vol. 50, no. 12, pp. 2966–2984, Dec 2004.

[31] R. M. Tanner, "A recursive approach to low complexity codes," *IEEE Transactions on Information Theory*, vol. 27, no. 5, pp. 533–547, Sep 1981.

[32] D. Haley and A. Grant, "Reversible low-density parity-check codes," *IEEE Transactions on Information Theory*, vol. 55, no. 5, pp. 2016–2036, May 2009.

[33] "Space data links - Telemetry transfer frame protocol," European Cooperation for Space Standardization, E-50-03A, Nov 2007.

[34] [Online]. Available: http://www.inference.phy.cam.ac.uk/mackay/

[35] F. Olivier and R. Wolhuter, "Evaluation of LDPC forward error correction schemes for telemetry satellite systems," no. 9. Royal Swazi Spa, Swaziland: Southern Africa Telecommunication Networks and Applications Conference, September 2009.

[36] J. Chen, "Reduced complexity decoding algorithms for low-density parity check codes and turbo codes," Ph.D. dissertation, University of Hawai, 2003.

[37] "TM synchronization and channel coding," Consultative Committee for Space Data Systems, 131.0-B-1, Sept 2003.

[38] "Virtex-5 family overview," XILINX, Tech. Rep. DS100, Sep 2008.

[39] "Spacecraft identification field: code assignment control procedures," Consultative Committee for Space Data Systems, Blue Book 320.0-B-5, Sep 2007.

[40] *ML501 Evaluation Platform*, Ug226(v1.2.1) ed., XILINX, Jun 2008.

[41] "TM synchronization and channel coding Summary of concept and rationale," Consultative Committee for Space Data Systems, 130.1-G-1, Jun 2006.

[42] D. J. C. MacKay, "Gallager codes – recent results," in *Coding, Communications and Broadcasting*, P. Farrell, M. Darnell, and B. Honary, Eds. Baldock, Hertfordshire, England: Research Studies Press, 2000, pp. 139–150.