VLSI Implementation of LDPC Codes

Soumya Ranjan Biswal
209EC2124

Department of Electronics and Communication Engineering
National Institute of Technology, Rourkela
Rourkela-769008, Odisha, INDIA
May 2013.

VLSI Implementation of LDPC Codes

A dissertation submitted in partial fulfillment of the
requirement for the degree of

"Master of Technology"

in

VLSI Design and Embedded System

by

Soumya Ranjan Biswal

(Roll-209EC2124)

Under the Guidance of

Dr. Sarat Kumar Patra

Department of Electronics and Communication Engineering

National Institute of Technology, Rourkela

Rourkela-769008, Odisha, INDIA

Dedicated

To

MY LOVING PARENTS AND MY SISTER

Declaration

I certify that

- The work contained in this thesis is original and has been done by me under the guidance of my supervisor (s).
- The work has not been submitted to any other Institute for the award of any other degree or diploma.
- I have followed the guidelines provided by the Institute I preparing the thesis.
- I have confirmed to the norms and guidelines in the Ethical Code of Conduct of the Institute.
- Whenever I used materials (data, theoretical analysis, figures, and text) from other sources, I have given due credit to them by citing them in the text of the thesis and giving their details in the references. Further, I have taken permission from the copyright owners of the sources, whenever necessary.

Soumya Ranjan Biswal

Rourkela, May 2013

Department of Electronics and Communication Engineering

National Institute of Technology, Rourkela

C E R T I F I C A T E

*This is to certify that the thesis entitled **"VLSI Implementation of LDPC Codes"** being submitted by **Mr. Soumya Ranjan Biswal**, to the National Institute of Technology, Rourkela (Deemed University) for the award of degree of Master of Technology in **Electronics and Communication Engineering** with specialization in **"VLSI Design and Embedded System"**, is a bonafide research work carried out by him in the **Department of Electronics and Communication Engineering**, under my supervision and guidance. I believe that this thesis fulfills a part of the requirements for the award of degree of Master of Technology. The research reports and the results embodied in this thesis have not been submitted in parts or full to any other University or Institute for the award of any other degree or diploma.*

Dr. Sarat Kumar Patra
Dept. of Electronics & Communication.
National Institute of Technology
Rourkela, Odisha, 769008

Place: N.I.T., Rourkela
Date:

Acknowledgements

First and foremost, I am truly indebted and wish to express my gratitude to my supervisor Professor Sarat Kumar Patra for his inspiration, excellent guidance, continuing encouragement and unwavering confidence and support during every stage of this endeavour without which, it would not have been possible for me to complete this undertaking successfully. I also thank him for his insightful comments and suggestions which continually helped me to improve my understanding.

I express my thanks to my co-guide Neelesh Kumar Rathore who helped me in my project. I express my deep gratitude to the members of Masters Scrutiny Committee, Professors D. P. Acharya, and A. K. Swain for their loving advice and support. I am very much obliged to the Head of the Department of Electronics and Communication Engineering, NIT Rourkela for providing all possible facilities towards this work. Thanks to all other faculty members in the department.

I would also like to express my heartfelt gratitude to my friend Abhisek Kumar & Kumar Prasannjit Pradhan who have inspired me and particularly helped in the project.

My wholehearted gratitude to my parents, Chaitanya Biswal, Ranita Biswal and my sister Sagarika Biswal for their constant encouragement, love, wishes and support. Above all, I thank Almighty who bestowed his blessings upon us.

Soumya Ranjan Biswal

Rourkela, May 2013

Table of Contents

ABSTRACT xi

Introduction 1

1.1 Overview 1

1.2 Error Detection and Correction Schemes 2

1.2.1 Error Detection Scheme 3

1.2.2 Forward error correction (FEC) 4

1.3 Objective of this Thesis 5

1.4 Organization of the Thesis 6

Chapter 2 8

Low-Density Parity-Check (LDPC) Codes 8

2.1 Basics of LDPC codes 8

2.1.1 Linear block codes 8

2.1.2 Definition of LDPC codes 12

2.1.3 Tanner graphs 12

2.1.4 Regular and irregular LDPC codes 14

2.2 Construction of LDPC codes 15

2.2.1 Gallager codes 16

2.2.2 Quasi-cyclic (QC) LDPC codes 16

2.3 Encoding of LDPC codes 17

2.3.1 Conventional encoding based on Gauss-Jordan elimination 17

2.3.2 Lower Triangular Based Encoding 18

2.3.3 Other encoding schemes 19

2.4 Iterative Decoding Algorithm 21

2.4.1 Overview of Different Decoding Algorithms 21

2.4.2 Probability-Domain SPA Decoder 23

2.4.3 Log-Domain SPA Decoder: 27

2.4.4. Reduced Complexity Decoders 29

Chapter 3 32
LDPC coded Communication System 32
3.1 LDPC based Communication System 32
3.2 LDPC Encoder 32
3.2.1 Construction of Parity check Matrix 32
3.2.2 Encoder hardware Implementation 35
3.3 LDPC Decoder: 36
3.3.1 Sum Product Algorithm 36
3.3 Performance of LDPC System 38
3.3.1 Performance over an AWGN channel 39
3.3.2 Different Rates of LDPC and their Performance review 39
3.3.3 Performance Observation with different number of Iteration 40
3.3.4 Time of decoding for different types of code for different kind of Rate 41
Chapter 4 43
FPGA Implementation of LDPC Code 43
4.1 VHDL Basics 43
4.1.1 Capabilities of VHDL 44
4.2 VHDL Implementation of LDPC Encoder and Decoder 45
4.2.1 LDPC Encoder 45
4.2.2 LDPC Decoder 49
4.3 Result Analysis 53
4.3.1 SNR vs BER plot for H Matrix used for Implementation of LDPC code 54
4.3.2 Test Bench Wave Form 54
Chapter 5 55
Conclusion and future works 55
5.1 Conclusion 55
5.2 Future Works 56
Bibliography 57

List of Figures

Figure 1-1 An FEC Encoded Communication System. 1
Figure 1-2 Classification of different types of ECC Codes[22] 2
Figure 2-1 Systematic form of a codeword of a block code 10
Figure 2-2 Diagram of a block coding system 11
Figure 2-3 Tanner graph corresponding to the parity check matrix H in (2.6) 13
Figure 2-4 Shape of parity check matrices for efficient encoding, by MacKay et al. (MacKay, Wilson, and Davey 1999) (a) and Richardson et. al. (Richardson and Urbanke 2001) (b) 19
Figure 2-5 Sub graph of a tanner graph; Arrows indicate message passing between c-node to v-node 22
Figure 2-6 Sub graph of Tanner graph, showing message passed from c-node to v-node 22
Figure 2-7 Illustration of message passing half-iteration for the computation *qij* (b) 24
Figure 2-8 Illustration of message passing half –iteration for the computation of *rji(b)* 25
Figure 3-1 LDPC Coded BPSK Modulation in AWGN Channel 32
Figure 3-2 (a) AR3A Protograph (b) AR4A protograph 34
Figure 3-3 *H* Matrix used for Performance Study 35
Figure 3-4 Structure of Circulant Encoder using Shift Register 36
Figure 3-5 State Machine for SPA[23] 38
Figure 3-6 Normal BPSK vs. LDPC Coded BPSK System 39
Figure 3-7 BER performance curve for Different rates of H matrix 39
Figure 3-8 LDPC coded System with 2048X4096 matrix with varying no. of Iteration. 40
Figure 4-1 H and G Matrix used for VHDL implementation 46
Figure 4-2 Top Level Schematic of Encoder 47
Figure 4-3 RTL Schematic of Encoder 48
Figure 4-4 Device utilization of Encoder Implementation (Spartan 3E) 48
Figure 4-5 Tanner graph representation for SPA 49
Figure 4-6 Decoder Top Level Schematic 52
Figure 4-7 Decoder RTL Schematic 52
Figure 4-8 Enlarged portion of Decoder RTL Schematic 53
Figure 4-9 Device utilization for Decoder in Virtex 4 board 53
Figure 4-10 SNR Waveform for H Matrix used for Encoding/Decoding 54
Figure 4-11 Encoder Test Bench Wave Form 54
Figure 4-12 Decoder Test Bench Wave Form 54

List of Tables

Table 1 Summary of the different LDPC encoding schemes 20
Table 2 Characteristic of LDPC System under consideration 38
Table 3 Time required for Decoding for various cases..... 42
Table 4 Quantization table for *tanh* and $\tanh^{-1}$ approximation 51

ABSTRACT

Coded modulation is a bandwidth-efficient scheme that integrates channel coding and modulation into one single entity to improve performance with the same spectral efficiency compared to uncoded modulation. Low-density parity-check (LDPC) codes are the most powerful error correction codes (ECCs) and approach the Shannon limit, while having a relatively low decoding complexity. Therefore, the idea of combining LDPC codes and bandwidth-efficient modulation has been widely considered.

In this thesis we will consider LDPC codes as an Error Correcting Code and study it's performance with BPSK system in AWGN environment and study different kind of characteristics of the system. LDPC system consists of two parts Encoder and Decoder. LDPC encoder encodes the data and sends it to the channel. The LDPC encoding performance depends on Parity matrix behavior which has characteristics like Rate, Girth, Size and Regularity. We will study the performance characteristics according to these characteristics and find performance variation in term of SNR performance. The decoder receives the data from the channel and decodes it. LDPC decoder has characteristics like time of iteration in addition all parity check matrix characteristics. We will also study the performance according to these characteristics.

The main objective of this thesis is to implement LDPC system in FPGA. LDPC Encoder is implementation is done using Shift-Register based design to reduce complexity. LDPC decoder is used to decode the information received from the channel and decode the message to find the information. In the decoder we have used Modified Sum Product (MSP) Algorithm to decode, In the MSP we have used some quantized values to decode the data using Look Up Table (LUT) approximation. Finally we compare the SNR performance of theoretical LDPC system's with FPGA implemented LDPC system's performance

Chapter 1

Introduction

1.1 Overview

Communication system transmits data from source to transmitter through a channel or medium such as wired or wireless. The reliability of received data depends on the channel medium and external noise and this noise creates interference to the signal and introduces errors in transmitted data. Shannon through his coding theorem showed that reliable transmission could be achieved only if data rate is less than that of channel capacity. The theorem shows that a sequence of codes of rate less than the channel capacity have the capability as the code length goes to infinity [1].Error detection and correction can be achieved by adding redundant symbols to the original data called as error correction and correction codes (ECCs).Without ECCs data need to retransmitted if it could detect there is an error in the received data. ECC are also called as for error correction (FEC) as we can correct bits without retransmission. Retransmission adds delay, cost and wastes system throughput. ECCs are really helpful for long distance one way communications such as deep space communication or satellite communication. They also have application in wireless communication and storage devices.

Figure 1.1 shows a communication system diagram showing data movement from source to destination. Data from input is given to the Encoder for Encoding and then it is modulated using standard modulation technique then it is transmitted through AWGN channel. The output then fed to demodulation and finally it is decoded with the decoder.

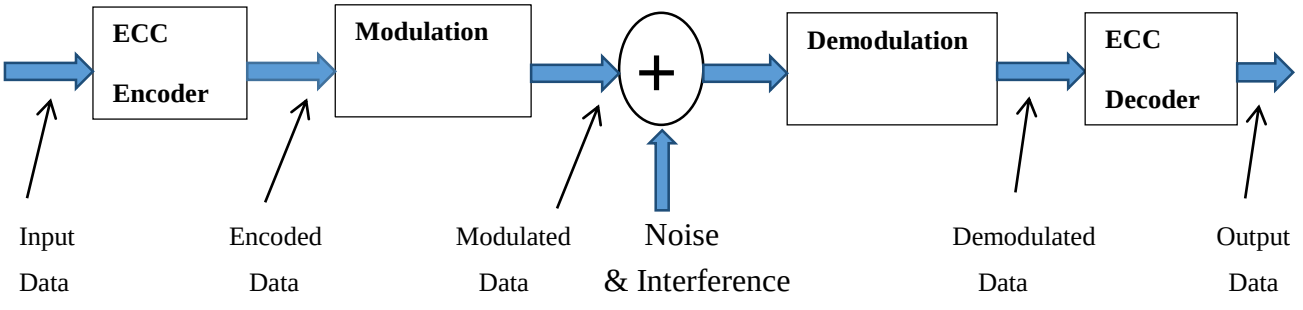


Figure 1-1 An FEC Encoded Communication System.

1.2 Error Detection and Correction Schemes

Error detection and correction helps in transmitting data in a noisy channel to transmit data without errors. Error detection refers to detect errors if any received by the receiver and correction is to correct errors received by the receiver. The overall classification of error correction and detection can be classified as shown in figure 1.2

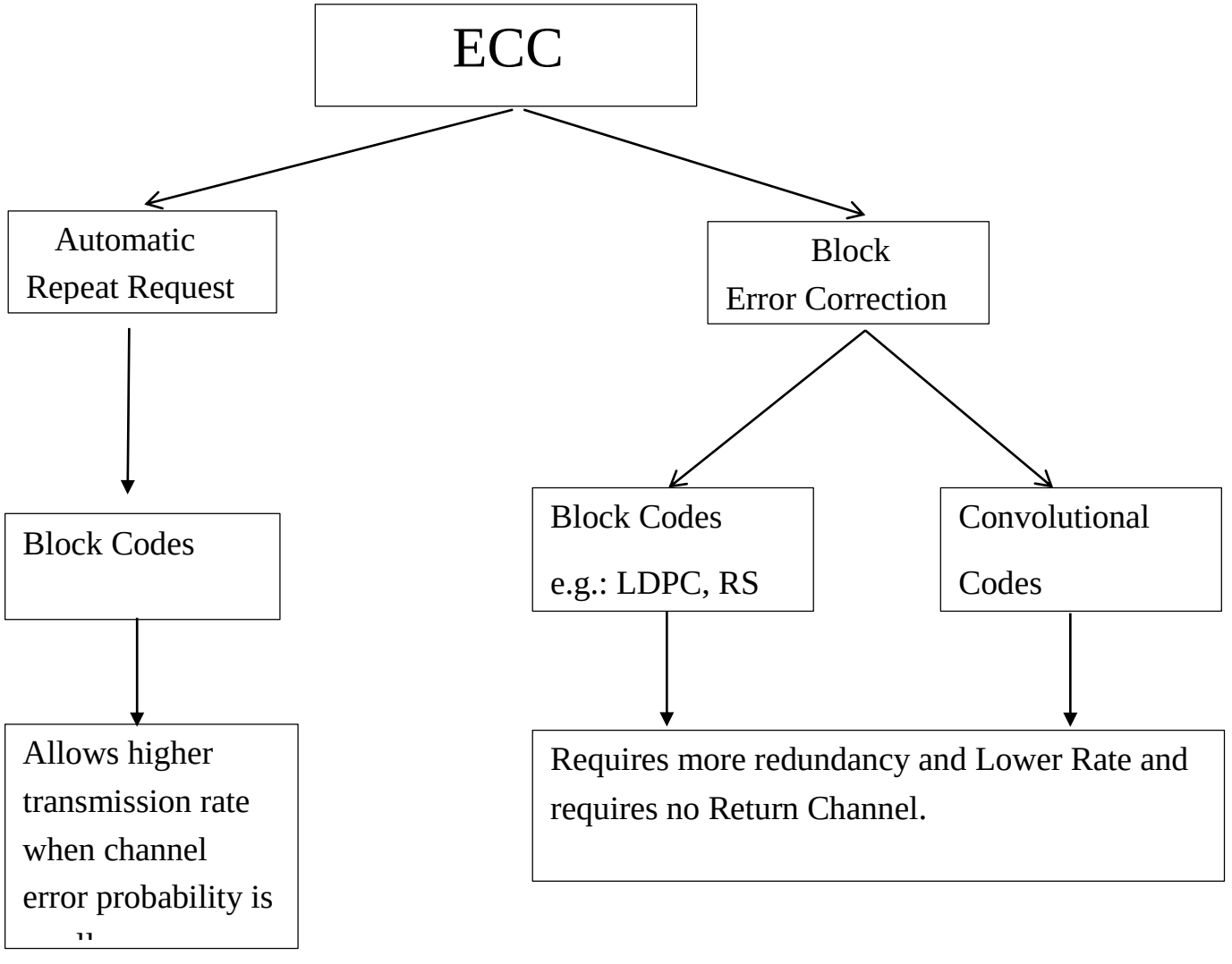


Figure 1-2 Classification of different types of ECC Codes [22]

Different errors correcting codes are there and can be used depending on the properties of the system and the application in which the error correcting is to be introduced. Generally error correcting codes have been classified into block codes and convolutional codes. The distinguishing feature for the classification is the presence or absence of memory in the encoders for the two codes.

To generate a block code, the incoming information stream is divided into blocks and each block is processed individually by adding redundancy in accordance with a prescribed algorithm. The decoder processes each block individually and corrects errors by exploiting redundancy.

In a convolutional code, the encoding operation may be viewed as the discrete–time convolution of the input sequence with the impulse response of the encoder. The duration of the impulse response equals the memory of the encoder. Accordingly, the encoder for a convolutional code operates on the incoming message sequence, using a sliding window equal in duration to its own memory. Hence in a convolutional code, unlike a block code[3] where code words are produced on a block─ by ─ block basis, the channel encoder accepts message bits as continuous sequence and thereby generates a continuous sequence of encoded bits at a higher rate [7].

An error-correcting code (ECC) or forward error correction (FEC) code is a system of adding redundant data, or parity data, to a message, such that it can be recovered by a receiver even when a number of errors (up to the capability of the code being used) were introduced, either during the process of transmission, or on storage. Since the receiver does not have to ask the sender for retransmission of the data, a back-channel is not required in forward error correction, and it is therefore suitable for simplex communication such as broadcasting. Error-correcting codes are frequently used in lower-layer communication, as well as for reliable storage in media such as CDs, DVDs, hard disks, and RAM modules.

1.2.1 Error Detection Scheme

Error detection is about detecting errors and is mostly achieved through parity bits or CRC. There is various error detection schemes used in communication system. Some of the schemes discuss below [3] [4]

1. Parity scheme ─ in parity scheme all the data sets are assigned a particular parity i.e. either even or odd. In the receiver parity of received data is checked. If it does not satisfy the assigned parity,

it is found to be in error. It is effective only for odd number of errors. It cannot detect even number of errors as even number of errors will leave the parity unchanged.

2. Checksum Scheme ─In this scheme a checksum is calculated in the transmitter and sent with the actual data. In receiver checksum is calculated and compared with the received checksum. A mismatch is an indication of error. If data and checksum both are received with error then the detection may not be possible.
3. Cyclic Redundancy Check (CRC) scheme ─ In this scheme the message is interpreted as polynomial and is divided by a generator polynomial. Then the reminder of the division is added to the actual message polynomial to form a code polynomial. This code polynomial is always divisible by the generator polynomial. This property is checked by the receiver. If failed to satisfy this property the received code word is in error. It is complex but efficient error detection scheme.
4. Hamming distance Based Check scheme ─ This scheme is basically parity based scheme but here parity of different combination of bits are checked for parity. It can detect double errors and can correct single errors.
5. Polarity scheme ─ In this scheme the actual message along with its inversion format. In receiver it is checked whether two sets are inverse of each other. If not it is an indication of error. It is not as popular as the code occupies double the bandwidth for the actual message. Moreover if corresponding bits in the data and is inverse are in error then it will not be able to detect the error.

1.2.2 Forward error correction (FEC)

The sender encodes the data using an error-correcting code (ECC) prior to transmission. The additional information (redundancy) added by the code is used by the receiver to recover the original data. In general, the reconstructed data is what is deemed the "most likely" original data [3]. There are several ways of classifying the forward error correction codes as per different characteristics.

1. Linear vs. Nonlinear─ Linear codes are those in which the sum of any two valid code words is also a valid code word. In case of nonlinear code the above statement is not always true.
2. Cyclic vs. Non-Cyclic ─ Cyclic code word are those in which shifting of any valid code word is also a valid code word. In case of non-circular code word the above statement is not always true.
3. Systematic vs. Nonsystematic─ Systematic codes are those in which the actual information appears unaltered in the encoded data and redundant bits are added for detection and correction

of error. In non-systematic code the actual message does not appear in its original form in the code rather there exists one mapping method from the data word to code word and vice versa.

4. Block vs. convolutional ─The block codes are those in which one block of message is transformed into on block of code. In this case no memory is required. In case of convolutional code a sequence of message is converted into a sequence of code. Hence encoder requires memory as present code is combination of present and past message.
5. Binary vs. Non binary ─Binary codes are those in which error detection and correction is done on binary information i.e. on bits. Hence after the error is located, correction means only flipping the bit found in error. In Non-binary code error detection and corrections are done on symbols, symbols may be binary though. Hence both the error location and magnitude is required to correct the symbol in error.

1.3 Objective of this Thesis

1. In communication system we need a good SNR vs BER performance and to do so we use Error correction and detection. Error Correcting is to re-structure and re-build bits to find the correct bits. Most notable performance of Error correction code is given by Block Error Correction codes which perform Block-by-Block basis. LDPC is a type of Block error correction codes which has SNR vs BER performance close to Shanon's Limit. The main Objective of this thesis is to implement of LDPC codes using FPGA.
2. In this thesis we will compare the characteristics curve between normal BPSK system in Gaussian channel with LDPC coded system in same environment .We will also study different characteristics of LDPC performance parameters and study their characteristics.
3. Finally we will implement the LDPC codes using FPGA system using VHDL programming and then will study it's performance behavior in comparison to theoretical values.

1.4 Organization of the Thesis

The main goal of the Thesis is FPGA implementation of LDPC codes and for that we will start from LDPC performance then study it's viability and finally we will go on to implement using VHDL programming.

Chapter 1 deals with basics of error correction and detection schemes and classification of various FEC codes.

Chapter 2 deals with, History of LDPC codes, it's performance behavior and discussion of different types of Encoding and Decoding algorithms.

Chapter 3 Discussion about the performance of algorithms used in the project for Encoding and Decoding and study it's performance analysis using BER vs SNR curves.

Chapter 4 is about implementation of LDPC codes using VHDL coding environment and studies its performance in comparison to theoretical values as obtained in Chapter 3.

Chapter 5 is about Conclusion and future works.

Chapter 2

Low-Density Parity-Check (LDPC) Codes

Low-density parity-check (LDPC) codes are a class of linear block error correction codes (ECC) which provide near-capacity performance. They were invented by Robert Gallager in 1962 [1]. However, these codes were neglected for more than 30 years, since the hardware at that time could not attain the requirements needed by the encoding process. With the increased capacity of computers and the development of relevant theories such as the belief propagation algorithm and Turbo codes, LDPC codes were rediscovered by Mackay and Neal in 1996 [3] .In the last decade, researchers have made great progress in the study of LDPC codes due to technological advancement.

This chapter provides the basics for the study and practice of LDPC codes. We start with the concept of linear block codes and LDPC codes, as well as their representation, classification and degree distribution. Then, we briefly review construction techniques and an efficient encoding method for LDPC codes. Finally, the iterative decoding of LDPC codes which provides near-optimal performance and low decoding complexity is presented via simulation results.

2.1 Basics of LDPC codes

2.1.1 Linear block codes

Assume that the message to be encoded is a k-bit block constituting a generic message m = (m1, m2… mk), that is one of 2k possible messages. The encoder takes this message and generates a codeword c = (c1, c2,... cn), where n > k; that is, redundancy is added. Besides block coding, convolutional coding is also a mechanism for adding redundancy in error correcting coding (ECC) techniques.

***Definition of linear block codes*:** A block code c is a linear code if the codewords form a vector subspace of the vector space V^n; there will be k linearly independent vectors that in turn are codewords, such that each possible codeword is a linear combination of them [11].This definition means that the set of 2k

codewords constitutes a vector subspace of the set of words of n bits. A linear code is characterized by the fact that the sum of any two codewords is also a codeword.

Generator matrix

Let c (n,k) be a linear block code and let (g_1,g_2, ….., g_k)be k linearly independent vectors. Each codeword is a linear combination of them:

$$c = m_1.g_1 + m_2.g_2 + \ldots\ldots + m_k.\, g_k \qquad (2.1)$$

Unless stated otherwise, all vector and matrix operations are modulo 2. These linearly independent vectors can be arranged in a matrix called the generator matrix G:

$$G = \begin{pmatrix} g_1 \\ g_2 \\ . \\ . \\ g_k \end{pmatrix} = \begin{pmatrix} g_{1,1} & g_{1,2} & g_{1,3} \ldots\ldots\ldots\ldots g_{1,n} \\ g_{2,1} & g_{2,2} & g_{2,3} \ldots\ldots\ldots\ldots g_{2,n} \\ \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \\ \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \\ \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \\ \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \\ g_{k,1} & g_{k,2} & g_{k,3} \ldots\ldots\ldots\ldots g_{k,n} \end{pmatrix} \qquad (2.2)$$

For a given message vector m = (m_1,m_2…m_k), the corresponding codeword is obtained by matrix multiplication:

$$c = \mathbf{m.G} = (m_1, m2, \ldots m_k).\begin{bmatrix} g_1 \\ g_2 \\ g_3 \\ . \\ . \\ g_k \end{bmatrix} = m_1g_1 + m_1g_2 + \ldots\ldots + m_kg_k \qquad (2.3)$$

Parity-check matrix

The parity-check matrix H is an (n - k) x n matrix with (n -k) independent rows. It is the dual space of the code c, i.e. $GH^T = 0$.

$$H = \begin{bmatrix} h_1 \\ h_2 \\ . \\ . \\ . \\ h_{n-k} \end{bmatrix} = \begin{bmatrix} h_{1,1} & h_{1,2} & h_{1,3} & \ldots\ldots\ldots\ldots\ldots & h_{1,n} \\ h_{2,1} & h_{2,2} & h_{2,3} & \ldots\ldots\ldots\ldots\ldots & h_{2,n} \\ \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots & & & & \\ \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots & & & & \\ \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots & & & & \\ h_{n-k,1} & h_{n-k,2} & h_{n-k,3} & \ldots\ldots\ldots\ldots & h_{n-k,n} \end{bmatrix} \quad (2.4)$$

It can also be verified that the parity-check equations can be obtained from the parity check matrix H, i.e. $cH^T = 0$. Hence, this matrix also specifies completely a given block code.

2.1.1.1 Block Codes in Systematic form

The structure of a codeword in systematic form is shown in Fig. 2.1. In this form, a codeword consists of k message bits followed by (n-k) parity-check bits.

k message bits	n-k parity bits

Figure 2-1 Systematic form of a codeword of a block code

Thus, a systematic linear block code c(n, k) can be specified by the following generator matrix:

$$G = \left[\begin{array}{ccc|c} 1 & 0 & 0\ldots\ldots\ldots\ldots 0 & p_{1,k+1} & p_{1,k+2} \ldots\ldots\ldots & p_{1,n} \\ 0 & 1 & 0\ldots\ldots\ldots\ldots 0 & p_{2,k+1} & p_{2,k+2} \ldots\ldots\ldots & p_{2,n} \\ 0 & 0 & 1\ldots\ldots\ldots\ldots 0 & p_{3,k+1} & p_{3,k+2} \ldots\ldots\ldots & p_{3,n} \\ \ldots. & & \ldots. & \ldots.. & & \ldots\ldots\ldots \\ \ldots\ldots & & \ldots.. & \ldots\ldots & & \ldots\ldots\ldots \\ \ldots\ldots\ldots & & & & & \\ 0 & 0 & 1 & p_{k,k+1} & p_{3,k+2} \ldots\ldots\ldots & p_{k,n} \end{array}\right] \quad (2.5)$$

*Identity Matrix (k*k)* *Parity Matrix (k*n-k)*

Which, in a compact notation, is G = [I_{k*k} $P_{k*(n-k)}$]. The corresponding parity-check matrix is given by H = [$P^T_{(n-k)*k}$ $I_{(n-k)*(n-k)}$]

2.1.1.2 Decoding of linear block codes:

We can observe from Fig. 2.2 that as a consequence of its transmission through a noisy channel, a codeword could be received containing some errors. The received vector can therefore be different from the corresponding transmitted codeword, and it will be denoted as r = (r_1, r2,……,r_n). An error event can be modeled as an error vector or error pattern e = (e_1, e_2, ……., en) where e = r + c

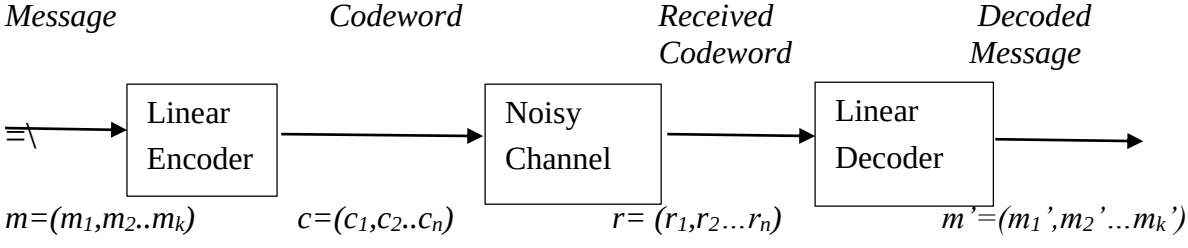


Figure 2-2 Diagram of a block coding system

To detect the errors, we use the fact that any valid codeword should obey the condition $\mathbf{cH^T = 0}$. An error-detection mechanism is based on the above expression, which adopts the following form: $\mathbf{s = r*H^T}$, where s = (s_1; s_2, ….. s_n) is called the syndrome vector. The detecting operation is performed over the received vector.

- If s is the all-zero vector, the received vector is a valid codeword.
- Otherwise, there are errors in the received vector. The syndrome array is checked to find the corresponding error pattern e_j for j = 1, 2, ..., n and the decoded message is obtained by m'= r + e_j .

2.1.2 Definition of LDPC codes

LDPC codes are linear block codes that can be denoted as (n, k) or (n, w_c, w_r), where n is the length of the codeword, k is the length of the message bits, w_c is the column weight (i.e. the number of nonzero elements in a column of the parity-check matrix), and w_r is the row weight (i.e. the number of nonzero elements in a row of the parity-check matrix).

There are two obvious characteristics for LDPC codes:

- ***Parity-check***
  LDPC codes are represented by a parity-check matrix H, where H is a binary matrix that, must satisfy $cH^T = 0$, where c is a codeword.
- ***Low-density***
  H is a sparse matrix (i.e. the number of '1's is much lower than the number of '0's). It is the sparseness of H that guarantees the low computing complexity.

2.1.3 Tanner graphs

Besides the general expression as an algebraic matrix, LDPC codes can also be represented by a bipartite Tanner graph, which was proposed by Tanner in 1981 [2].

The Tanner graph consists of two sets of vertices: n vertices for the codeword bits (called variable nodes), and k vertices for the parity-check equations (called check nodes). An edge joins a variable node and a check node if that bit is included in the corresponding parity-check equation and so the number of edges in the Tanner graph is equal to the number of ones in the parity-check matrix.

Cycle

A cycle (loop) in a Tanner graph is a sequence of connected vertices which starts and ends at the same vertex in the graph, and which contains other vertices no more than once. The length of a cycle is the number of edges it contains. Since Tanner graphs are bipartite, every cycle will have even length [12].

Girth

The girth is the minimum length of the cycles in their Tanner graph. We will illustrate the cycle and girth by a simple example. Let H be the parity-check matrix of an irregular (10, 5) LDPC code:

$$H = \begin{array}{c} \\ c1 \\ c2 \\ c3 \\ c4 \\ c5 \end{array} \begin{array}{c} \begin{array}{cccccccccc} v1 & v2 & v3 & v4 & v5 & v6 & v7 & v8 & v9 & v10 \end{array} \\ \begin{bmatrix} 1 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \end{bmatrix} \end{array} \tag{2.6}$$

The corresponding Tanner graph is illustrated in Fig. 2.3. For the LDPC code defined above, the path (c1→v8→c3→v10→p1) with the black bold lines is a cycle of length 4. This cycle is also the girth of this graph since it is the smallest cycle length.

This structure is crucial for the performance of LDPC codes. LDPC codes use an iterative decoding algorithm based on the statistical independence of message transitions between the different nodes. When there exists a cycle, the message generated from one node will be passed back to itself, thus negating the assumption of independence, so that the decoding accuracy is impacted. Therefore, it is desirable to obtain matrices with high girth values.

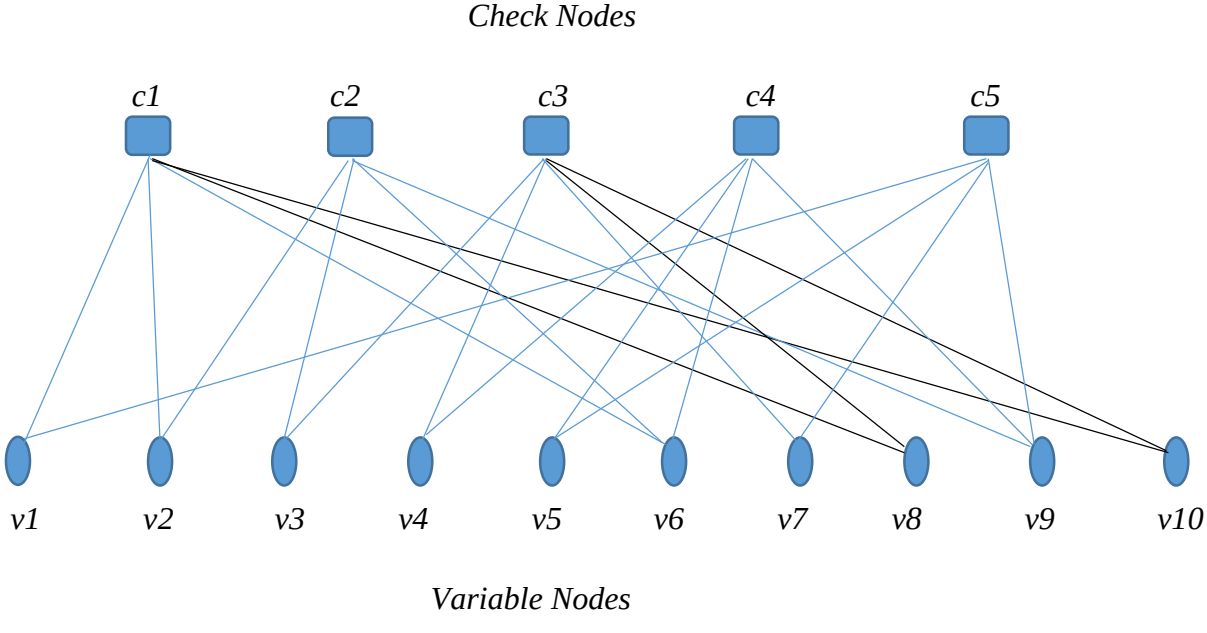


Figure 2-3 Tanner graph corresponding to the parity check matrix H in (2.6)

2.1.4 Regular and irregular LDPC codes

2.1.4.1 Regular codes

The conditions to be satisfied in the construction of the parity-check matrix H of a binary regular LDPC code are:

• The corresponding parity-check matrix H should have a fixed column weight w_c.

• The corresponding parity-check matrix H should have a fixed row weight w_r.

• The number of "l"s between any two columns is no greater than 1.

• Both w_c and w_r should be small numbers compared to the code length n and the number of rows in H.

Normally, the code rate of LDPC codes is $R = 1- (w_c/w_r)$.

2.1.4.2 Irregular codes

An irregular LDPC code has a parity-check matrix H that has a variable w_c or w_r. In general, the bit error rate (BER) performance of irregular LDPC codes is better than that of regular LDPC codes [22].

2.1.4.3 Degree distribution

In general, we want the length L of each cycle to satisfy $L \geq 4$, and L is a multiple of 2 [12]. The basic structure of an LDPC code is defined by its degree distribution [23], which are two polynomials that give the fraction of edges in the graph that are connected to the check-nodes and the variable-nodes, respectively. We call them degree distribution polynomials, denoted by $\gamma(x)$ and $\rho(x)$, respectively.

$$\gamma(\mathrm{x}) = \sum_{i=1}^{d_v} \gamma_i x^{i+1} \qquad (2.7)$$

Where γ_i corresponds to the fraction of edges connected to variable nodes and d_v denotes the maximum variable node degree. Similarly,

$$\rho(\mathrm{x}) = \sum_{i=1}^{d_c} \rho_i x^{i-1} \qquad (2.8)$$

Where ρ_i corresponds to the fraction of edges connected to check nodes and d_c denotes the maximum check node degree.

2.2 Construction of LDPC codes

The most obvious method for the construction of LDPC codes is via constructing a parity-check matrix with the properties described in the previous section. A larger number of construction designs have been researched and introduced in the literature; for example, see [1] and [13]. LDPC code construction is based on different design criteria to implement efficient encoding and decoding, in order to obtain near-capacity performance.

Several methods for constructing good LDPC codes can be summarized into two main classes: random and structural constructions. Normally, for long code lengths, random constructions [7], [14] of irregular LDPC codes have been shown to closely approach the theoretical capacity limits for the additive white Gaussian noise (AWGN) channel. Generally, these codes outperform algebraically constructed LDPC codes. But because of their long code length and the irregularity of the parity-check matrix, their implementation becomes quite complex.

On the other hand, for short or medium-length LDPC codes, the situation is different. Irregular constructions are generally not better than regular ones, and graph-based or structured constructions can outperform random ones [15].

Structured constructions of LDPC codes can be decomposed into two main categories. The first category is based on finite geometries [7], while the second category is based on circulant permutation matrices. In this thesis, we will focus on the second category and study a fast efficient encoding algorithm based on a matrix having an approximate triangular form [7], [16], which has been adopted in the WiMAX standard.

2.2.1 Gallager codes

The original LDPC codes presented by Gallager [10], [11] are regular LDPC codes and are defined by a banded structure in H. Let

$$H = \begin{bmatrix} H_1 \\ H_2 \\ . \\ . \\ H_{wc} \end{bmatrix} \qquad (2.9)$$

Where the sub matrix H_i has the following structure: for any integers μ and w_r that are greater than 1, each sub matrix H_i is μ x w_r with row weight w_r and column weight 1. For sub matrix H_1 the i^{th} row (i = 1,2,... , μ) contains all of its w_r 1 's in columns $(i - 1)\, w_r + 1$ to w_r. The other sub-matrices are simply column permutations of H1. It is easy to show that H is regular with fixed row and column weights w_r and w_c, respectively. The absence of 4 cycles in H is not guaranteed, but they can be avoided via computer design of H [1], [17].

2.2.2 Quasi-cyclic (QC) LDPC codes

Compared with randomly constructed LDPC codes, the quasi-cyclic (QC) LDPC codes are a category of structured constructions with girth of at least 6 which can be encoded in linear time with shift registers. QC-LDPC codes are well known for their low encoding complexity and low memory requirement, while preserving a high error correcting performance [18].

The QC-LDPC codes are characterized by their parity-check matrix consisting of small square blocks which are zero matrices or circulant permutation matrices [16], [18]. Assume that a QC-LDPC code has column-size n and row-size m that are multiples of an integer q. Let P^i be the q x q circulant permutation which shifts the identity matrix I to the right i times for any integer i, $0 < i < q$. For simplicity of notation, P^{∞} denotes the all-zero matrix.

Let the parity-check matrix H be the m_q x n_q matrix defined by

$$H=\begin{bmatrix} P^{a}_{11}\ P^{a}_{12}\ \ldots\ldots\ldots\ldots\ P^{a}_{1(n-1)}\ P^{a}_{1n} \\ P^{a}_{21}\ P^{a}_{22}\ \ldots\ldots\ldots\ldots\ P^{a}_{2(n-1)}\ P^{a}_{2n} \\ \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \\ \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \\ P^{a}_{m1}\ P^{a}_{m2}\ \ldots\ldots\ldots\ldots\ldots P^{a}_{m(n-1)}\ P^{a}_{mn} \end{bmatrix} \quad (2.10)$$

Where $a_{i,j} \in \{0,1,\ldots\ldots q\text{-}1,\infty\}$ .H has full rank, its codeword size is $N = nq$ and information bit size is $M = (n\text{-}m)q$. Therefore, its code, rate is given by

$$R=\frac{qn-qm}{qm}=\frac{n-m}{m}=1-\frac{m}{n} \quad (2.11)$$

Thus, we can obtain larger size block LDPC codes by increasing the size of the circulant permutation matrices P^{i} which are element matrices of *H*. Hence, this method enables an efficient implementation of the encoder. The required memory for storing the parity-check matrix of the QC-LDPC codes can be reduced by a factor *1/q,* as compared to randomly constructed LDPC codes.

2.3 Encoding of LDPC codes

Regardless of their many advantages, the encoding of LDPC codes can be an obstacle for their commercial applications, since they have high encoding complexity and encoding delay. The encoding for LDPC codes basically comprises two tasks:

• Construct a sparse parity-check matrix.

• Generate codewords using this matrix.

2.3.1 Conventional encoding based on Gauss-Jordan elimination

The conventional encoding algorithm is based on Gauss-Jordan elimination and re-ordering of columns to calculate the codeword. Similar to the general method of encoding linear block codes, Neal has proposed a simple scheme [19]. For a given codeword c and an *m x n* irregular parity-check matrix H, we partition the codeword c into message bits, *x*, and check bits, p.

$$\mathbf{c = [x|p]} \quad (2.12)$$

After Gauss-Jordan elimination, the parity-check matrix H is converted to systematic form and then divided into an *m x (n -m)* matrix A on the left and an *m x m* matrix B on the right.

$$\mathbf{H = [A|B]} \tag{2.13}$$

From the condition that for all code words $cH^T = 0$, we have

$$\mathbf{Ax^T + Bp^T = 0} \tag{2.14}$$

Hence,

$$\mathbf{p^T = B^{-1}Ax^T} \tag{2.15}$$

So (2.15) can be used to compute the check bits as long as *B* is non-singular and not just when A is an identity matrix (*H* in a systematic form). In general, the parity-check matrix H will not be sparse after the pre-processing. Thus the complexity of conventional methods for the encoding of LDPC codes is high.

2.3.2 Lower Triangular Based Encoding

A first approach in (MacKay, Wilson, and Davey 1999) is to create a parity check matrix with an almost lower-triangular shape, as depicted on figure 2.4-(a). The performance is a little bit affected by the lower-triangular shape constraint. Instead of computing the product $c = uG^T$, the equation $H.c^T = 0$ is solved, where c is the unknown variable. The encoding is systematic:

$$\{c_1, \cdots, c_{N-M}\} = \{u_1, \cdots, u_{N-M}\} \tag{2.16}$$

The next *M1* ci are recursively computed by using the lower-triangular shape:

$$c_i = -pc_i \times (c_1, \cdots, c_{i-1})^T, \text{ for } i \in \{N - M + 1, \cdots, N - M + M1\} \tag{2.17}$$

The last $M - M1$ c_i, $i \in \{N - M + M1 + 1, \cdots, N\}$ have to be solved without reduced complexity. Thus, the higher *M1* is, the less complex the encoding is. In (Richardson and Urbanke 2001) T. Richardson and R. Urbanke propose an efficient encoding of a parity check matrix H. It is based on the shape depicted on figure 2.4-(b). They also propose some "greedy" algorithms which transform any parity check matrix H into an equivalent parity check matrix H' using columns and rows permutations,

minimizing *g*. So H' is still sparse. The encoding complexity scales in *O(N + g2)* where *g* is a small fraction of *N*.

As a particular case the authors of (Bond, Hui, and Schmidt 2000) and (Hu, Eleftheriou, and Arnold 2001) construct parity check matrices of the same shape with $g = 0$.

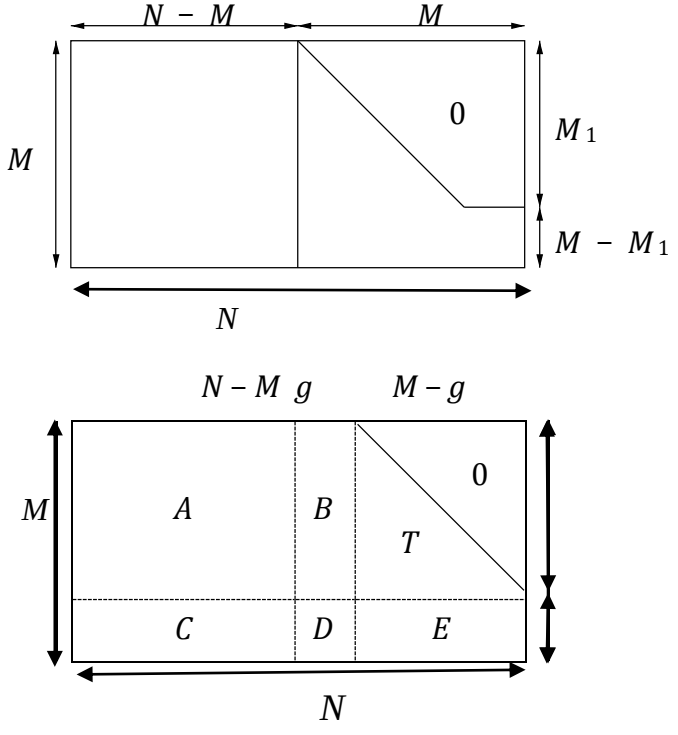


Figure 2-4 Shape of parity check matrices for efficient encoding, by MacKay et al. (MacKay, Wilson, and Davey 1999) (a) and Richardson et. al. (Richardson and Urbanke 2001) (b)

2.3.3 Other encoding schemes

Iterative encoding

In (Haley, Grant, and Buetefuer 2002), the authors derived a class of parity check codes which can be iteratively encoded using the same graph-based algorithm as the decoder. But for irregular cases, the codes does not seem to perform as well as random ones.

Low-density generator matrices

The generator matrices of LDPC codes are usually not sparse, because of the inversion. But if H is constructed both sparse and systematic, then:

$$\boldsymbol{H = (P, I_M)} \text{ and } \boldsymbol{G = (I_{N-M}, P^t)} \quad (2.18)$$

Where G is a sparse generator matrix (LDGM) (Oenning and Moon 2001): they correspond to parallel concatenated codes. They seem to have high error floors (Mackay 1999) (asymptotically bad codes). Yet, the authors of (Garcia-Frias and Zhong 2003) carefully chose and concatenate the constituent codes to lower the error floor. Note that this may be a drawback for applications with high rate codes.

Cyclic parity-check matrices

The most popular codes that can be easily encoded are the cyclic or pseudo-cyclic ones. In (Okamura 2003), a Gallager-like construction using cyclic shifts enables to have a cyclic based encoder, like in (Hu, Eleftheriou, and Arnold 2001). Finite geometry or BIBDs constructed LDPC codes are also cyclic or pseudo-cyclic (Kou, Lin, and Fossorier 2001; Ammar et al. 2002; Vasic 2002). Table 1 gives a summary of the different encoding schemes.

Encoding scheme	Description	Comments
Generator matrix product	$H \Rightarrow G$; $c = uG^t$	Use sparse generator matrices (LDGM). Bad error floor
Triangular system Solving	Using Back-Substitution as much as possible	High complexity post processing
Iterative encoding	Solve $Hc^t = 0$ using the Sum-product algorithm	Such iterative encodable codes seem to have weak performance.
Cyclic encoding	Multiplications with a shift register	Few constructions

Table 1 Summary of the different LDPC encoding schemes

2.4 Iterative Decoding Algorithm

2.4.1 Overview of Different Decoding Algorithms

In addition to introducing Low-Density Parity Check (LDPC) code in 1960[], Gallagar also developed a decoding algorithm that is near optimal and after that there has been many modifications to that algorithm and has been developed independently for different types of applications. The algorithm iteratively computes the distribution of variables in graph-based models and comes under different names depending upon applications. These algorithms are Sum-Product Algorithm (SPA), Belief Propagation Algorithm (BPA), and Message Passing Algorithm (MPA). All types of algorithm are types of "*message-passing*". There are types which are modified versions of these types of algorithms such as Min-Sum Algorithm (MSA)

Much like optimal (maximum *a posteriori*, MAP) symbol by symbol decoding of trellis code we are interested in computing the *a posteriori* probability (APP) that a given bit in the transmitted codeword $c= [c_0\ c_1\ c_2\ c_3 \ldots . c_{n-1}]$ equals '1', given the received word $y= [y_0\ y_1\ y_2\ \ldots . y_{n-1}]$. Without loss of generality, let us focus on the decoding of bit c_i so that we are interested in computing the APP

$$P_r(c_i=1 \mid y)$$

or the APP ratio (also called the likelihood ratio, LR)

$$l(c_i) \cong \frac{Pr(ci=0 \mid y)}{Pr(ci=1 \mid y)}$$

Later we will extend this to the more numerically stable computation of the log-APP ratio, also called as log-likelihood ratio (LLR)

$$l(c_i) \cong log\left(\frac{Pr(ci=0 \mid y)}{Pr(ci=1 \mid y)}\right) \qquad (2.19)$$

Where here and in the sequel the natural logarithm is assumed

The MPA for the computation of $P_r(c_i=1 \mid y)$, $l(c_i)$ is an iterative algorithm which is based on the code of tanner graph. Specifically, we imagine that the v-nodes represent processors of one type, c-

nodes represent processors of another type, and the edges represent message paths. In one half iteration, each v-node processes its input message and passes its resulting output messages up to neighboring c-nodes (two nodes are said to be neighbors if they are connected by an edge). The information passed concerns $P_r(c_0=b \mid \textit{input messages}),\ b \in \{0,1\}$, the ratio of such probabilities. In the figure 2.5 the information passed to c-node v_2 is all the information available to v-node c_0 from the channel and through the neighbors, excluding c-node v_2 ; that is only *extrinsic information* is passed . Such *extrinsic information 'm$_{ij}$'* is computed for each connected v-node/c-node pair c_i / v_i at each half iteration

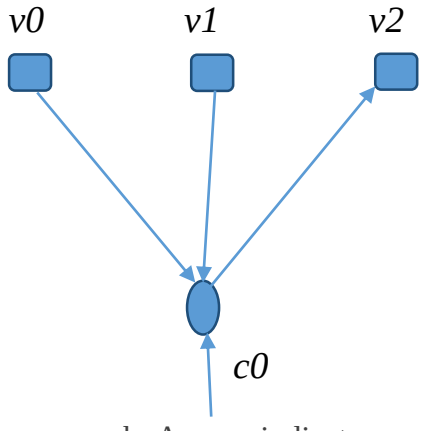


Figure 2-5 Sub graph of a tanner graph; Arrows indicate message passing between c-node to v-node

In the other half iteration, each c-node processes its input messages and passes its resulting output messages down to its neighboring v-nodes. As previous case only extrinsic message is passed to v-node. Such extrinsic information is computed for each connected c-node/v-node pair v_j/c_i at each half iteration.

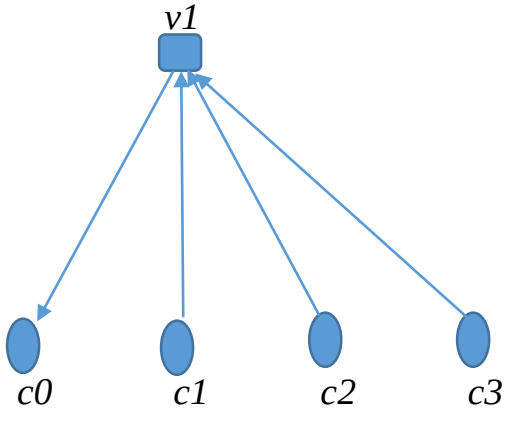


Figure 2-6 Sub graph of Tanner graph, showing message passed from c-node to v-node

After prescribed number of iterations or after some stopping criteria has been met, the decoder computes the *AAP, LR or LLR* and the decision on the bits c_i are made. The stopping criteria include verifying the codeword for validation $cH^T=0$, where c is a code-word.

The MPA assumes that the messages passed are spastically independent throughout the decoding process. When the y_i are independent, this independence assumption would hold true id the Tanner graph possessed no-cycles. Still if we can avoid the length 4-cycle we can see that message passing algorithm is showing effectiveness. The cycle of an H-matrix is called as girth of the matrix called as girth. We will now proceed to discuss about individual algorithms and their application in to different channels like BSC channels, BEC channels, BI-AWGN channels and study their algorithm.

2.4.2 Probability-Domain SPA Decoder

We start by introducing some notation

V_j: V-nodes connected to check-node 'v_j'

$V_j \backslash i$: V-nodes connected to check-node 'v_j'\ v-node 'c_j'

C_i= c-nodes connected to v-node c_i

$C_i \backslash j$= c-nodes connected to v-node c_i\c-node v_j

$Mc(\sim j))$=Messages from all check-node except node v_j

$Mv(\sim i))$=Messages from all variable-node except node c_i

$P_i=P_r(c_i=1 \mid y_i)$

S_i=Event that the check equations involving c_i are satisfied

$q_{ij}(b)=P_r(c_i=b \mid S_i, y_i, M_c(\sim j))$, where $b \in \{0,1\}$. For the APP algorithm presently under consideration $m_{ij}=q_{ij}(b)$; for the LR algorithm , $m_{ij}=q_{ij}(0)/q_{ij}(1)$; and for the LLR algorithm $m_{ij}=\log[q_{ij}(0)/q_{ij}(1)]$.

$r_{ji}(b)=P_r$(check equation f_j is satisfied $\mid c_i=b$, $M_v(\sim i)$, where $b \in \{0,1\}$. For the APP algorithm presently under consideration $m_{ji}=r_{ji}$ (b); for the LR algorithm $m_{ji} = r_{ji}(0)/r_{ji}(1)$;and for the LLR algorithm $m_{ji}=\log[r_{ji}(0)/r_{ji}(1)]$.

Note that messages $q_{ij}(b)$, while interpreted as probabilities here , are random variables(rv) as they are functions of rv y_i and other messages which are themselves rv. Similarly by virtue of the message passing algorithm, the messages $r_{ji}(b)$ are rv.

Consider now the form of $q_{ij}(0)$ which , given our new notation and the independence assumption , we may express as (see Fig 2.7)

$$q_{ij}(0) = P_r(c_i=0 \mid y_i, S_i, M_c(\sim j))$$

$$= (1-P_i)P_r(S_i \mid c_i=0, y_i, M_c(\sim j))/P_r(S_i))$$

$$= K_{ij}(1-P_i)\prod_{j' \in ci \setminus j} r_{j'i}(1) \qquad (2.20)$$

Where we used Bayes' rule twice to obtain the second line and the independence assumption to obtain the third line

$$q_{ij}(1) = K_{ij}P_i \prod_{j' \in ci \setminus j} r_{j'i}(1) \qquad (2.21)$$

The constraints K_{ij} are chosen to ensure that $q_{ij}(0) + q_{ij}(1) = 1$.

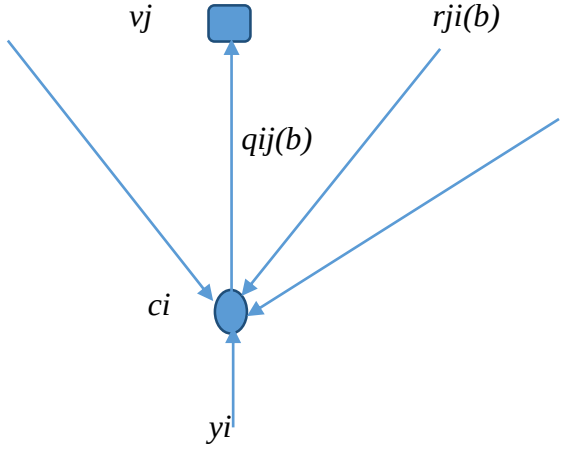


Figure 2-7 Illustration of message passing half-iteration for the computation *qij* (b) .

To develop an equation for $r_{ji}(b)$,we need the following result

Result 1 Gallagar considered a sequence of M independent binary digits a_i for which $P_r(a_i=1)=p_i$. Then the probability that $\{a_i\}_{i=1}^{M}$ contains an even number of '1's is

$$\frac{1}{2}+\frac{1}{2}\sum_{l=i}^{M}(1-2p_i) \qquad (2.22)$$

Proof: Induction on M

In view of this result, together with the correspondence $p_i \leftrightarrow q_{ij}(1)$,we have (see Fig 2.8)

$$r_{ji}(0)=\frac{1}{2}+\frac{1}{2}\sum_{i'\in V_{j\backslash i}}^{M}(1-2q_{i'j}(1)) \qquad (2.23)$$

Since, when $c_i=0$ the bits must contain even number of '1's to check equation v_j to be satisfied. Clearly,

$$r_{ji}(1)=1-r_{ji}(0) \qquad (2.24)$$

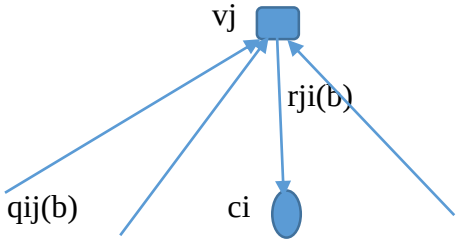


Figure 2-8 Illustration of message passing half –iteration for the computation of *rji(b)*

The MPA of the computation of the APP's initialized by setting $q_{ij}(b)= P_r(c_i=b \mid y_i)$ for all i,j for which $h_{ij}=1$, where y_i represents channel symbol that is actually received. Now we will consider some channel cases.

BEC Channel:

In this case, $y_i \in \{0,1,E\}$ where E is the erasure symbol , and we define $\delta = P_r(y_i = E \mid c_i = b)$ to be the erasure probability. Then it is easy to see that

$$P_r(c_i = b \mid y_i) = \begin{cases} 1 \; when \; y_i = b \\ 0 \; when \; y_i = b^c \\ 1/2 \; when \; y_i = E \end{cases} \qquad (2.29)$$

Where b^c represents compliment of b

BSC Channel:

In this case, $y_i \in \{0, 1\}$ and we define $\in = P_r(y_i=b^c \mid c_i=b)$ to be the error probability. Then it is obvious that

$$P_r(c_i = b \mid y_i) = \begin{cases} 1-\in \ when\ y_i = b \\ \in \ when\ y_i = b^c \end{cases} \quad (2.30)$$

Bi-AWGN Channel:

For Bi-AWGN case we can easily show that the error probability that.

$$P_r(c_i = b \mid y_i) = [1 + exp(-2yx/\sigma^2)]^{-1} \quad (2.31)$$

Summary of the Probability-Domain SPA Decoder

1. For $i=0,1.....n-1$, set $P_i=P_r(c_i=1 \mid y_i)$ where y_i is the i-th received channel symbol. Then set $q_{ij}(0)= 1-P_i$ and $q_{ij}(1)= P_i$ for all i,j for which $h_{ij} =1$.

2. Update $\{r_{ji}(b)\}$ using equations (2.23) and (2.24).

3. Update $\{q_{ji}(b)\}$ using equations (2.20) and (2.21). Solve for constant K_{ij}.

4. For $i=0,1,.....,n-1$,compute

$$Q_i(0) = K_i(1 - P_i)\prod_{j \in C_i} r_{ji}(0) \quad (2.32)$$

and

$$Q_i(1) = K_i(P_i)\prod_{j \in C_i} r_{ji}(1) \quad (2.33)$$

Where the constants K_i are chosen so that $Q_i(0) + Q_i(1)=1$

5. For *i=0,1,2…n-1* set

$$c_i = \begin{cases} 1 \;\; if \;\; Q_i(1) > \; Q_i(0) \\ 0 \qquad\qquad\quad else \end{cases}$$

6. If $cH^T=0$, then c is the valid code word else go to step 2

2.4.3 Log-Domain SPA Decoder:

In case of probability domain SPA decoder, mostly multiplication is involved, but in case of log domain decoder multiplication is replaced with addition as log-function is used instead of normal function, hence complexity of implementation reduces. Now we will declare some of the LLRs.

$$L(c_i) = \log\left(\frac{P_r(c_i=0 \mid y_i)}{P_r(c_i=1 \mid y_i)}\right)$$

$$L(r_{ji}) = \log\left(\frac{r_{ji}(0)}{r_{ji}(1)}\right)$$

$$L(q_{ij}) = \log\left(\frac{q_{ij}(0)}{q_{ij}(1)}\right)$$

$$L(Q_i) = \log\left(\frac{Q_i(0)}{Q_i(1)}\right)$$

The initialization steps for the three channels under consideration will now be

$$L(q_{ij}) = L(c_i) = \begin{cases} +\infty\,, y_i = 0 \\ -\infty, y_i = 1 \quad (BSC) \\ 0, y_i = E \end{cases} \tag{2.34}$$

$$L(q_{ij}) = L(c_i) = (-1)^{y_i} \log(\tfrac{1-\epsilon}{\epsilon}) \quad (BEC) \tag{2.35}$$

$$L(q_{ij}) = L(c_i) = 2y_i/\sigma^2 \tag{2.36}$$

For step 1, we first replace $r_{ji}(0)$ with $1\text{-}r_{ji}(1)$ in equation (2.24) and rearrange it as

$$1 - r_{ji}(1) = \prod_{i' \epsilon V_j} \left(1 - 2q_{i'j}(1)\right)$$

Now using the fact that $tanh\,[\frac{1}{2}log(p_0/p_1] = p_0\text{-}p_1=1\text{-}2p_1$, we may rewrite the equation above as

$$tannh\left(\frac{1}{2}L(r_{ji})\right) = \prod_{i\,\epsilon V_j\backslash i}\left(\frac{1}{2}L(q_{i\prime j})\right) \qquad (2.37)$$

The problem with these expressions is that we are still left with a product and the complex *tanh* function. We can remedy this as follows .First factor $L(q_{i'j})$ in to its sign and magnitude.

$$\begin{aligned} L(q_{ij}) &= \alpha_{ij}\beta_{ij} \\ \alpha_{ij} &= sign\left[L(q_{ij})\right] \\ \beta_{ij} &= \left|L(q_{ij})\right| \end{aligned}$$

So that 4.9 can be rewritten as

$$\tanh(\frac{1}{2}L(r_{ji})) = \prod_{i'\in V_j\backslash i}\alpha_{i'j}.\prod_{i'\in V_j\backslash i}\tanh(\frac{1}{2}\beta_{i'j})$$

We then have

$$\begin{aligned} L(r_{ji}) &= \prod_{i'}\alpha_{i'j}.2\tanh^{-1}\left(\prod_{i'}\tanh(\frac{1}{2}\beta_{i'j})\right) \\ &= \prod_{i'}\alpha_{i'j}.2\tanh^{-1}\log^{-1}\log\left(\prod_{i'}\tanh(\frac{1}{2}\beta_{i'j})\right) \\ &= \prod_{i'}\alpha_{i'j}.2\tanh^{-1}\log^{-1}\sum_{i'}\log\left(\tanh(\frac{1}{2}\beta_{i'j})\right) \\ &= \prod_{i'\in V_j\backslash i}\alpha_{i'j}\phi\left(\sum_{i'\in V_j\backslash i}\phi\left(\beta_{i'j}\right)\right) \end{aligned}$$

Where we defined

$$\phi(x) = -\log\left[\tanh\left(x/2\right)\right] = \log\left(\frac{e^x+1}{e^x-1}\right)$$

and used the fact that $\phi^{-1}(x) = \phi(x)$ when $x > 0$. The function is fairly well behaved, as shown in Fig.7, and so many are implemented by a look up table.

For Step 2, we simply divide equation (4.1) by (4.2) and take the logarithm of both sides to obtain

$$L(q_{ij}) = L(c_i) + \sum_{j' \in C_i \setminus j} L(r_{j'i})$$

Step 3 is similarly modified so that

$$L(Q_i) = L(c_i) + \sum_{j' \in C_i} L(r_{j'i})$$

Summary of Log-Domain SPA Decoder

1. For $i = 0,1,.....,n-1$, initialize $L(q_{ij})$ according to (4.8) for all i, j for which $h_{ij} = 1$

2. Update $\{L(r_{ji})\}$ using equation (4.10)

3. Update $\{L(q_{ji})\}$ using equation (4.11)

4. Update $\{L(Q_i)\}$ using equation (4.12)

5. For $i = 0,1,.....,n-1$, set

$$\hat{c}_i = \begin{cases} 1 \; if \; L(Q_i) < 0 \\ 0 \qquad else \end{cases}$$

If $\hat{c}_i H^T = 0$ or the number of iterations equals the maximum limit, stop; else, go to step 2.

Remark. This algorithm can be simplified further for the BEC and BSC channels since the initial LLRs (see (4.8)) are ternary in the first case and binary in the second case. See the discussion of the min-sum decoder below.

2.4.4. Reduced Complexity Decoders

It should be clear from the above that the log-domain SPA algorithm has lower complexity and is more numerically stable than the probability-domain SPA algorithm. We know present decoders of lower complexity which often suffer only a little in terms of performance. The degradation is typically on the order of 0.5, but is a function of the code and the channel as demonstrated in the example below.

2.4.4.1. The Min-Sum Decoder

Consider the update equation (4.10) for $\{L(r_{ji})\}$ in the log-domain decoder. Note from the shape of $\phi(x)$ that the term corresponding to the smallest β_{ij} in the summation dominates, so that

$$\phi\left(\sum_{i'}\phi(\beta_{i'j})\right) \approx \phi(\phi(\min_{i'}\beta_{i'j}))$$

$$= \min_{i'\in V_j\backslash i}\beta_{i'j}$$

Thus, the min-sum algorithm is simply log-domain SPA with Step 1 replaced by

$$L(r_{ji}) = \prod_{i'\in V_j\backslash i}\alpha_{i'j}\min_{i'\in V_j\backslash i}\beta_{i'j}$$

It can also be shown that, in the BI-AWGNC case, the initialization $L(q_{ji}) = 2y_i/\sigma^2$ may be replaced by $L(q_{ji}) = y_i$ when the min-sum algorithm is employed. The advantage, of course, is that knowledge of the noise power σ^2 is unnecessary in this case.

Concluding Remarks on LDPC codes

Low density parity check codes are studied for a large variety of application, much as turbo-codes, trellis-codes and other codes were when they were first introduced to the coding community. As indicated above LDPC codes have advantage over turbo codes as

1. They allow parallelizable decoder.
2. They are more amenable to high rates.
3. They generally possess a lower error rate floor.
4. They require no interleavers in encoder and decoder.

The disadvantage of LDPC codes are

1. Most LDPC codes have complex encoders.
2. Connectivity among the decoder component will be tedious task.

Applications of LDPC codes:

1. LDPC code is used in current Broadcasting standard called DVB-S2.
2. LDPC is a strong candidate for Wi-Max standard.
3. LDPC can be used along with RS-codes for OFDM application for high data rates.
4. LDPC is a strong candidate for future communication standards such as 4G or 5G as error correcting code.

Chapter 3

LDPC coded Communication System

3.1 LDPC based Communication System

In this project we have taken a BPSK base AWGN channel in which as encoding process LDPC is used. The Block diagram of the system can be shown as below.

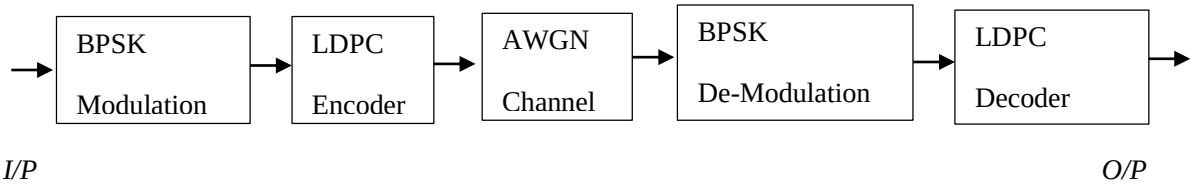


Figure 3-1 LDPC Coded BPSK Modulation in AWGN Channel

We will transmit random bits and first it will be modulated through BPSK channel and then it is LDPC encoded and then transmitted through AWGN channel and de-modulated through BPSK demodulator and then through LDPC Decoder the output is obtained.

3.2 LDPC Encoder

A low-density parity-check (LDPC) code is defined by a parity- check matrix that is sparse. A regular *(n,k)* LDPC code is defined by an *(n- k)* × *n* parity-check matrix with *n*- block length of the code and *k* information bits generated by the binary source. There are kinds of LDPC codes regular and irregular, irregular performs better than regular but regular codes are easy to implement. We will take a special case of LDPC codes as cyclic codes which is used to construct the parity check code and study the behavior.

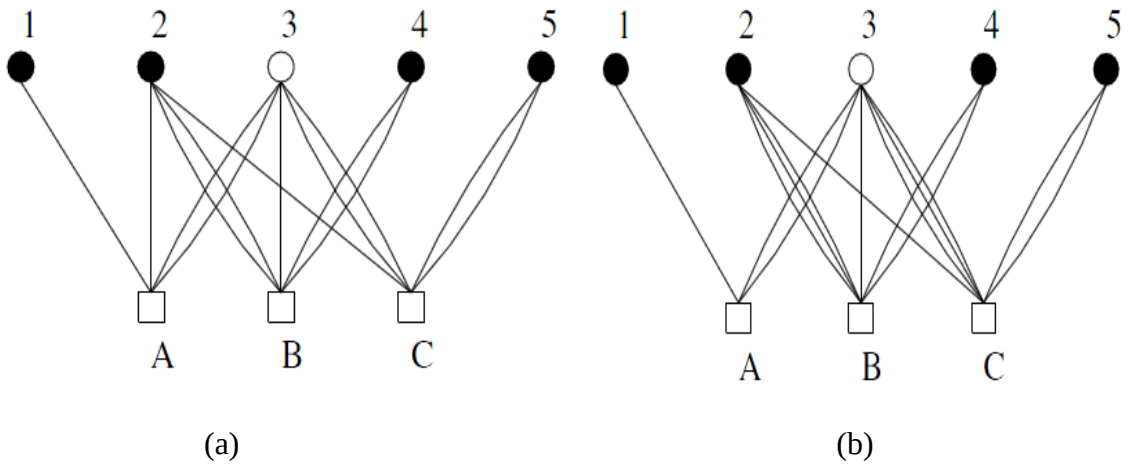
3.2.1 Construction of Parity check Matrix

Construction of parity check matrix is the important part of Encoding process. We have used the block-circulant LDPC code construction for creating parity check matrix. The main advantage for circulant code is at par error correcting performance and well-structured decoder architectures. We define a circulant as a square binary matrix where each row is constructed from the previous row by a single

right cyclic shift; we do not require that each row has Hamming weight 1. An $rT \times nT$ parity check matrix H can be constructed by concatenating $r \times n$ sparse circulants of size $T \times T$. The density of each circulant matrix is indicated by the corresponding value in an $r \times n$ base matrix H base. The Tanner graph corresponding to this matrix is called a protograph [7]. Entries greater than 1 in the base matrix correspond to multiple edges in the protograph. Base matrices can be expanded into block-circulant LDPC codes by replacing each entry in H base with a circulant containing rows of the specified Hamming weight; the resulting codes are quasi-cyclic. Alternatively, they can be expanded into less structured codes by replacing each entry with a sum of arbitrary permutation matrices weight; the resulting codes are quasi-cyclic. Alternatively, they can be expanded into less structured codes by replacing each entry with a sum of arbitrary permutation matrices.

Protographs for our AR3A and AR4A codes of rate ½ are shown in Figures 3.2(a) and (b) , and we use these as examples throughout the paper. Squares are parity check nodes and circles are variable nodes, where the solid circles represent transmitted symbols and the open ones are punctured. These designs were derived from a three step encoding procedure: accumulate, repeat-by-3 (or 4), and accumulate [8]; hence their names. Each protograph describes a 3×5 block-circulant parity check matrix, and the number of parallel edges shows the degree of the corresponding circulant.

In practice, these protographs cannot be directly expanded into block-circulant codes without introducing low weight codewords, regardless of the choice of circulants. A practical solution is to expand the protographs twice, first with small permutation matrices, such as of size 4×4 or 8×8, and then with circulants to build the full code. The result is a parity check matrix such as the one shown in Figure 3 for a very small AR4A code, where each nonzero entry in the matrix is represented by a dot. This code was constructed by putting the AR4A protograph variable nodes in the order (4, 2, 1, 5, 3) and check nodes in order (A, B, C) as demarcated by the solid lines, expanding with 4×4 permutations, and then expanding with 16×16 circulants. The resulting 12×20 block-circulant structure is emphasized by dotted lines.



(a) (b)

Figure 3-2 (a) AR3A Protograph (b) AR4A protograph

An encoder for any *(N, K)* LDPC code can be built from an erasure correcting decoder. A set of K linearly independent variable nodes are selected as the systematic symbols, and these are initialized with the *K* information bits to be encoded. If there are no stopping sets, then the remaining $N-K$ parity symbols are computed iteratively with the standard erasure correcting algorithm. Because the known symbol positions are known a priori, the existence of stopping sets is also known. This method is equivalent to Richardson and Urbanke's low complexity encoding algorithm [9] when their variable g = 0 If H has full rank R = N −K, and this iterative encoding method succeeds, then each of the N − K parity check equations is solved exactly once to determine one of the N−K unknown parity symbols. For a check equation with d terms, d − 2 exclusive-OR operations are required. Thus, iterative encoding requires exactly E − 2R exclusive-OR operations, where E is the number of nonzero elements in H. For an arbitrary LDPC code, the scheduling of these computations can be complex; for block-circulant codes, they can be performed in well organized groups of T operations. The amount of memory required in such a decoder varies depending on the code structure; it is sufficient to store all N code symbols. We illustrate these ideas with the AR3A and AR4A code examples. When the rows and columns of the AR4A base matrix are reordered as (B, A, C) and (4, 2, 3, 1, 5), we get,

$$H=\begin{bmatrix}2 & 3 & 1 & 0 & 0\\ 0 & 0 & 2 & 1 & 0\\ 0 & 1 & 3 & 0 & 2\end{bmatrix}$$

Iterative encoding begins by applying the $kT = 2T$ information symbols to the first two columns in the base matrix. The first row of *T* check equations can be solved in parallel to determine the third column of code symbols, and then the next row can be solved to determine the fourth column. The *2* in the lower

right corner means that each remaining check equation has two unknowns, and iterative encoding is halted by the stopping set. However, note that this parity check matrix is not full rank: the sum of the first *T* and last *T* rows of *H* is the all-zero vector, independent of the circulants chosen. This means that one of the remaining *T* undetermined code symbols can be assigned an additional information bit, and iterative encoding now completes successfully, operating (in a permuted order) as an accumulator of length *T*.

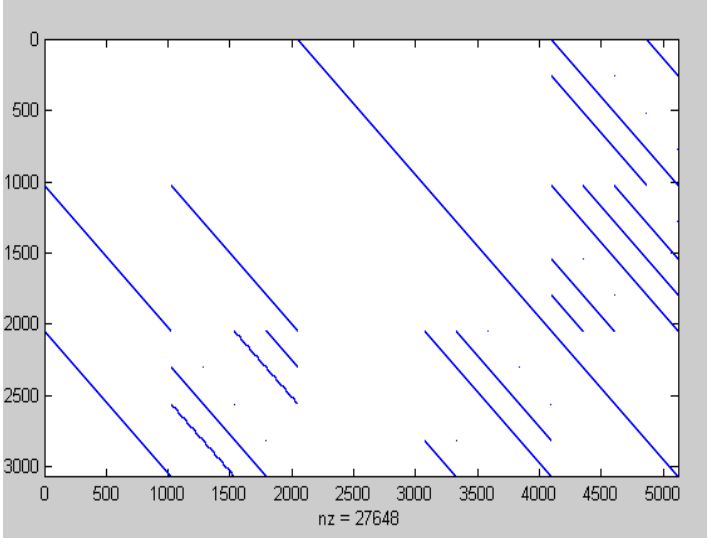


Figure 3-3 *H* Matrix used for Performance Study

3.2.2 Encoder hardware Implementation

Encoder of the block-circulant matrix is implemented using shift registers and cyclically shifting row after row. This implementation is extremely similar to a set of n−k encoders for recursive convolutional codes, each of constraint length T. With the switches set as drawn, the k message bits are fed through the encoder one at a time, and the registers are updated and shifted once per bit. Then the switches are changed and the contents of the registers are sequentially read out as the parity portion of the codeword

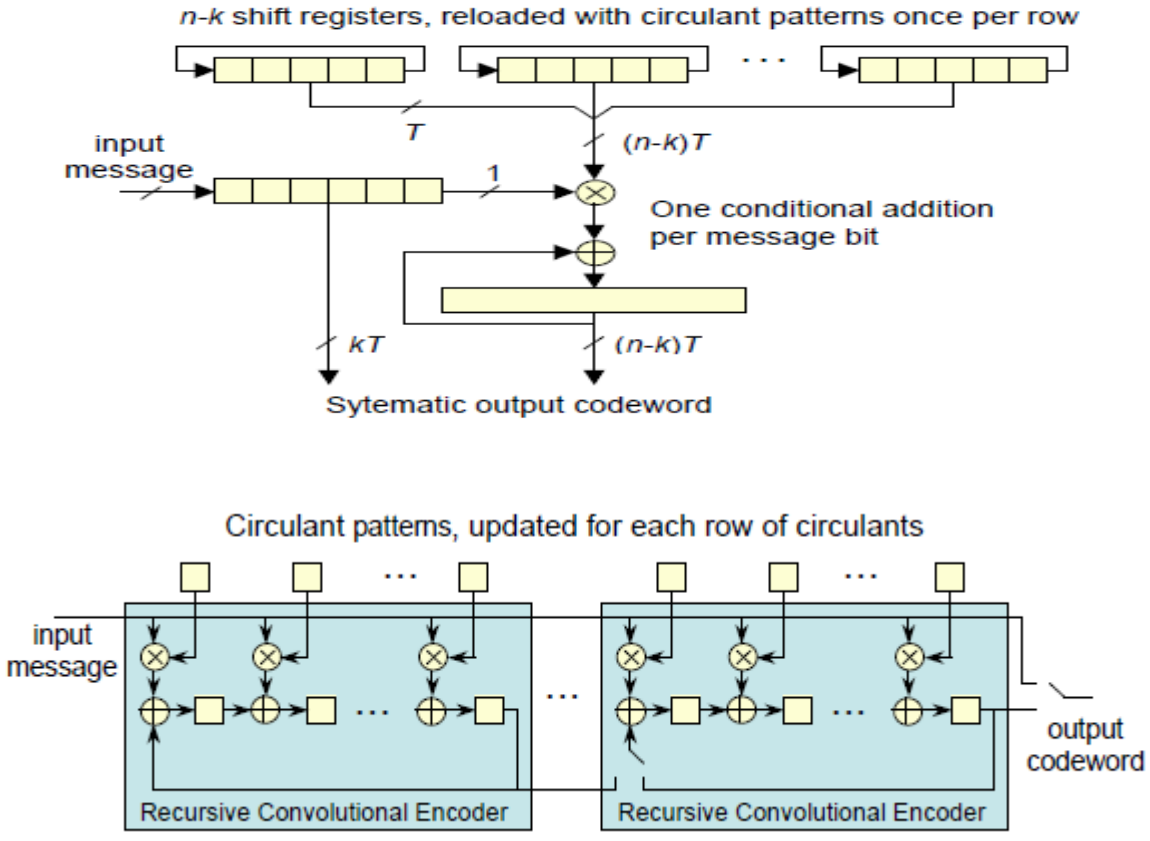


Figure 3-4 Structure of Circulant Encoder using Shift Register

3.3 LDPC Decoder:

Decoder is the most important part of LDPC system as it is required to correct bits in the block if there is any and give back. LDPC decoder is based under many algorithm that has been already discussed in chapter 2 and for Matlab simulation we have used SPA (Sum Product Algorithm) algorithm.

3.3.1 Sum Product Algorithm

As per the algorithm discussed in chapter 2, sum product algorithm is about calculate LLR, transmit and passing in between variable nodes and check nodes till the iteration is achieved or valid code word is obtained.

The sum product algorithm consists of the following steps. With the the input the LLRs for the a priori message probabilities, the parity check matrix H and the maximum number of allowed iterations *Imax.*

3.3.2.2 Steps for Sum Product Algorithm

Initialise

Set $Q_{ij} = \lambda_j$, this initialises the check nodes with the a priori message probabilities.

Update Check Messages

For each check node *j*, and for every bit node associated with it *j* compute:

$$R_{ij} = 2\ tannh^{-1} \prod_{\alpha \epsilon V(j), \alpha \neq i} \tanh(\frac{Q_{\alpha j}}{2}) \qquad (3.1)$$

Test for Valid Codeword

Make a tentative decision on codeword

$$L_i = \lambda_j + \sum_{j \epsilon c(j)} Q_{\alpha j} \qquad (3.2)$$

$$c = \begin{cases} 1\ if & L_i \leq 0 \\ 0\ if & L_i > 0 \end{cases}$$

If number of iterations is *Imax* or a valid codeword has been found ($cH^T = 0$) then finish

Update Bit Messages

For each bit node *j*, and for every check node associated with it *j* compute:

$$Q_{ij} = \lambda_j + \sum_{j \epsilon c(j)} R_{\alpha j}[k-1] \qquad (3.3)$$

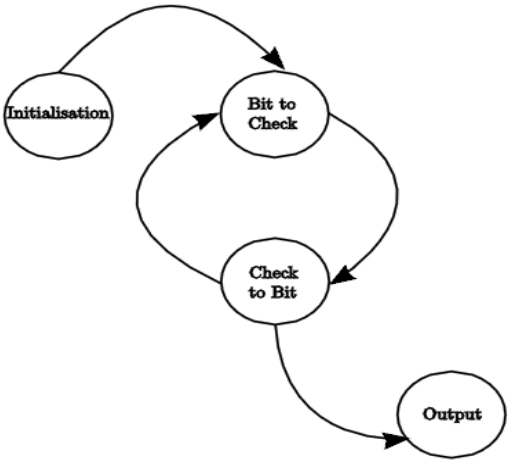


Figure 3-5 State Machine for SPA [23]

3.3 Performance of LDPC System

In this section, we will evaluate the LDPC codes specified in the WiMAX standard by performing simulations assuming binary phase-shift keying (BPSK) modulation over additive white Gaussian noise (AWGN). The iterative Sum-Product Algorithm (SPA) decoder is used for decoding. It terminates when either a valid codeword is found or the maximum of 50 iterations is reached. These simulations have been carried out using MATLAB and the parameters used in the simulation are shown in Table 2.

Modulation	BPSK
Channel	AWGN
LDPC Codes	Rate (1/2)
Encoding	Circulant Encoder(R-U Method)
Decoding	SPA
Maximum No. Of Iteration	50

Table 2 Characteristic of LDPC System under consideration

3.3.1 Performance over an AWGN channel

We have taken a normal BPSK system and compared it with a LDPC coded BPSK system and it's performance in Matlab is plotted below

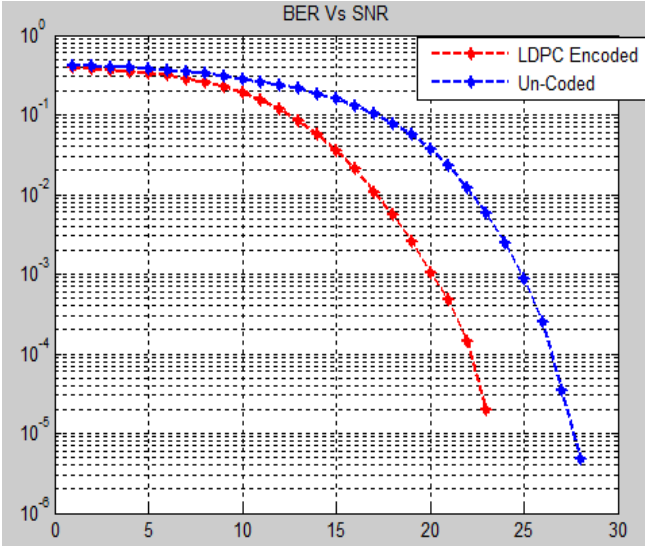


Figure 3-6 Normal BPSK vs. LDPC Coded BPSK System

Observations:

1. LDPC coded System performs better than normal BPSK system.
2. LDPC system is having at least 5dB performance improvement.

3.3.2 Different Rates of LDPC and their Performance review

We have taken different rates of *'H'* matrix and compared their performance to find out what is the impact of rates on BER performance.

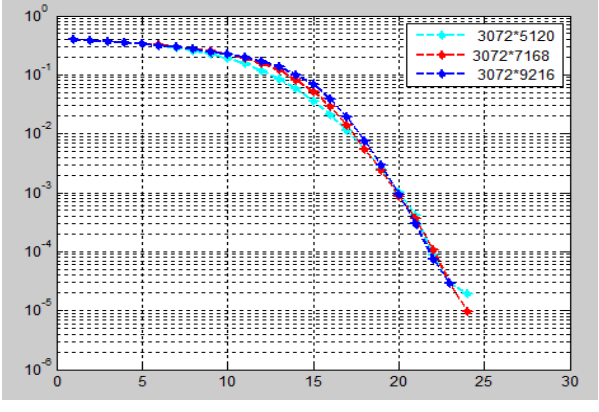


Figure 3-7 BER performance curve for Different rates of H matrix

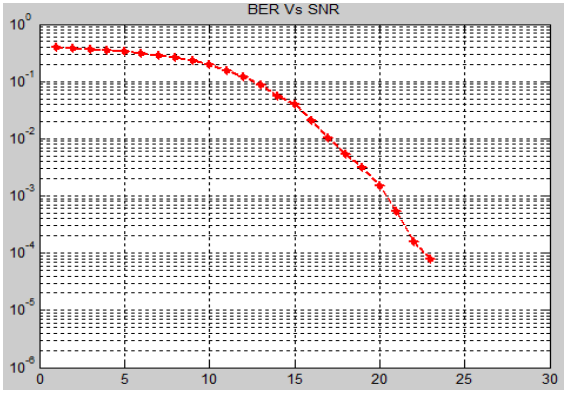
Observations:

1. With the rate of the matrix decreasing the performance seems to be decreasing.
2. There are kind of 1-2 dB performance difference between various kinds of rates of H.

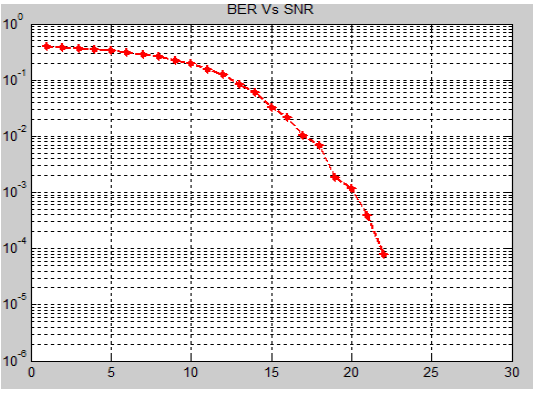
3.3.3 Performance Observation with different number of Iteration

No. of iteration is an important consideration while decoding. Because as with increasing no. of iterations error floor converges and attains a good performance. With increasing iteration it is obvious that timing of decoding increases with no. of iteration, hence we have to find an trade-off between number of iteration and SNR performance.

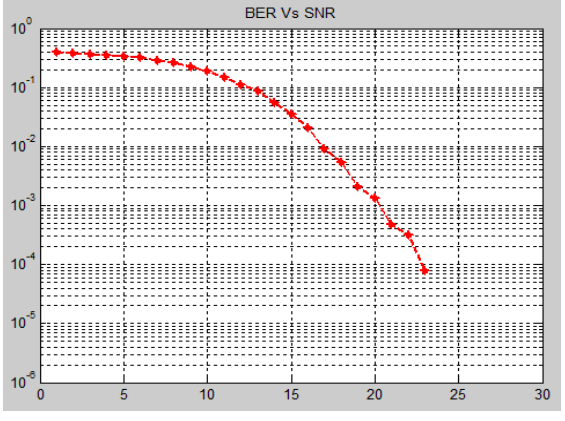
We have plotted 4 different plots for 4 types of iteration for constant parity matrix, girth and rate of decoding.



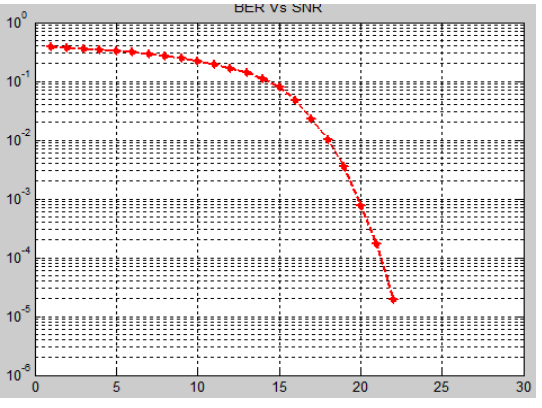
(No. Of Iteration 10)



(No. Of Iteration 20)



(No. Of Iteration 50)



(No. Of Iteration 100)

Figure 3-8 LDPC coded System with 2048X4096 matrix with varying no. of Iteration.

Observation:

1. With increasing no. of Iteration the BER vs SNR curve is becoming smoother and better BER performance.
2. The time of Decoding is going up with increasing no. of iterations.
3. For better SNR performance we have to compromise with timing constraint.

3.3.4 Time of decoding for different types of code for different kind of Rate

The simulation results show that the LDPC codes are one powerful class among the LDPC codes for FEC. We can conclude that these LDPC codes can achieve a better error performance with a greater code length and maximum number of iterations while sacrificing the output delay, since the larger the maximum number of iterations is set, the longer the decoding process will last. We will now do the VLSI implementation of the LDPC codes in FPGA and study it's behavior in detail.

We will study the time of iterations and the total time for decoding to study the time taken for the decoder for total decoding.

From the table 3 it is observed that

1. With increasing number of iteration the time of decoding is increasing.
2. With rate of *H* matrix increasing the timing is increasing.
3. With more message bit the time taken for decoding is also increasing steadily.

Message Bit	Rate	No. Of Iteration	Time Taken (in second)
8	1/2	10	33.072028
		20	33.268106
		50	34.496226
	2/3	10	33.202228
		20	34.120415
		50	36.510296
	3/4	10	34.171052
		20	34.751116
		50	38.304744
	4/5	10	34.095604
		20	35.796721
		50	40.096659
16	1/2	10	38.968651
		20	40.090315
		50	45.944988
	2/3	10	39.419841
		20	42.915639
		50	47.264385
	3/4	10	42.786026
		20	46.830233
		50	51.582669
	4/5	10	43.015565
		20	47.082823
		50	56.399975
64	1/2	10	38.205363
		20	42.350193
		50	56.067338
	2/3	10	42.474863
		20	48.977386
		50	71.414107
	3/4	10	45.135953
		20	55.140872
		50	86.926654
	4/5	10	48.914682
		20	62.058402
		50	102.91186

Table 3 Time required for Decoding for various cases

Chapter 4

FPGA Implementation of LDPC Code

4.1 VHDL Basics

VHDL is an acronym for VHSIC Hardware Description Language (VHSIC is an acronym for Very High Speed Integrated Circuits).It is a hardware description language that can be used to model a digital system at many levels of abstraction, ranging from the algorithmic level to the gate level [9].

Hardware description languages are especially useful to gain more control of parallel processes as well as to circumvent some of the idiosyncrasies of the higher level programming languages. The compilers often add latency to loops during compilation for implementation. This can be difficult to fix in the higher-level languages, though the solution may be quite obvious at the hardware description level. One particularly frustrating peculiarity is the implementation of multipliers. For all multiply commands, the complier requires three multipliers to be used, though typically one is sufficient. The compiler‘s multipliers also are intended for integers. For a fixed-point design, the decimal point must be moved after every multiply. This is much easier to implement at the hardware description level [1]

VHDL is a programming language that has been designed and optimized for describing the behavior of digital systems. VHDL has many features appropriate for describing the behavior of electronic components ranging from simple logic gates to complete microprocessors and custom chips. Features of VHDL allow electrical aspects of circuit behavior such as rise and fall times of signals, delays through gates, and functional operation to be precisely described. The resulting VHDL simulation models can then be used as building blocks in larger circuits using schematics, block diagrams or system-level VHDL descriptions for the purpose of simulation.

VHDL is also a general-purpose programming language: just as high-level programming languages allow complex design concepts to be expressed as computer programs, VHDL allows the behavior of complex electronic circuits to be captured into a design system for automatic circuit synthesis or for system simulation. Like Pascal, C and C++, VHDL includes features useful for

structured design techniques, and offers a rich set of control and data representation features. Unlike these other programming languages, VHDL provides features allowing concurrent events to be described. This is important because the hardware described using VHDL is inherently concurrent in its operation. One of the most important applications of VHDL is to capture the performance specification for a circuit, in the form of what is commonly referred to as a test bench. Test benches are VHDL descriptions of circuit stimuli and corresponding expected outputs that verify the behavior of a circuit over time. Test benches should be an integral part of any VHDL project and should be created in tandem with other descriptions of the circuit. One of the most compelling reasons for learning VHDL is its adoption as a standard in the electronic design community. Using a standard language such as VHDL virtually guarantees that the engineers will not have to throw away and recapture design concepts simply because the design entry method chosen is not supported in a newer generation of design tools. Using a standard language also means that the engineer is more likely to be able to take advantage of the most up-to-date design tools and that the users of the language will have access to a knowledge base of thousands of other engineers, many of whom are solving similar problems.

4.1.1 Capabilities of VHDL

The following are the major capabilities that the language provides along with the features that differentiated from other hardware description language [18].

- The language can be used as an exchange medium between chip vendors and CAD tool users. Different chip vendors can provide VHDL description of their components to system designers.
- The language can also be used as communication medium between different CAD and CAE tools.
- The language supports hierarchy; that is a digital system can be modelled as a set of interconnected components.
- The language supports flexible design methodologies: top-down, bottom-up, or mixed.
- The language is not technology-specific, but is capable of supporting technology-specific features. It can also support various hardware technologies.
- It supports both synchronous and asynchronous timing models.
- Various digital modelling techniques, such as finite-state machine descriptions, algorithmic descriptions and Boolean equations, can be model using the language.

- The language is publicly available, human-readable, and machine-readable and above all, it is not proprietary.
- It is an IEEE and ANSI standard; therefore model described using this language is portable.
- The language supports three basic different description styles: structural, data flow and behavioral. A design may also be expressed in any combination of these three.
- It supports a wide range of abstraction levels ranging from abstract behavioral descriptions to very precise gate-level descriptions.
- Arbitrarily large designs can be modelled using this language, and there are no limitations imposed by the language on the size of design.
- The language has elements that make large-scale design modelling easier.
- Nominal propagation delays, min-max delays, setup and hold timing, timing constraints and spike detection can all be described very naturally in this language.

4.2 VHDL Implementation of LDPC Encoder and Decoder

4.2.1 LDPC Encoder

The LDPC Encoding is about multiplication of message matrix with generator matrix. For large codes, it is very difficult to implement an LDPC Encoder with higher matrix rate. The steps for Encoding are given below.

Step 1

Parity matrix *(H)* is selected from circulant encoding. The *H* matrix is of the form

$$\mathbf{H=[-P^T:I_{n-k}]} \quad (4.1)$$

Step 2

Generator matrix (*G*) is generated from the parity matrix as

$$\boldsymbol{G=[I_k:P]} \quad (4.2)$$

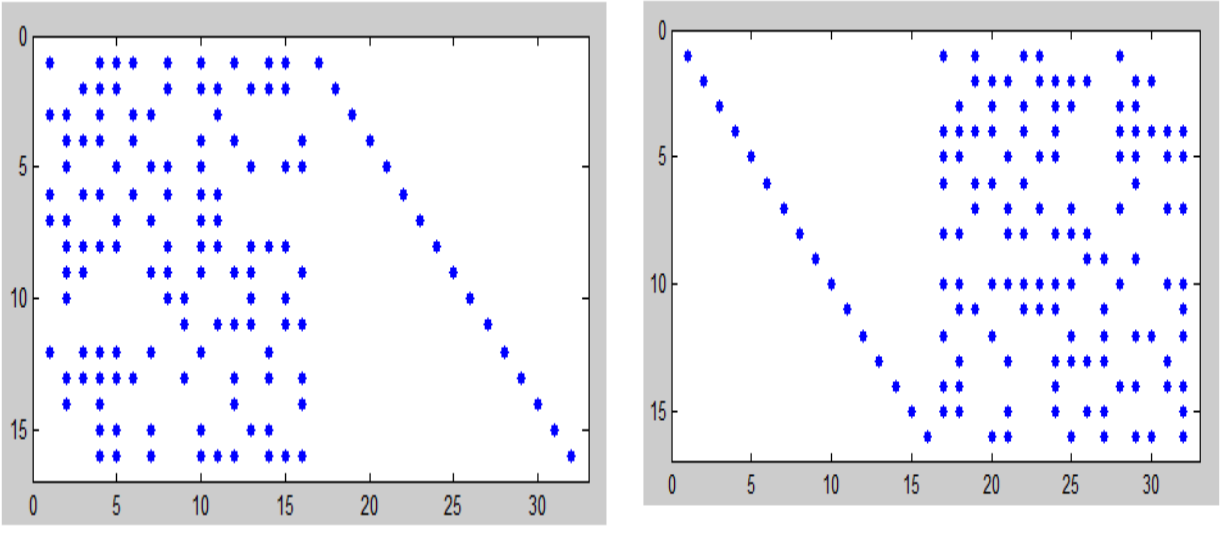
Step 3

Taking the message signal *(m)*. The transmitted codeword(c) will be

$$\boldsymbol{c=mG} \quad (4.3)$$

H is a sparse matrix of rate (1/2).Generally the H matrix is of the form 2048X4096 .Then G matrix will be of the form 2048X4096, but practical implementation of such a big code is a cumbersome task as we need to write for each link between check node and variable node. For implementation purpose we have taken a small matrix (16X32) and the same can be repeated to implement big matrix of greater size.

The 'H' matrix and 'G' matrix for implementation is given in sparse form as below.



(H-Matrix) *(G-Matrix)*

Figure 4-1 H and G Matrix used for VHDL implementation

4.2.1.1 VHDL Implementation of Encoder

The VHDL implementation involves multiplication of message matrix with Generator matrix (*G*).The generator matrix is of 16X32 dimension, so the message dimension will be of 16 bit. We will use matrix multiplication between message stream and generator matrix. We have taken a *Xilinx XC3S500E FPGA* (Spartan 3E) board for implementation of the Encoder.

Step 1

Taking the required number of registers for storing the Generator matrix. Here we require 16, 32-bit registers to store the 16X32 matrix.

Step 2

Write the matrix multiplication between message matrix (1X16) and generator matrix which is a combination of *'AND'* & *'OR'* gates.

Step 3

We will transmit the message (m) bits through "*Test Bench*".

Step 4

The output is now given out for transmission through the *AWGN* channel.

Encoder Top Level Schematic:

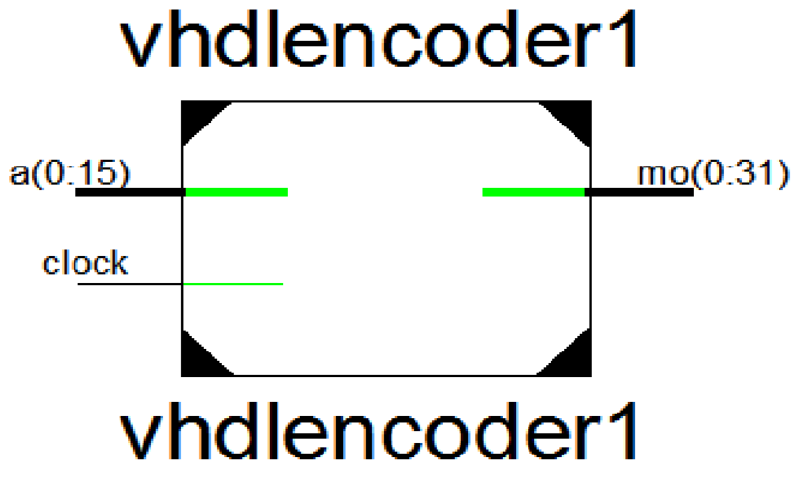


Figure 4-2 Top Level Schematic of Encoder

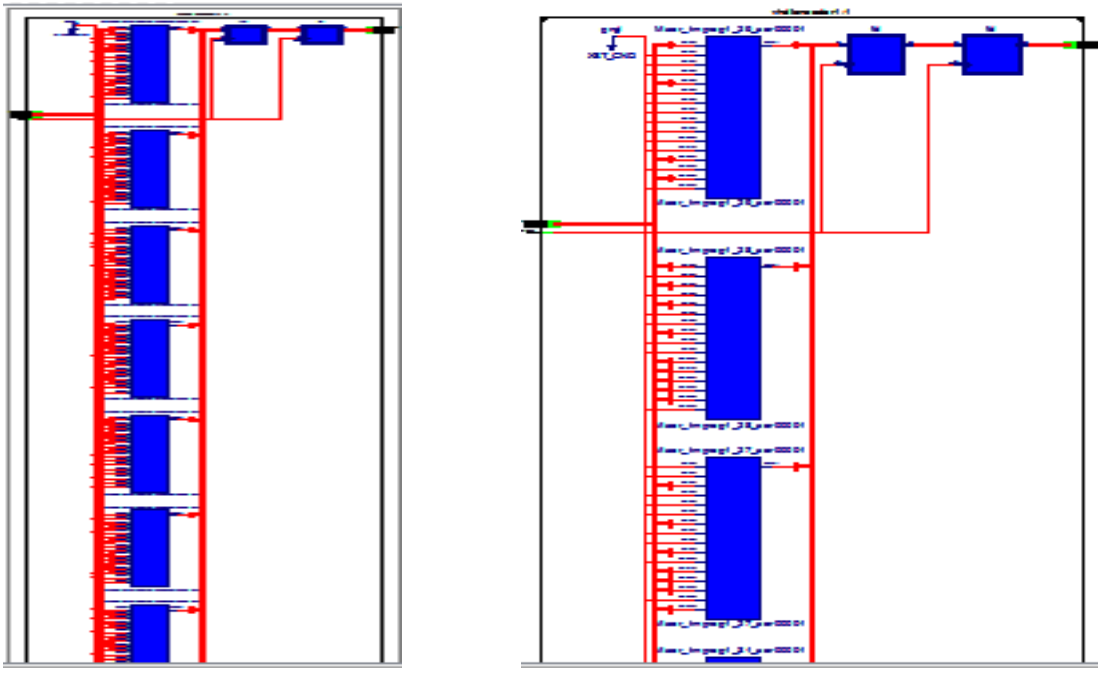
I/O Bus in Encoder

a (0:15): Message Input -16 bit

clock: Clock Signal - 1 bit

mo: Encoded Signal -32 bit

Encoder RTL Schematic



(RTL Schematic) (RTL Schematic Zoomed)

Figure 4-3 RTL Schematic of Encoder

Device Utilization

The Encoder is implemented on *Xilinx XC3S500E FPGA* (Spartan 3E) board .The device utilization are as follows.

Device Utilization Summary (estimated values)			[-]
Logic Utilization	**Used**	**Available**	**Utilization**
Number of Slices	30	960	3%
Number of Slice Flip Flops	47	1920	2%
Number of 4 input LUTs	53	1920	2%
Number of bonded IOBs	49	66	74%
Number of GCLKs	1	24	4%

Figure 4-4 Device utilization of Encoder Implementation (Spartan 3E)

4.2.2 LDPC Decoder

The decoder we have implemented it using a *Modified Sum Product Algorithm (MPSA)* [20], which is an approximation algorithm in context with normal SPA algorithm. The modified SPA is used for implementation ease of the decoder.

4.2.2.1 Modified SPA Decoder

After the encoder output is transmitted through the AWGN channel the output is given to the decoder's variable node. Let *M(n)* denote the set of check nodes connected to the symbol node n, that is, the position of ones in the nth column of *H*, and *N(m)* the set of symbol nodes participating in the *m-th* parity-check equation, that is, the position of ones in the *m-th* row of H. In addition, *M(n)\m* represents the set *M(n)*, excluding the *m-th* check node and *N(m)\n* the set *N(m)*, excluding the nth symbol node. Furthermore, $q_{n\to m}(x)$, $x\in[0, 1]$ is the message (i.e. the probability of being 0 or 1) that the symbol node n sends to check node m, based on all the checks involving n except m .Similarly, $r_{m\to n}(x)$, $x\in[0, 1]$ is the message (i.e. the probability of being 0 or 1) that the *m-th* check node sends to the n-th symbol node, based on all the symbols checked by m except n. By operating in the logarithmic domain, we define two log-likelihood ratio (LLR) values.

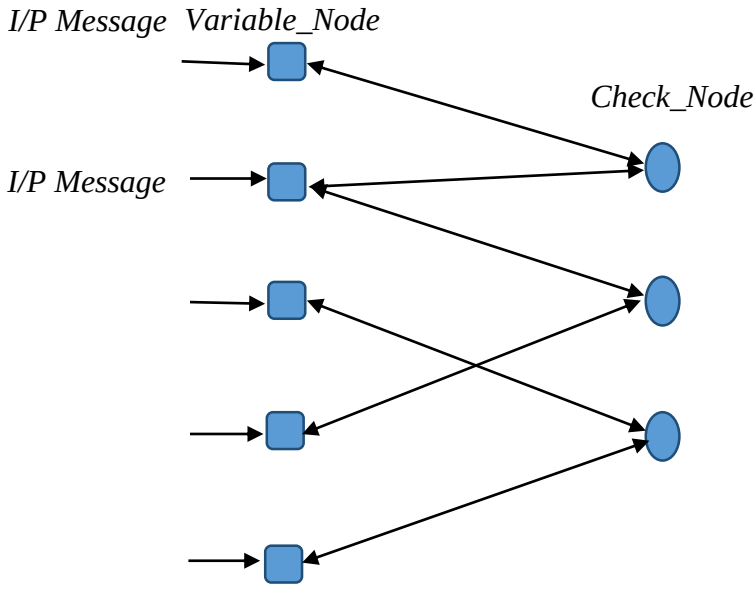


Figure 4-5 Tanner graph representation for SPA

$$\lambda_{n\to m}(u_n) \triangleq ln\left(\frac{q_{n\to m}(0)}{q_{n\to m}(1)}\right) \qquad (4.4)$$

and

$$\Lambda_{n\to m}(u_n) \triangleq ln\left(\frac{r_{n\to m}(0)}{r_{n\to m}(1)}\right) \qquad (4.5)$$

Where $\lambda_{n\to m}$ and $\Lambda_{n\to m}$ represents variable node update and check node update respectively .Now the SPA can be summarized in three steps.

Step1

Initialization:

After transmission through the channel, compute the a posteriori probability of each symbol node 'n' as $L(u_n)=L_c y_n$ Assuming the AWGN channel with noise varianceσ^2, the reliability value is $Lc = 2/\sigma^2$,. The initialization is done in every position *(m, n)* of the parity check matrix *H*, where $H_{m,n}$,=1 as

$$\lambda_{n\to m}(u_n) = L(u_n) \tag{4.6}$$

$$\Lambda_{m\to n}(u_n) = 0 \tag{4.7}$$

Step 2

Iterative Process:

Update the check-node LLR, for each m and for each $n \in N(m)$, as

$$\Lambda_{m\to n}(u_n) = 2\ tanh^{-1}\left\{\prod_{n'\in N(m)/n} tanh\left[\frac{\lambda_{n'\to m(u_{n'})}}{2}\right]\right\} \tag{4.8}$$

Note that both the *tanh* and $tanh^{-1}$ functions are monotonically increasing and have odd symmetry. Thus, the sign and the magnitude of the incoming messages (l) can be used in a simplified version, as

$$\Lambda_{m\to n}(u_n) = 2\left\{\prod_{n'\in N(m)/n} sign\left[\lambda_{n'\to m(u_{n'})}\right]\right\} tanh^{-1}\left\{\prod_{n'\in N(m)/n} tanh\left[\frac{\lambda_{n'\to m(u_{n'})}}{2}\right]\right\} \tag{4.9}$$

Step 3

Variable node update

Update the variable node LLR, for each n and for each $m \in M(n)$, as

$$\lambda_{n\to m}(u_n) = L(u_n) + \sum_{m'\in M(n)/m} \Lambda_{m'\to n}(u_n) \tag{4.10}$$

Step 4

Decision Process

Decide if $\lambda_n(u_n) \geq 0$, then $u_n = 0$ and if $\lambda_n(u_n) \leq 0$ then $u_n = 1$. Then compute the syndrome $uH^T=0$, then the codeword (u) is the final codeword, otherwise the iteration takes place till valid code word is obtained or iteration (by going to step 1 and carrying on till step 4)

Approximation or Quantized value for *tanh* & *tanh*$^{-1}$ used for implementation purpose

x	$tanhx$
(-7.0,-3.0)	-.99991
(-3.0,-1.6)	-.9801
(-1.6,-0.8)	-.8337
(-0.8,0.0)	-.3799
(0.0,0.8)	.3799
(0.8,1.6)	.8337
(1.6,3.0)	.9801
(3.0,7.0)	.99991

(tanh function)

x	$tanh^{-1}x$
(-.99998,-.9951)	-3.3516
(-.9951,-.9217)	-.1.9259
(-.9217,-.6640)	-.1.0791
(-.6640,0.0)	-.3451
(0.0,.6640)	.3451
(.6640,.9217)	1.0791
(.9217,.9951)	1.98259
(.9951,.99998)	3.3516

($\tanh^{-1}$ function)

Table 4 Quantization table for *tanh* and tanh^{-1} approximation

We have used LUT approximation in decoder as per the quantized value according to the table. For example if for *tanh* case if *tanh* is 0.5 then it will take the value .3799 from the LUT and we implemented the decoder according to this.

4.2.2.2 VHDL Implementation of LDPC Decoder

The VHDL implementation of the decoder involves use of LUT (Look up Table) for storing the value and selecting the approximate value for transmission through variable node and check node as iteration process between check node and variable node. We have implemented the Decoder using *XilinxVirtex-4 ML401* as the board.

Top Level Schematic of Decoder

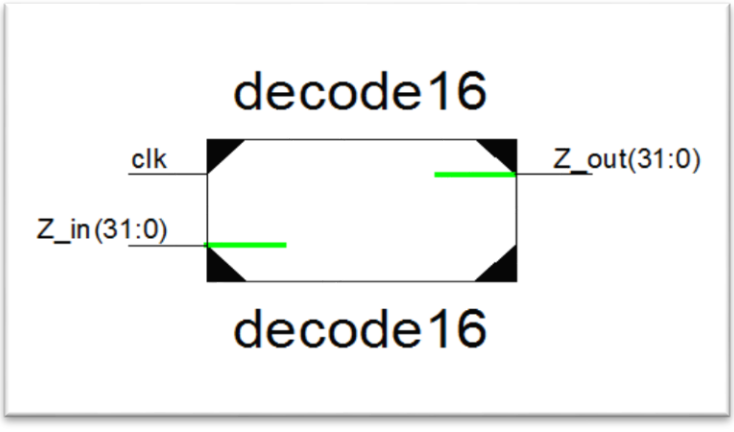


Figure 4-6 Decoder Top Level Schematic

I/O Bus in Decoder

Z_in (31:0): Message Input -32 bit

clock: Clock Signal - 1 bit

Z_out: Decoded Signal -32 bit

RTL Schematic of the Decoder

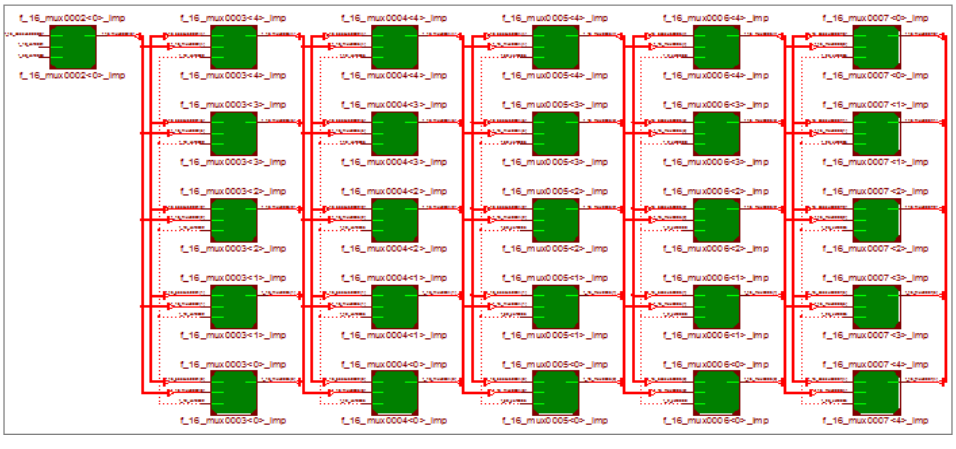


Figure 4-7 Decoder RTL Schematic

Enlarged portion mux002

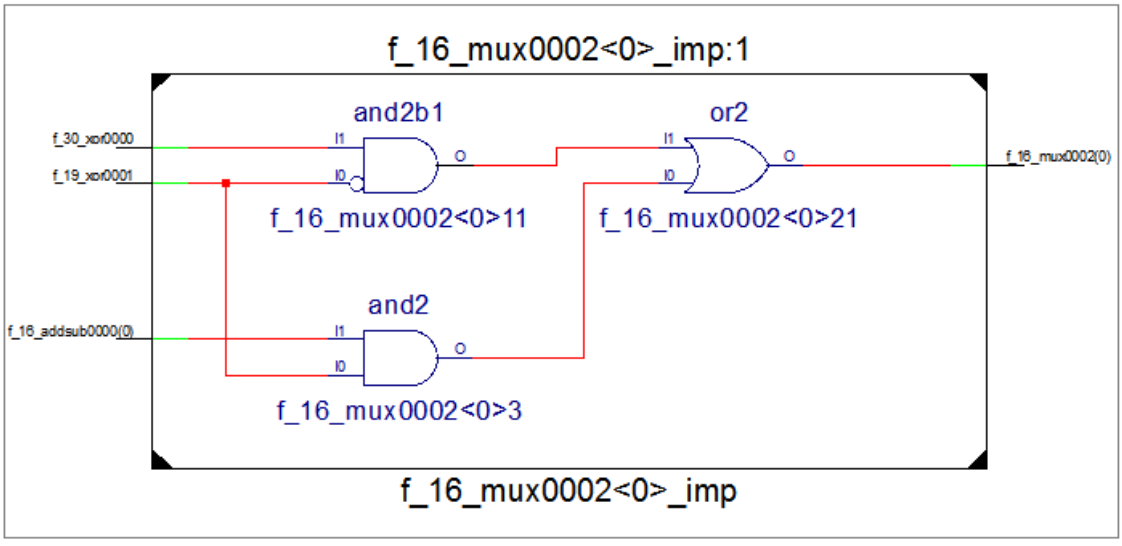


Figure 4-8 Enlarged portion of Decoder RTL Schematic

Device Utilization:

Device Utilization Summary (estimated values)			[-]
Logic Utilization	**Used**	**Available**	**Utilization**
Number of Slices	100	5472	1%
Number of Slice Flip Flops	40	10944	0%
Number of 4 input LUTs	181	10944	1%
Number of bonded IOBs	43	240	17%
Number of GCLKs	1	32	3%

Figure 4-9 Device utilization for Decoder in Virtex 4 board

4.3 Result Analysis

The message bits are given through test bench wave form input and the output as shown in the test bench wave form and the output sent to decoder and tested with some flipped bits to check the encoder-decoder system.

The performance of the system is studied first under MatLab.Then it is checked using taking some message bits and transmitting it through Test Bench Wave form .The same H matrix and G matrix used in this encoder/decoder is first tested under MatLab environment and it's BER vs SNR curve is as given below.

4.3.1 SNR vs BER plot for H Matrix used for Implementation of LDPC code

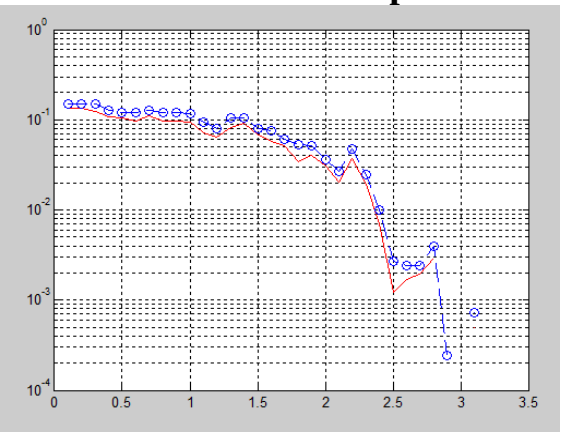


Figure 4-10 SNR Waveform for H Matrix used for Encoding/Decoding

4.3.2 Test Bench Wave Form

We checked for different inputs in the encoder and transmitting in the input and sending the encoded output to the decoder and checked to get the correct bits in case of some flipped bits if it has been flipped in the channel transmission.

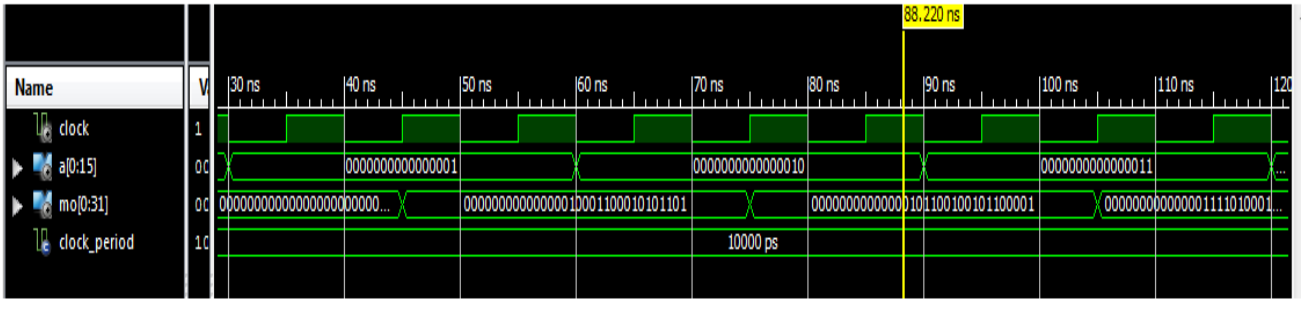


Figure 4-11 Encoder Test Bench Wave Form

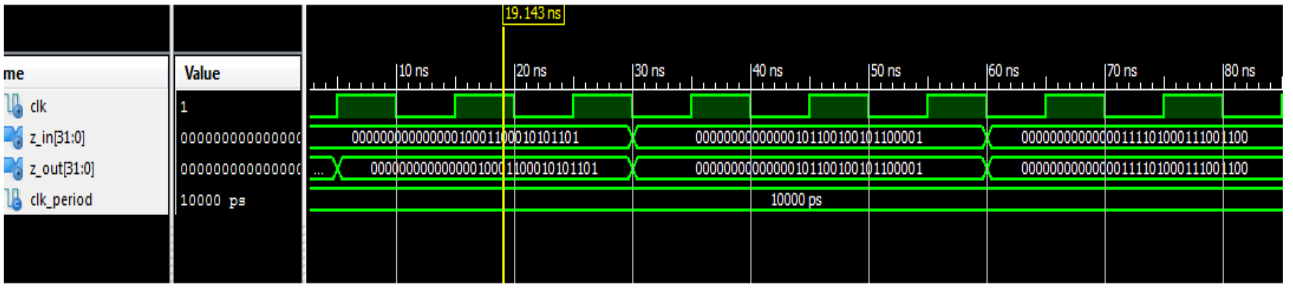


Figure 4-12 Decoder Test Bench Wave Form

Chapter 5

Conclusion and future works

5.1 Conclusion

In this project we have done the detailed analysis of LDPC code and studied different types of encoding and decoding algorithms, and found LDPC code to be good block error correction code. Its error performance was found to be near to Shannon's limit when we implemented this LDPC coded BPSK system in Matlab and then this is implemented in FPGA for testing and the decoder Modified Sum-Product Algorithm was found to be effective for decoding also. The conclusion for this project can be summarized as below.

1. LDPC Code's error correcting performance is near to Shanon's error correction limit.
2. LDPC Code's performance depends on various characteristics of the parity matrix such as matrix dimension, girth of the matrix, regularity etc.
3. Cyclic codes are easy to implement due to its parallel structure nature.
4. Sum-product Algorithm (SPA) have good error performance in log-domain as compared to probability domain.
5. Performance of MSPA (Modified SPA) is also near to SPA performance and easy to implement in FPGA.
6. Decoder behavior of various decoding algorithm was studied including Min-Sum and Message passing and their behavior was found near to Shanons.
7. The FPGA implementation of Encoder and decoder was studied on *Spartan 3E* and *Virtex 4* board respectively and found device utilization permissible.

5.2 Future Works

LDPC code is found to be a good error correcting code and it's performance near to Shanon's limit makes it a good candidate for various kind of modulation schemes such as OFDM,DVB-S2 and 802.3 an(Wi-Max) . Future goal is to study LDPC based OFDM System and its implementation.

OFDM is a candidate for future communication technique which are bandwidth hungry such as 4G and 5G and as there is a need of error correcting codes in this technique and also OFDM suffers from some problems like PAPR, ISI and we can use LDPC and then try to find its improvement and finally we will go for it's implementation.

Some times LDPC is used with other Block error correction codes for better performance such as with RS codes and performance is seem to be improved .We will also consider RS-codes along with LDPC codes for future works.

Finally we will try to improve the implementation complexity by improving the decoder architecture or by modifying decoder algorithm.

Bibliography

[1] R. Gallagar "Low Density Parity Check Code" *IRE Transaction paper Theory* pp 21-28, Jan 1962.

[2] R.M.Tanner "A recursive approach to Low complexity codes" *IEEE Trans. Information Theory*, pp. 533-547 Sept 1981.

[3] D. MacKay and R.Neal "Good codes based on very sparse matrices" in *Cryptography and Coding,5th IMA Conf* Springer 1995.

[4] D. MacKay "Good error correcting codes based on very sparse matrices" *IEEE Trans. Information Theory* ,pp. 399-431,March 1999.

[5] N.Alon and M. Luby "A linear time erasure-resilient code with nearly optimal recovery" *IEEE Trans. Inf. Theory* ,pp.1732-1736 , Nov 1996

[6]T.J Richardson and R.Urbanke "Efficient encoding of low-density parity-check codes" *IEEE Trans. Inf. Theory* vol 47 ,pp.638-656 , Feb 2001.

[7] Error detection and correction, *http://en.wikipedia.org/wiki/Error_detection_and _correction*, July 20, 2010..

[8]M.Luby,M.Mitzenmacher,M.Shokrollahi and D.Spielman "Improved low-density parity check codes using irregular graphs " *IEEE Trans. Inf. Theory* pp.585-598 , Feb 2001.

[9] J.Bhasker, ―A VHDL PRIMER, *Prentice-Hall India, Edition 3*, 1998.

[10]D.J.C. MacKay http://wol/ra.phy.cam.ac.uk/mackay

[11] C. M. Jorge and G. F. Patrie, Essentials of error-control coding, *John Wiley and Sons, Ltd, Chichester*, UK, 2006

[12] D. J. C. Mackay, S. T. Wilson and M. C. Davey, "Comparison of construction of irregular Gallager codes", *IEEE Transactions on Communications, Vol. 47, pp. 1449-1454,* Oct. 1999.

[13] Y. Kou, S. Lin and M. P. C. Fossorier, "Low-density parity-check codes based on finite geometries: a rediscovery and new results", *IEEE Transactions on Information Theory*, Vol. 47, no. 7, pp. 2711-2736, Nov. 1981.

[14] Y. Kou, S. Lin and M. P. C. Fossorier, "Low-density parity-check codes based on finite geometries: a rediscovery and new results", *IEEE Transactions on Information Theory*, Vol. 47, no. 7, pp. 2711-2736, Nov. 1981.

[15] N. Wiberg, "Codes and Decoding on General Graphs", Ph.D. dissertation, Linkoping University, Sweden, 1996.

[16] S. Myung, K. Yang and J. Kim, "Quasi-cyclic LDPC codes for fast encoding", *IEEE Transactions on Information Theory*, Vol. 51, no.8, pp. 2894- 2900, Aug. 2004.

[17] W. E. Ryan, "An introduction to LDPC codes", Department of Electrical and Computer engineering, University of Arizona, Aug. 2003.

[18] M. P. C. Fossorier, "Quasi-cyclic low-density parity-check codes from circulant permutation matrices", IEEE Transactions on Information Theory, Vol. 50, no. 8, pp. 1788-1794, Aug. 2004

[19] LDPC Code using MATLAB, *http://sites.google.com/site/bsnugroho/ldpc*

[20] S.Papaharalabos ,P.Sweeney,B.G.Evans, *"Modified sum product algorithm for decoding LDPC"* ,IET Comm ,2007

[21] William Y .Ryan "An Introduction to LDPC codes" University of Arizona.

[22] Das Shubhashree "*FPGA implementation of RS Codes*" NIT Rourkela 2011

[23]Hayes David "FPGA implementation of a Flexible LDPC decoder" University of Newcastle