DE-40 (DC&SE)

NUST COLLEGE OF

ELECTRICAL AND MECHANICAL ENGINEERING

LDPC codes implementation on FPGA for 5G communication

A PROJECT REPORT

DE-40 (DC&SE)

Submitted by

NS SHAHID AKHTAR

NS USAMA BIN KHALID

NS MUHAMMAD FARHAN

BACHELORS

IN

COMPUTER ENGINEERING

YEAR

2022

PROJECT SUPERVISOR

DR. SHOAIB AHMED KHAN

DR. SAJID GUL KHAWAJA

COLLEGE OF

ELECTRICAL AND MECHANICAL ENGINEERING

PESHAWAR ROAD, RAWALPINDI

YEAR 2022

LDPC codes implementation on FPGA for 5G communication

A PROJECT REPORT

DEGREE 40

Submitted by

NS SHAHID AKHTAR

NS USAMA BIN KHALID

NS MUHAMMAD FARHAN

BACHELORS

IN

COMPUTER ENGINEERING

Year

2022

PROJECT SUPERVISOR

DR. SHOAIB AHMED KHAN

DR. SAJID GUL KHAWAJA

COLLEGE OF

ELECTRICAL AND MECHANICAL ENGINEERING

PESHAWAR ROAD, RAWALPINDI

DECLARATION

As a result, we attest that no portion of the work referred to in this Project Thesis was submitted in support of any other degree or certification at any other university. If plagiarism is detected, we are fully accountable for any disciplinary action taken against us, regardless of the severity of the verified offence.

COPYRIGHT STATEMENT

ACKNOWLEDGEMENTS

First and foremost, Alhamdulillah, our FYP has finally been completed, and we are grateful to Allah for providing us with the strength and morale to continue pushing forward and for assisting us at every step of the road.

Second, we would like to express our heartfelt gratitude to our supervisors, Dr. Shoaib Ahmed Khan and Dr. Sajid Gul Khawaja, who assisted us greatly on every single issue, and whose assistance and direction became a source of strong determination for us. Thank you, sirs; you played an important role in our lives that we will never forget.

Finally, we would want to express our gratitude to our parents and friends, without whom we would not have been able to accomplish our final year project. They were invaluable throughout our journey, and we will be eternally grateful to them. Their unwavering support pushed us to accomplish more than we could have imagined, and they instilled new hope in us when we had lost all hope in ourselves.

ABSTRACT

LDPC codes are a hot topic in information coding theory right now. For ordinary to lengthy code lengths, Low-Density Parity-Check (LDPC) codes are among the most powerful channel codes. They were invented by Gallager in 1962, but they had to be re-invented by Mackay and Neal in 1997 before their powers were discovered: The channel capacity can be achieved with irregular LDPC coding. LDPC codes, unlike many other types of codes, already have very fast (probabilistic) encoding and decoding methods. The question is how to construct the codes so that these algorithms can recover the original codeword even in the presence of a lot of noise. The design problem can now be solved using new analytic and combinatorial tools. As a result, LDPC codes are not only appealing from a theoretical standpoint, but also ideal for practical applications. This study will provide a brief introduction of the origins of LDPC codes, their coding and decoding procedures, and the methodologies used to analyse and create them.

Contents

Table of Figures

Table of Tables

Chapter 1: Introduction

1.1 Introduction

Conversation structures rely heavily on error-correcting codes (ECC). It enables low-energy and trustworthy transmission over noisy channels for conversation architectures. The best-known ECC is Low-Density Parity Codes (LDPC). Although LDPC and other error-correcting codes cannot ensure faultless transmission, they can reduce the likelihood of lost statistics as much as feasible. LDPC emerged as the primary code that allows record transmission costs to approach the theoretical limit of the Shannon Limit. As a result, it is frequently referred to as a next-generation error-correction code. For trustworthy communication with low energy usage over noisy channels, error-correcting codes must be used. Error-correcting codes offer redundancy to the supplied data stream, allowing the receiver to detect and perhaps remedy mistakes that occur during channel transmission. There are various types of code exits with unique applications. To fit into the gap of realistic hardware implementation, the encoding/deciphering set of rules for each code must be altered. However, we can deal with the LDPC. These codes work exceptionally well in noisy channels.

1.2 Error Correction

1.2.1 Basic Concept of error correcting

In Computer language bit is either "0" or 1". An organization of bits shapes a byte that consists of eight bits. Error-correcting code includes the transmission of extra redundant bits with the information so that it will permit the correction and detection of a few errors on the receiver end.

1.2.2 History of error correction

In 1940, Shannon reason that the most effective manner to get the maximum efficient storage potential in a storage tool or the short transmission via a communications channel is thru using effective error-correcting structures called Shannon's "Mathematical Theory of Communication". Richard Hamming determined and applied the single-bit error code. Shannon gave the formulation approximately the transmission of records over a noise channel with a low error charge because the charge of the records is much less than channel potential.

$$C = B\ log_2\ (1\ + \frac{S}{N})$$

In early 1960, Irving Reed and Gustave Solomon determined a way to assemble error-correcting codes that might an accurate arbitrary wide variety of bits In early 1968 Elwyn Berlekamp and James Massey determined a set of rules for constructing a decoder for more than one error correcting codes and called Berlekamp- Massey set of rules. Euclid prolonged this set of rules to best not unusual place divisor of polynomials Now a day the algorithms of Berlekamp-Massey and Euclid used to resolve the decoding equations.

1.2.3 What is error correction?

The technique of finding and rectifying bit-wise faults using hardware or software is known as error correction. Error detection refers to the capacity to identify errors produced by noise or other constraints inside the transmission line from the transmitter to the receiver. Error correction includes an additional feature that enables for the localization and correction of errors. Given the goal of error rectification, the concept of error detection may appear inadequate.

However, in which performance is important, it's miles viable to hit upon accurate errors with much less redundant information. There are some techniques used for error correction as given below:

1. Redundancy is used to check validity.
2. The information from redundancy is used to correct bad blocks.
3. The errors can't be corrected then concealed.
4. When uncorrectable errors are found the system mute the output. Then by turning all the bits to "0" which cancels the error. Replace the faulty data with the original data when error is detected.

The parity check bits are used to detect error and also for creating codes for correction

1.3 History of LDPC

Gallager invented LDPC in the 1960s, and Mackay rediscovered it in 1999. It also provides Shannon-limit nearing overall performance, smooth decoding complexity and implementation, but higher encoding complexity. These are sometimes referred to as linear block codes. When LDPC was invented in 1963, it was considered impractical to utilise. Nothing even one-third as strong was generated in the subsequent 30 years of information theory, and LDPC remains, in principle, the simplest evolved to date (2006).

The phenomenal rise of the information age has led in a corresponding surge in business interest in the creation of very efficient data transmission codes, as such codes influence everything from signal quality to battery life. Although LDPC code implementation has lagged behind that of other codes, particularly the faster code, the absence of encumbering software programme patents has made LDPC appealing to some, and LDPC codes are poised to become a standard within the growing market for extremely efficient data transmission methods.

1.4 Types of LDPC

LDPC codes are classified into two types: regular and irregular. The regular codes with the same row and column weights in the parity-check matrix H. The codes have uneven weights in the column and row. LDPC codes can also be binary or nonbinary in nature. Nonbinary codes contain elements (GF(q)), where q=2p for any integer p, whereas binary codes have two elements (0 and 1).

1.5 Low-Density Parity Codes (LDPC) Description

A low-density parity-check code (LDPC code) is an error-correcting code that is used to transfer data across a noisy transmission channel. A linear error-correcting code (LDPC) has the following properties:

- A Linear Block Code generated by distributed Bipartite Graphs.
- The parity check matrix is H (n-k, n), and it is SPARSE.
- H is made up of n-j rows, j columns, and k 1' columns.
- Weight distribution across rows and columns.
- H is evenly dispersed, with only a few '1' in rows and columns.
- There are n Variable Nodes (on the left) that correspond to the columns of H.
- Check Nodes (on the right) correspond to rows, and there are n-k of them.
- Valid codewords are those vectors (c1, c2,... cn) for which the sum of the nearby locations among the message nodes is zero for all check nodes.
- As a result, LDPC Code - H.C' = 0.

1.6 Block Diagram

Working of a communication system is shown in figure 1. A message M to be sent is Encoder to create a codeword C. Codeword C is then modulated and transmitter over a channel. During transmission the signal can catch noise from outside and become corrupted. At the receiver end this signal is received and passed through de-modulator. After that decoder tries to calculate the message that was sent, removing errors from the message.

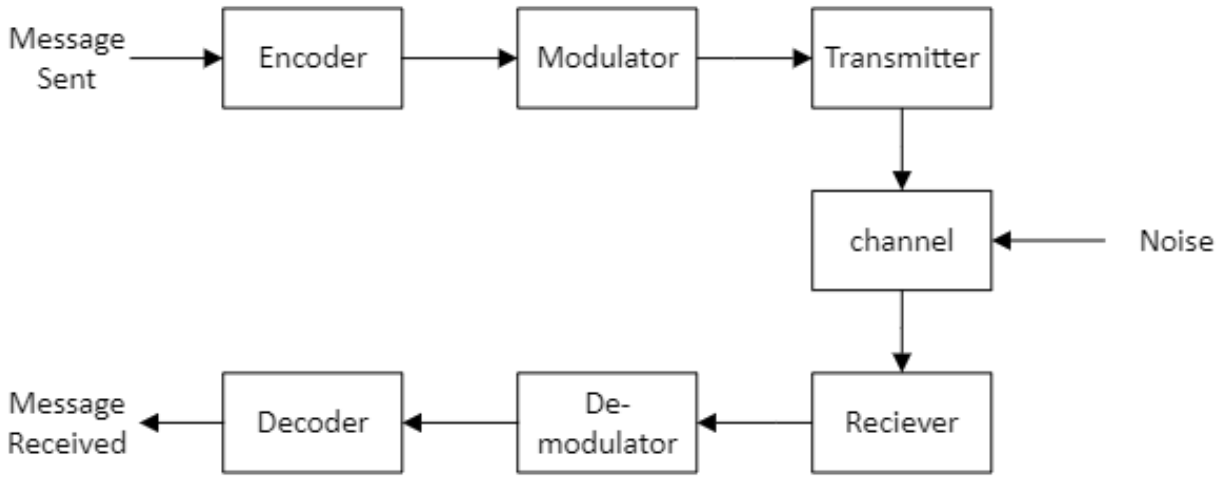

Figure 1 Block Diagram

1.7 Advantages of LDPC

Some advantages of LDPC are as follows

- LDPC codes may attain Shannon limit error performance comparable to Turbo codes.
- It is possible to deal with low SNRs and variable block size.
- Allow for varying coding rates.
- improved block error performance
- An iterative decoding approach rather than a trellis-based one.
- Some LDPC Codes outperform Turbo Codes in terms of error ceilings at significantly lower Bit Error Rate (BER) values.
- Decoding is quite quick in the Log Domain.
- Lower decoding complexity

1.8 LDPC codes Disadvantages

Some disadvantages of LDPC are as follows

- High decoding difficulty
- Extremely long code word lengths for accurate decoding efficiency
- Iterative convergence is slow, under general conditions, it takes 1000 iterations to converge.
- Because encoding, transmission, and decoding all take time, there is a significant initial lag.
- The (4086,4608) LPDC codeword has a delay of about 2 hours.
- It takes a long time to find a decent solution.

1.9 LDPC codes application

Some applications of LDPC are as follows

- Satellite Transmission
- Recording in Magnetic disk
- Satellite Global Exploration
- High Resolution capable
- Low SNR
- For intergalactic probes
- Days for data transfer
- Latency enormously small
- LDPC can make a sure amazing resolution and correct information telemetry
- Data transmission for non-real-time applications

1.10 Motivation

Communication is the most important part of the modern digital world and many steps are being taken to improve it because the digital age required multitude of data to be transmitted over long distances with high accuracy. LDPC codes are one of the most efficient error correction codes discovered until now and are going to be used in the upcoming communication technology i.e 5^{th} generation as NR-LDPC codes. Complexity of LDPC codes did not allow them to be used effectively at larger scales but now through advancement in technology and improvement in the

algorithms to perform LDPC decoding they are a feasible method to be used in any level of communication. The error correction capability at very low SNRs is the most important aspect of LDPC codes. Implementing of LDPC codes on FPGA would allow to take advantage of Hardware acceleration thus giving a huge boost to throughput of the system.

1.11 Scope

The aim of this project is to create and implement a digital system design for NR-LDPC Encoder and Decoder. The design will be coded in Verilog and then implanted onto an FPGA. Taking the advantage of parallel processing in FPGA, not only the design will be optimized for better performance than a general-purpose CPU, but multiple instances of Encoder or Decoder can be implemented on a single FPGA causing a factorial improvement in throughput although latency remains the same as for a single instance.

1.12 Structure

Following is the structure of the report ahead:

- Chapter 2 deals with the explanation of LDPC, design of LDPC codes & Parity check matrix, and encoding and decoding.
- Chapter 3, it deals with the encoding and decoding algorithm.
- Chapter 4, it deals with the architecture design and Verilog implementation of LDPC encoder and decoder.
- Chapter 5, it includes with results, simulations and tool used.
- Chapter 6, it includes conclusion and future aspect.

1.13 Summary

The most well-known are Low Density Parity Codes for ECC. Although LDPC, and other error-correcting codes cannot ensure faultless transmission, the likelihood of lost data can be reduced to as low as desired. There are various sorts of code exits with various applications. These codes work admirably in noisy channels.

LDPC codes are one of the most efficient error correction codes discovered until now and are going to be used in the upcoming communication technology i. E 5th generation as NR-LDPC codes. Complexity of LDPC codes did not allow them to be used effectively at larger scales but now through advancement in technology and improvement in the algorithms to perform LDPC

decoding they are a feasible method to be used in any level of communication. The error correction capability at very low SNRs is the most important aspect of LDPC codes.

Implementing of LDPC codes on FPGA would allow to take advantage of Hardware acceleration thus giving a huge boost to throughput of the system.

Chapter 2: Low Density Parity Check Codes

2.1 Linear Block Codes

Because Low Density Parity Check Codes (LDPC) are a subset of linear block codes, we may present an overview of those codes in this chapter to lay the framework for further research into LDPC encoding and decoding.

2.2 Low Density Parity Codes

LDPCs are a sort of linear block coding that correlates to the parity check matrix H. The density of ones on the parity check matrix H(N-K) x N may be quite low since it contains just the simplest zeros and ones.

2.3 Tanner Graph

Tanner Graph has been used successfully to represent LDPC codes. A Tanner graph is a network in which nodes are divided into two distinct classes and edges connect nodes that do not belong to the same class. Tanner graphs are classified into two types:

a) Bit Nodes
b) Check Nodes

It might produce a rudimentary parity check matrix H. Figure 2 shows a network with each node (i.e., Bit node) linked to another node (i.e., Check node).

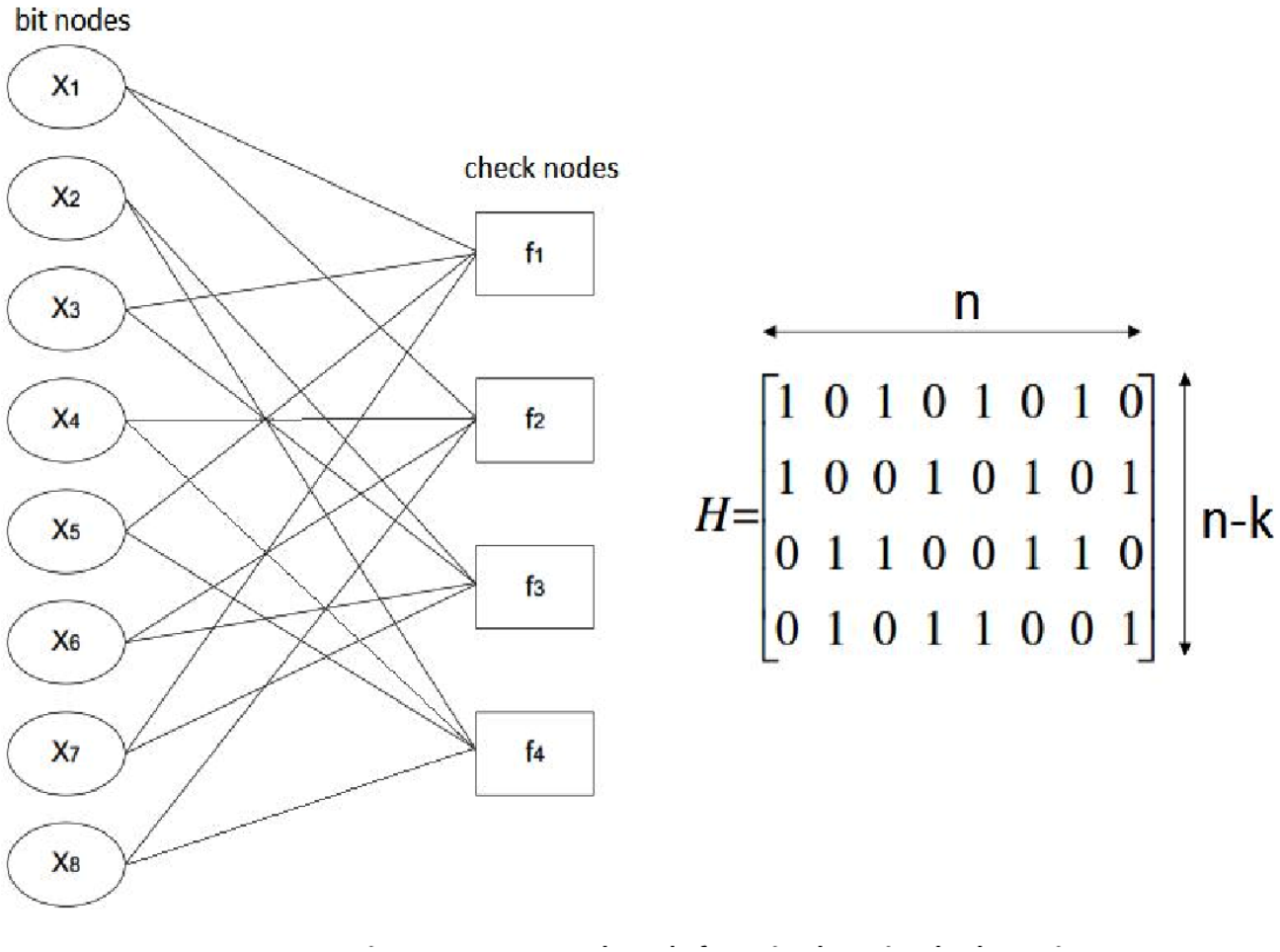

Figure 2 Tanner Graph for parity check matrix

2.4 Designing LDPC Codes

The first steps in designing an LDPC code are as follows:

a) Code with long block lengths could perform exceptionally well. The problem is that large block lengths are impractical in practise.

b) The second factor to evaluate is whether the code is regular or irregular. For irregular codes, each check node has a distinct degree, whereas each Bit node has the same degree. Normal codes have all Bit nodes with the same degree and all Check nodes with the same degree.

c) The third determinant is the number of Bit and Check nodes. What is the number of ones in each row and column of the parity matrix? For irregular codes, the number of ones and the number of unique degrees for Check nodes and Bit nodes must be determined. If the degree is bigger, more calculations are required to build the outgoing message. The degrees of all Bit nodes in regular codes may be the same, and check nodes may be constant.

d) The fourth consideration is the rate of coding and the amount of redundancy necessary inside the code.

e) The maximum number of decoding rounds is the fifth aspect.

2.5 Generate the Parity Check Matrix

The parity check matrix is an important part of LDPC's overall performance (encoding and decoding). As explained in Gallager's research piece, the matrix must be exceedingly sparse. It also determines the encoder and decoder complexity. The matrix must have random elements in order to be suitable for decoders. A well-formed matrix yields a very efficient hardware representation. It also takes significantly less RAM to keep the matrix running. The following are the ways for producing a sparse matrix H:

- Create H by generating a weight WC column at random.
- Use a uniform weight Wr row and a weight WC column to make H.
- Create the parity check matrix with a polynomial (H).
- Create the parity matrix (H) in the way specified.
- Generate H with WC column weights and Wr row weights that are uniform, with no more than one column overlap.
- Generate H in the same way as (e) and prevent any additional short cycles.
- Make a H matrix with all 0s of size (N-K) X N and randomly flip a few entries.

The generator matrix G may be easily identified after constructing the parity matrix H by setting GHT=0, then conducting Gaussian removal on the output matrix G and making it inside the shape G=[I|P].

2.6 Encoding

When encoding LDPC codes, each information bit is often allocated to a Bit node in the Tanner graph. By satisfying the parity check criterion, we find the remaining Bit node values.

Assume we need to encode a message m of K bits using LDPC. C is calculated using the following equation.

$$C = m\mathrm{G}$$

In which C is the N bit codeword and $G_{K \times N}$ is the generator matrix, and $GH^{T}=0$ was achieved.

Consider the following scenario: we need to send message m= [1011] across the channel. It is encrypted by us.

• Parity check matrix H

$$\underline{c} = [c_1, c_2, c_3, c_4, c_5, c_6, c_7]$$

$$H = \begin{bmatrix} 1 & 1 & 1 & 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 & 1 \end{bmatrix} \qquad H\underline{c}^{\mathrm{T}} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$$

•Generator matrix G

$$G = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 1 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 & 1 \end{bmatrix}$$

$$\underline{u} = [u_1, u_2, u_3 . u_4]$$

$$\underline{c} = [c_1, c_2, c_3, c_4, c_5, c_6, c_7]$$

$$\underline{u} \cdot G = \underline{c}$$

The codeword will change into:

$$C = m\mathrm{G} = [1\,0\,1\,1\,0\,1\,0]$$

Encoding appears to be a big computational operation; all of the parity check equations must be met. Encoding can be done quite quickly in practice, but decoding is a little more difficult.

2.7 Decoding

Gallager devised a decoding technique that is highly efficient in 1960. Numerous studies have been conducted on this topic, and academics have separately developed algorithms with novel applications.

For LDPC coding, we need a Tanner graph to describe iterative decoding. There are a few decoding algorithms listed below:

- Bit Flipping Algorithm
- Sum-product Algorithm -probability Domain
- Sum-product Algorithm -Log Domain
- Min-sum Algorithm
- Modified Min-sum Algorithm

2.7.1 Bit Flipping Algorithm:

The basic idea behind this strategy is to "flip" as few bits as possible till the parity checks pass. Consider that each Bit node starts with either a 0 or a 1. At each cycle, the Bit node decides whether to flip its value or keep it untouched. It is expected that the Bit node value in error has the most unsatisfied check equations. This method is simpler, but its density is somewhat smaller, suggesting that just a few bits are involved in each check equation.

2.7.2 Sum-product Algorithm -probability Domain:

Assume that you have a (N-K) X N parity check matrix with N K checked nodes and N bit nodes. Check nodes represent the check equation, whereas bit nodes represent the coding bits. Decoding is done iteratively. The bit node sends a message to the connected check node every iteration, and the check node sends a message to the bit nodes. It decides if the codeword is legitimate or not. Iterates until it finds the correct codeword or reaches the maximum number of iterations.

2.7.3 Sum-product Algorithm -Log Domain:

This technique performs operations on the parity check matrix's rows and columns, i.e., H.

It includes the following steps:

Step 1:

Initialize Lq_{ji} by :

$$Lq_{ji} = Lc_i = {}^{2y_i}/_{\sigma^2}$$

Step 2: Check Lr_{ji} by:

$$Lr_{ji} = \left(\pi_{i' \in R_j \atop i} \alpha_{ji'}\right) . \phi(\Sigma_{i' \in R_{j/i}} \phi(\beta_{ji'}))$$

where,

$$\alpha_{ji} = sign(Lq_{ji})$$

$$\beta_{ji} = |Lq_{ji}|$$

$$\phi(x) = -\log\left(tanh\left(\frac{x}{2}\right)\right)$$

$$= \log\left(\frac{e^x + 1}{e^x - 1}\right)$$

Step 3:

$$Lq_{ji} = Lc_{j'} + \Sigma_{j' \in C_i \setminus j} Lr_{j'i}$$

Step 4:

$$Lq_i = Lc_i + \Sigma_{j \in C_i} Lr_{ji}$$

Step 5: For every row index i:

$$\hat{C}_i = \begin{cases} 1 & if\ Lq_i < 0 \\ 0 & else \end{cases}$$

If $cH^T = 0$ or the maximum number of iterations is reached, halt; otherwise, iterate from step 1.

2.7.4 Min-sum Algorithm:

Consider the Sum-Product algorithm's update equation for Lr_{ji}:

$$Lr_{ji} = \left(\pi_{i' \in R_{\frac{j}{i}}} \alpha_{ji'}\right).\phi(\Sigma_{i' \in R_{j/i}} \phi(\beta_{ji'}))$$

$\phi(x)$ is a function that decreases as x increases. It is apparent that the word corresponding to the smallest ji inside the preceding summation dominates, such that

$$\phi\left(\Sigma_{i' \in R_{\frac{j}{i}}} \phi(\beta_{ji'})\right) = \phi\left(\phi(\min_{i'} \beta_{ji'})\right) = \min_{i'} \beta_{ji'}$$

Take note that the second equality arises from Á(Á(x)) = x. Thus, the Min-Sum approach is analogous to the Sum-Product method, except that the following equation replaces step 1:

$$Lr_{ji} = \left(\pi_{i' \in R_{\frac{j}{i}}} \alpha_{ji'}\right).\min_{i' \in R_{\frac{j}{i}}} \beta_{ji'}$$

Because of the approximation on this equation, the overall performance of the Min-Sum approach is inferior to the Sum-Product method.

2.7.5 Modified Min-sum Algorithm:

According to the theory, changing the size of the gentle data for the duration of the decoding process using a min-sum method yields better performance. When compared to the sum-product method, changing the size of the data reduces the convergence of iterative decoding and reduces error. To determine the best scaling element, density evolution strategies can be used. It also confirmed that an LDPC(3,6)code scaling element of 8/10 is optimal. It is far enough 35 in this method to alternate step 2 in the Min-Sum method with

$$[Step2' :]Lq_{ji} = \left(Lc_i + \Sigma_{j' \in C_{\overline{i}}^{j}} Lr_{j'i}\right) * \gamma$$

in which γ is the scaling element.

2.8 Literature Review

- Survey of Turbo, LDPC and Polar Decoder ASIC Implementations [1]
    - This paper explains the factors that caused this debate, with a focus on the Application Specific Integrated Circuit implementation of the decoders for these three codes. We show that variables other than computational complexity impact the overall implementation difficulties of turbo, LDPC, and polar decoders. We evaluate the throughput, error correction capabilities, flexibility, space efficiency, and energy efficiency of 110 ASIC implementations and use the results to characterise the benefits and drawbacks of these three codes, as well as to avoid traps and give design suggestions.
- High Performance LDPC Decoder Design using FPGA [2]
    - LDPC codes are a critical component of 5G communication systems. LDPC codes are among the most efficient error correcting codes for FPGA implementation. The primary goal is to develop a low complexity LDPC decoder architecture on the FPGA. VNU and CNU are the two fundamental components of LDPC.

      LDPC is an important component of deep space communications, and its potential application in this field is being investigated. It is critical in space data systems to have an LDPC decoder with a minimal complexity and high-performance architecture.

- Design of LDPC Decoder Using FPGA [3]
    - This study proposes a dual-mode low-density parity-check decoding structure with excellent error-correction functionality and a high parallelism layout for fifth-generation new-radio applications. To meet the high throughput requirements of 5G NR systems, we used an excessive parallelism layout with a tiered decoding schedule. We also developed a compensation technique to improve decoding overall performance loss by changing the probabilistic second minimum of a grouping search.

2.9 Summary

Because LDPCs are a subset of linear block codes, we reviewed them in this chapter to provide the framework for studying LDPC encoding and decoding.

Low-density parity codes are a sort of linear block coding that correlates to the parity check matrix H.

The parity check matrix is crucial to the performance of LDPC.

Chapter 3: LDPC Algorithm for Encoder and Decoder

Here we are going to explain the algorithm we used for implementing Encoder and Decoder in MATLAB/C.

3.1 Encoder:

Encoder implements the process of calculating parity bits for a given message that can be used for decoding on the receiving end of the system. The most important component of LDPC algorithm is a parity check or parity generator matrix. The matrix is used both in encoding and decoding. The parity check matrices are predefined for NR-LDPC codes. There are two base graphs, that are base graph 1 which has dimensions of 46 x 68 and base graph 2 having dimensions 42 x 52 and both have 51 different expansion factors. To explain the algorithm, we will only use base graph 1 and take expansion factor to be equal to 64.

As the dimensions of base graph 1 are 46 x 68, they can be used along with expansion factor (64 in this case) to find out length of message initially and length of the codeword after all the parities have been calculated.

$$\text{Message Size} = (68-46) \times 64 = 1408 \text{ bits}$$

$$\text{Codeword Size} = 68 \times 64 = 4352 \text{ bits}$$

$$\text{Parity bits} = \text{Codeword size} - \text{Message size} = 2944$$

Given below is figure 3 depicting the shape of base graph 1 matrix:

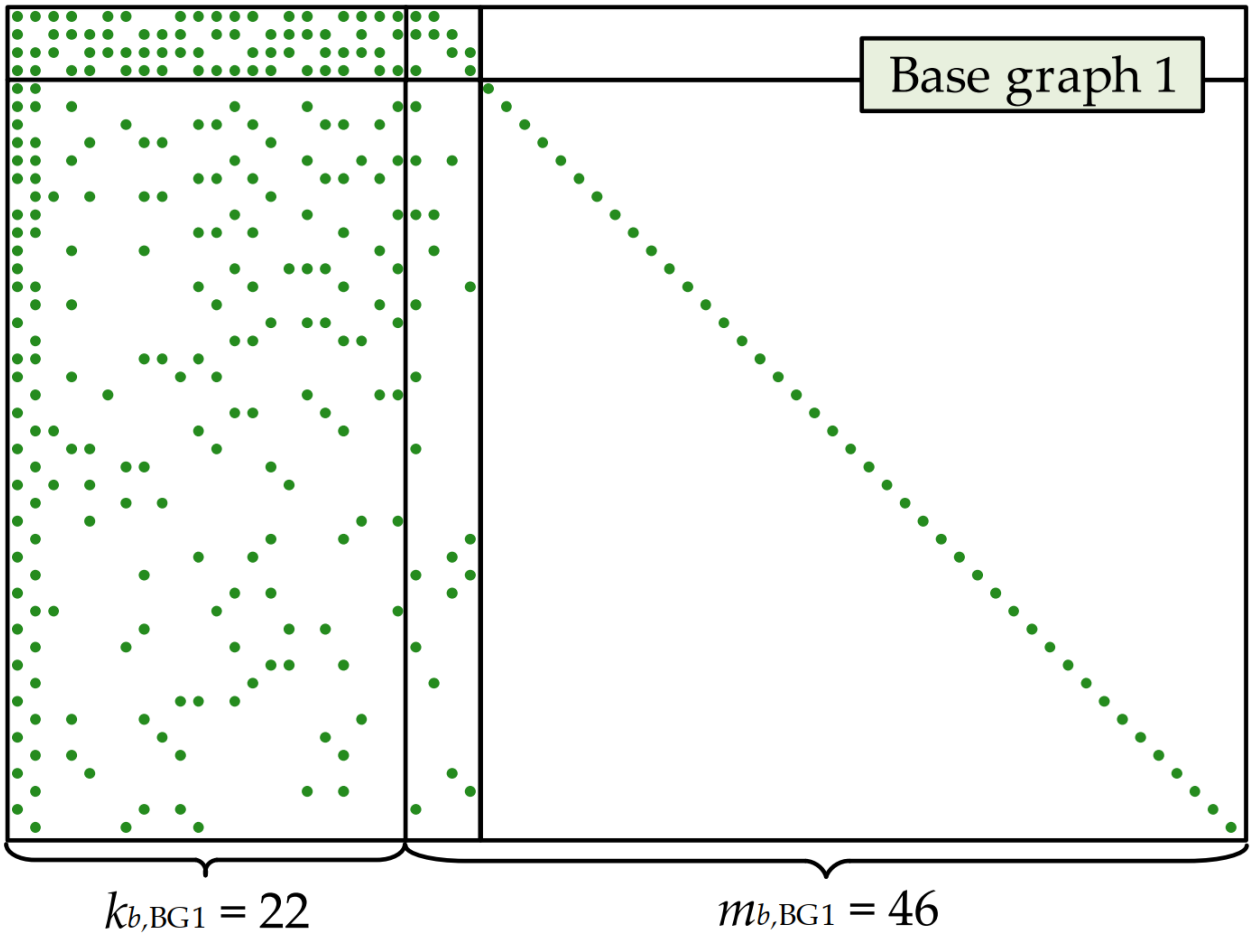

Figure 3 Base Graph

Depicted here are 46 rows and 68 columns with white spaces and dots positioned sparsely. Each dot represents a non-negative integer, and each space indicates a -1. Wherever the matrix contains a -1 no processing is required to be done there and at every dot there is a number which indicates the amount of circular shift required for respective message bits to calculate parity. Parities are grouped in a chunks of expansion factor i-e 64 bits. Almost each row of the matrix can used to calculate 1 group of parities for the codeword except for the first four groups of parities. This is indicated by a line in matrix after first four rows. The shape of the matrix is such that the first parity group (p1) can be calculated using all first four rows, then the second parity group (p2) is calculated using the first row only, the third parity group (p3) is calculated using second row and the fourth parity group (p4) is calculated using third row. After that all parities can be calculated from the respective rows. The final form of codeword is as follows:

m1	m2	m3	. . . . . .	m22	p1	p2	p3	. . . . . .	p46

To calculate parities p1-p46 we need m1-m22 message bits where each of them is a group of 64 bits and the entries in base matrix $B_{i,j}$ denoting j^{th} entry in i^{th} row. The formulas for calculating all the parities are given below:

$$\begin{aligned} p1 = {} & f(m1,B_{1,1}) \oplus f(m2,B_{1,2}) \oplus f(m2,B_{1,2}) \oplus f(m3,B_{1,3}) \ldots\ldots \oplus f(m22,B_{1,22}) \oplus \\ & f(m1,B_{2,1}) \oplus f(m2,B_{2,2}) \oplus f(m2,B_{2,2}) \oplus f(m3,B_{2,3}) \ldots\ldots \oplus f(m22,B_{2,22}) \oplus \\ & f(m1,B_{3,1}) \oplus f(m2,B_{3,2}) \oplus f(m2,B_{3,2}) \oplus f(m3,B_{3,3}) \ldots\ldots \oplus f(m22,B_{3,22}) \oplus \\ & f(m1,B_{4,1}) \oplus f(m2,B_{4,2}) \oplus f(m2,B_{4,2}) \oplus f(m3,B_{4,3}) \ldots\ldots \oplus f(m22,B_{4,22}) \end{aligned}$$

The function f(a,b) returns b times circular shifted vector a, or a zero vector if b is -1.

$$\begin{aligned} p2 = {} & f(m1,B_{1,1}) \oplus f(m2,B_{1,2}) \oplus f(m2,B_{1,2}) \oplus f(m3,B_{1,3}) \ldots\ldots \oplus f(p1,B_{1,23}) \\ p3 = {} & f(m1,B_{1,1}) \oplus f(m2,B_{1,2}) \oplus f(m2,B_{1,2}) \oplus f(m3,B_{1,3}) \ldots\ldots \oplus f(p1,B_{1,23}) \oplus \\ & f(p2,B_{1,24}) \\ p4 = {} & f(m1,B_{1,1}) \oplus f(m2,B_{1,2}) \oplus f(m2,B_{1,2}) \oplus f(m3,B_{1,3}) \ldots\ldots \oplus f(p1,B_{1,23}) \oplus \\ & f(p2,B_{1,24}) \oplus f(p3,B_{1,25}) \end{aligned}$$

For all other parities from p5-p46, the formula is as following:

$$\begin{aligned} pn = {} & f(m1,B_{n,1}) \oplus f(m2,B_{n,2}) \oplus f(m2,B_{n,2}) \oplus f(m3,B_{n,3}) \ldots\ldots \oplus f(p1,B_{n,23}) \oplus \\ & f(p2,B_{n,24}) \oplus f(p3,B_{n,25}) \oplus f(p4,B_{n,26}) \end{aligned}$$

3.2 Decoder

Decoder receives noisy data and tries to calculate the correct data using available message data and parity bits. The approach we have used for decoding is the stochastic min-sum approach. In stochastic min-sum, instead of performing all the calculations over whole of the matrix at once, we proceed with calculations with one row at a time and update the LLR (Log Likelihood

Ratio) or belief associated with the respective bit. LLR is an integer between 127 and -128 and its value tells probability of the bit to be zero or one. The stochastic approach usually converges much faster than non-stochastic approach, as the results are updated as soon as they are calculated. The computational complexity of min-sum is much lesser than that of the other algorithms available because it does not require calculation involving natural log and tanh etc.

Similar to the encoder, the decoder only has to perform calculations on matrix positions where there is a non -1 value. Also, each calculation requires a separate memory to save previous value for that location so that it can be used in subsequent iterations. As there are total of 316 valid positions in base matrix so there will be 316 memories associated with them, called nodes.

The stochastic min-sum approach takes a very complex LLR calculation process involving natural log, exponential, tanh and division to a very simple calculation process that just involves finding minimum absolute values, addition and subtraction. Convergence of LLR to give a valid codeword can take a very long time that is why we will limit the number of iterations to 8 as improvement in results after 8 iterations is negligible as compared to the computational and latency overhead. Results of simulation shown in figure 8.

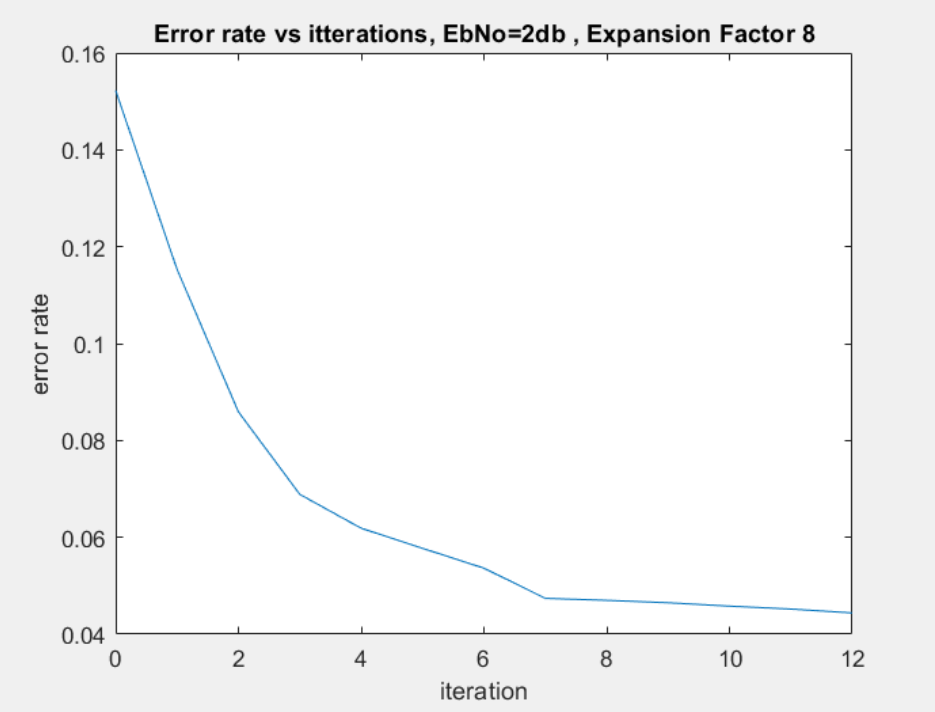

Figure 4 Error rate vs Iterations Graph

Over 8 iterations there are four steps that must be carried out for each row to update LLR. The steps are as following:

1. Update LLR by subtracting the previously stored value in node.

2. Find the 64 minimum absolute values for each respective column.
3. Update node value by respective minimum value.
4. Update LLR by adding node value into it.

The procedure mentioned above is quite simple as compared to using complex mathematical functions on which the original LDPC decoder was based.

3.3 Summary

In this chapter, we discussed the algorithm of LDPC encoder and decoder

Encoder implements the process of calculating parity bits for a given message that can be used for decoding on the receiving end of the system. The most important component of LDPC algorithm is a parity check or parity generator matrix. The matrix is used both in encoding and decoding. The parity check matrices are predefined for NR-LDPC codes.

Decoder receives noisy data and tries to calculate the correct data using available message data and parity bits. In stochastic min-sum, instead of performing all the calculations over whole of the matrix at once, we proceed with calculations with one row at a time and update the LLR or belief associated with the respective bit. Similar to the encoder, the decoder only has to perform calculations on matrix positions where there is a non -1 value. As there are total of 316 valid positions in base matrix so there will be 316 memories associated with them, called nodes.

Chapter 4: Architecture Design and Verilog Implementation

This Chapter explains the architecture, working and pre-processing required for Encoder and Decoder implementation.

4.1 Encoder

The complexity of encoder is lower as compared to that of decoder due to following reasons:

1. No complex operations. Only circular shift and xor is required.
2. Decoder is implemented with iterations and multiple steps that take a greater number of clock cycles whereas encoder takes a very limited number of cycles.
3. For decoder every bit requires an 8-bit LLR value to represent it as compared to a 1-bit in encoder.
4. Number of memories required in decoder is greater than that of encoder.

4.1.1 Pre-processing:

In order to reduce number of cycles to calculate parities, there is a pre-processing step required. The base matrix is of dimension 46 x 68 i.e., it contains 3128 entries. Out of these 3128 entries only 316 are valid processes. So, if we traverse the whole matrix to find the parities, we will be wasting 2812 cycles which is huge as compare to 316 required cycles. To overcome this wastage of cycles, the matrix is not going to be stored as it is, rather a transformation of the matrix is to be stored. For calculation of parities following is required from the matrix:

1. The row number which indicates parity number.
2. The column number selects bits required to calculate parity.
3. The shift value stored in the matrix.

The row number can be omitted as the parity number is not required because they are generated sequentially, and a counter can be used to get this information if required. The column number and the shift value are two vital requirements for parity calculation. In order to reduce number of cycles only valid positions of matrix can be stored in the memory. This causes loss of column and row number of the respective matrix value. Row number can be ignored but the column number is required and it can be stored with the matrix value so that whenever the value is read it also indicates the respective message bits required to calculate the parity. The bit order is given below:

NP	C1	C2	C3	C4	C5	S1	S2	S3	S4	S5	S6	S7	S8

NP → Next Parity

C → Column number

S → Shift value

4.1.2 Architecture Design:

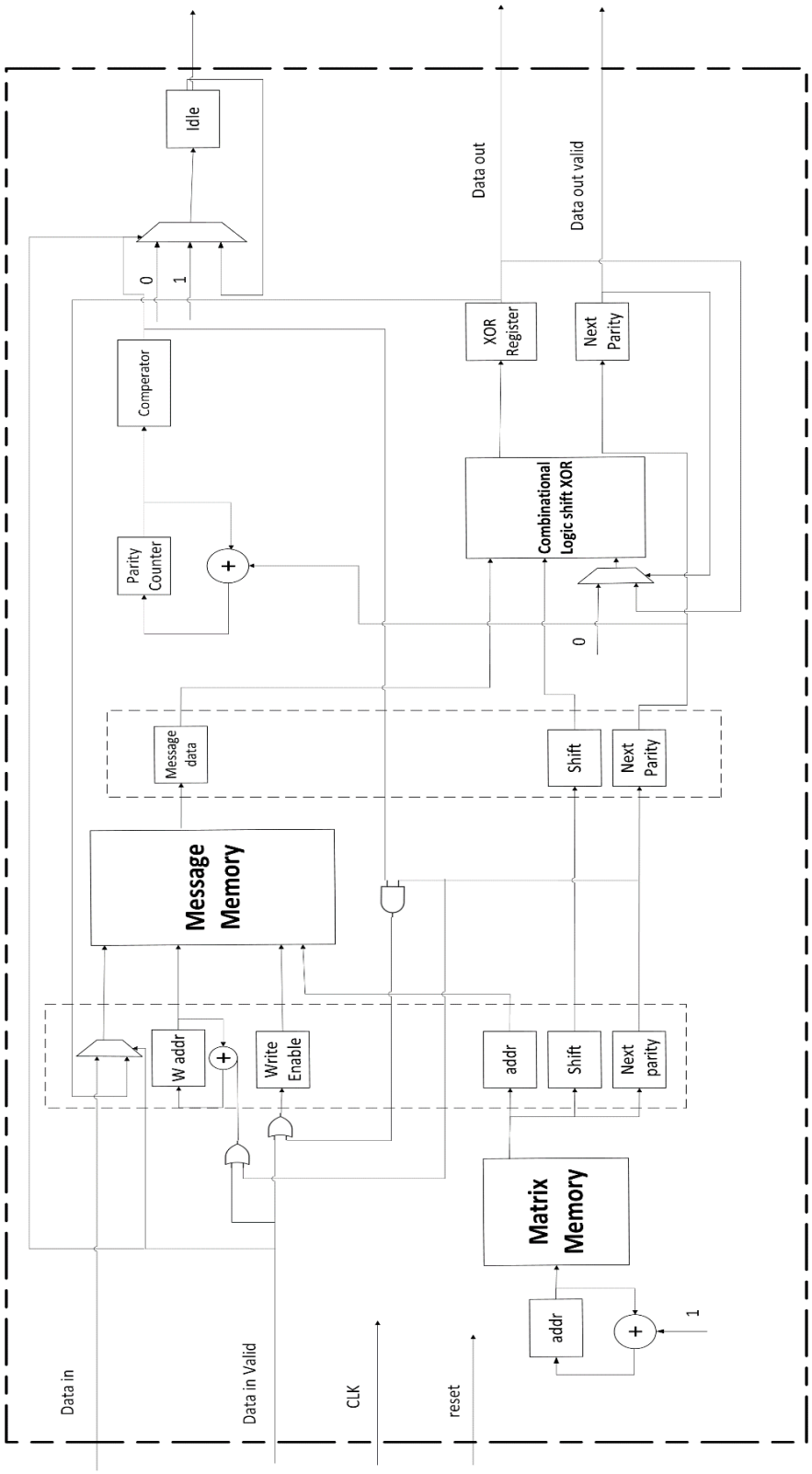

Figure 5 Encoder Architecture

4.1.3 Working:

This is explanation of Encoder Architecture showed in figure 5. The module continuously sends out an idle signal which indicates that it is ready to receive data to encode. Data can be sent after sending a data in valid signal and this signal must remain high till the time data is being sent. Data is received and stored in memory. As soon as the first group of bits arrive the encoder starts encoding. The shift and column values are read from the matrix memory. The column value is sent to message memory as read address. The data out of message memory and the shift value on next clock cycle are used to shift data circularly a specified number of time and the output value is xored with the previous value in the parity register. A single bit called next parity bit is used to indicate that a parity has been calculated. This causes the data out valid signal to become high and data is sent out. Simultaneously, parity registered is cleared to 0 in next clock cycle. After all the values have been calculated the system returns to idle state.

4.2 Decoder

Decoder is much more sophisticated and complicate piece of hardware as compared to encoder. It requires multiple iterations and multiple cycles per iteration to calculated codeword from a given LLR.

4.2.1 Pre-processing:

There are two-bit special formats used in decoder to reduce complexity of decoder and increase clock speed. First of all, LLR memory is arranged as following:

S	B6	B5	B4	B3	B2	B1	B0

First bit of LLR tells the sign of respective bit in codeword a 0 indicates a 0 in codeword and 1 indicates a 1 in codeword.

Arrangement of bits in Mins is as following:

P	M1	M1	M1	M1	M1	M2	M2	M2	M2	M2

M1 are the bits of first minimum and M2 are the bits of second minimum, while P is the parity of the whole row.

4.2.2 Architecture Design:

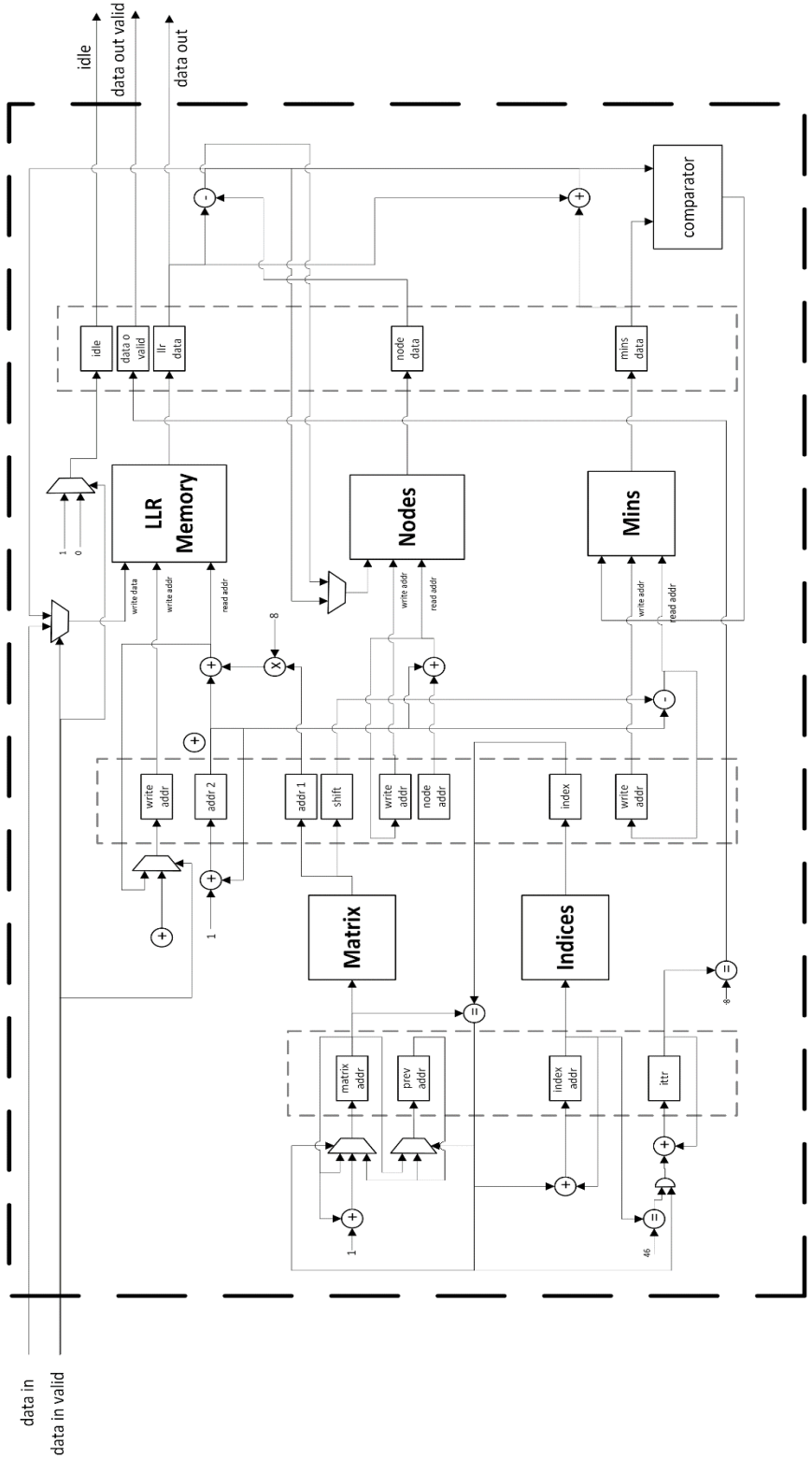

Figure 6 Decoder Architecture

4.2.3 Working:

This is explanation of Decoder Architecture showed in figure 6. The module continuously sends out idle signal, showing its availability to decode. Data can be sent to it after forcing the data in valid signal to 1. The data needs some clock cycles to arrive and be stored in LLR memory. After all the data has been received the decoding process can start. Each row in matrix is processed twice to update LLR. Firstly, Matrix memory sends out the address of LLR and Node data required to perform calculations i-e subtraction of node data from LLR, and the result is used by comparator to update minimums and it is also updated in node and LLR memories. After all the columns of row have been traversed, processing start from beginning and start to add minimum value to the LLR of the respective node. This whole process is performed for all 46 rows of the matrix for a total of 8 times each. After 8 iterations of the process the data out signal is set to high and first bit of LLR is sent out which is the required bit of the message module was able to decode.

4.3 Summary

In this chapter we discussed architecture design and Verilog implementation of encoder and decoder.

The complexity of encoder is lower as compared to that of decoder due to following reasons. The base matrix is of dimension 46 x 68 i. So, if we traverse the whole matrix to find the parities, we will be wasting 2812 cycles which is huge as compared to 316 required cycles. To overcome this wastage of cycles, the matrix is not going to be stored as it is, rather a transformation of the matrix is to be stored. This causes loss of column and row number of the respective matrix value.

Row number can be ignored but the column number is required, and it can be stored with the matrix value so that whenever the value is read it also indicates the respective message bits required to calculate the parity. The module continuously sends out an idle signal which indicates that it is ready to receive data to encode. Data can be sent after sending a data in valid signal and this signal must remain high till the time data is being sent. Data is received and stored in memory.

The data out of message memory and the shift value on next clock cycle are used to shift data circularly a specified number of time and the output value is xored with the previous value in the parity register. This causes the data out valid signal to become high and data is sent out.

In Decoder, M1 are the bits of first minimum and M2 are the bits of second minimum, while P is the parity of the whole row. Data can be sent to it after forcing the data in valid signal to 1. The data needs some clock cycles to arrive and be stored in LLR memory. After all the data has been received the decoding process can start.

Each row in matrix is processed twice to update LLR. Firstly, Matrix memory sends out the address of LLR and Node data required to perform calculations i-e subtraction of node data from LLR, and the result is used by comparator to update minimums and it is also updated in node and LLR memories. This whole process is performed for all 46 rows of the matrix for a total of 8 times each. After 8 iterations of the process the data out signal is set to high and first bit of LLR is sent out which is the required bit of the message module was able to decode.

Chapter 5: Results, Simulations and Tool used

This chapter shows the results and simulations performed for LDPC implementation for both MATLAB and Verilog.

5.1 MATLAB Simulations:

The implementation of LDPC codes on MATLAB helps get an overview of the performance of LDPC codes and the following results show the performance of LDPC decoder when iterated for 10,000 blocks each of size 4352 bits. The LLRs have been quantized to 8 bits and number of iterations performed for a single block is equal to 8. The numbers used here are the same as that are used for implementation in Verilog.

In the figure 7 & 8, Frame Error Rate/Block Error Rate is plotted vs EbNo in db. The y axis is in exponential scale and still the error rate is reducing very quickly, indicating very good performance of LDPC codes even at very low SNRs (Signal to Noise Ratios) as well as there humungous accuracy at average SNRs.

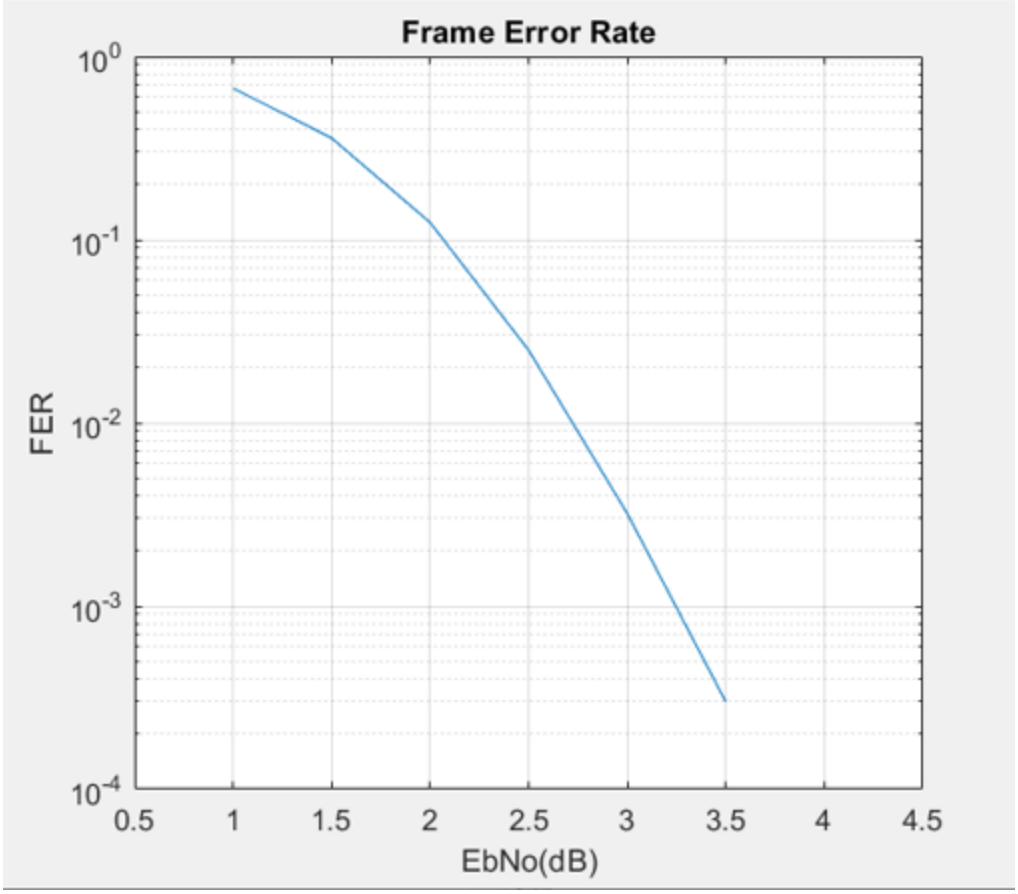

Figure 7 Frame Error Rate (FER)

The figure 8 has the same parameters as the previous one, but this plot is showing Bit Error Rate vs EbNo. The decay in Bit Error Rate is not as steep as in the plot above because once a received message has been distorted with very high noise the codeword may converge to a completely different codeword causing Bit Error Rate to be comparatively higher.

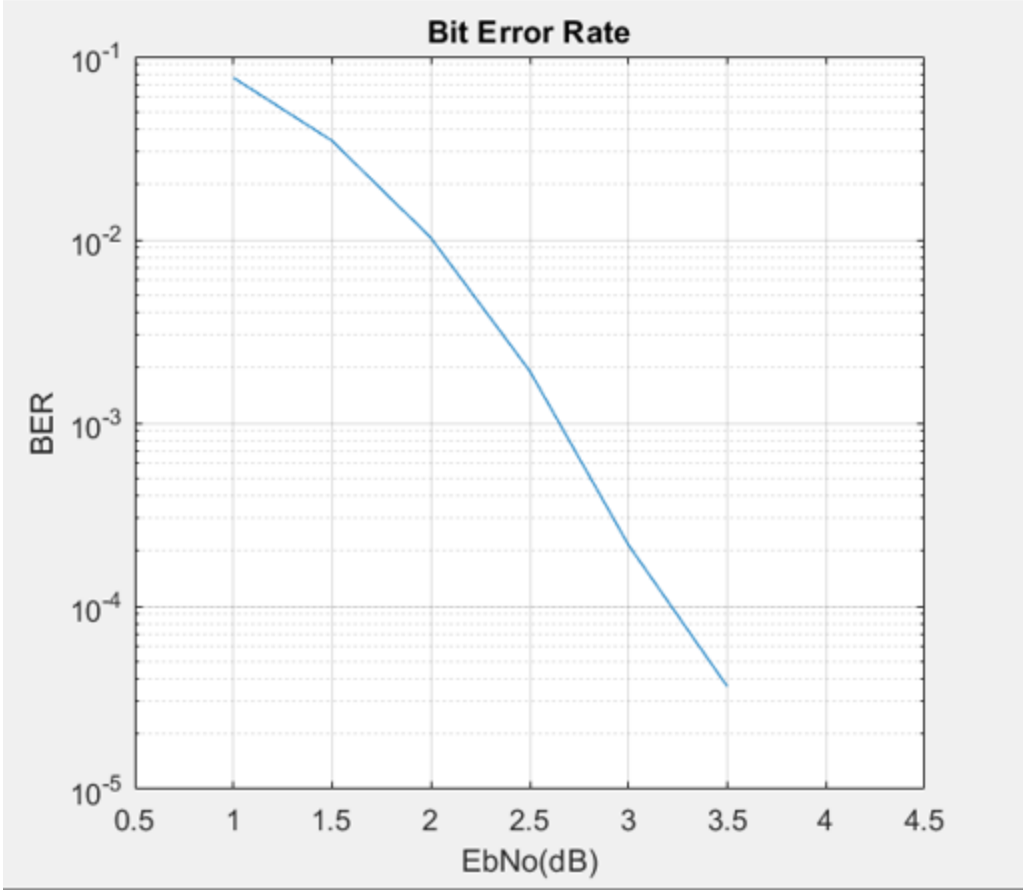

Figure 8 Bit Error Rate (BER)

5.2 Verilog Results and performance measurement:

The following figure shows the waveform of encoder output. The spikes in data out valid signal show that a parity has been calculated and is ready to be read at the data out output.

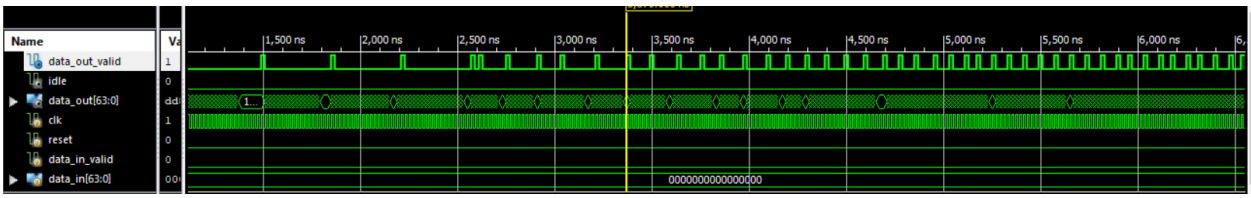

Figure 9 Output Waveform of Encoder

The result that we achieve from hardware design is the performance of the designed module. Table 1 is showing the results that we were able to achieve with the design implemented for Spartan 6.

Expansion Factor	Block Size (bits)	Clock Speed MHz	Throughput Mbps	Resources Used LUTs , Registers
2	136	317.25	90.29	41 , 46
4	272	310.71	176.59	46 , 52
8	544	280.79	319.82	55 , 76
16	1088	255.36	581.86	72 , 114
32	2176	190.01	865.92	73 , 164
64	4352	185.12	1686.21	118 , 356
88	5984	146.33	1833.53	618 , 138

Table (1) Encoder Performance Results

The results shown in table 1 depict the true power of FPGA and a good architectural design. The throughput was able to cross the barrier of 1Gbps at expansion factor of 64 while consuming as low as 2% of the resources available on FPGA.

Expansion Factor	Block Size (bits)	Clock Speed MHz	Throughput Mbps/cycle	Resources Used LUTs , Registers
2	136	130.31	28.04	141 , 146
4	272	102.44	44.09	246 , 252
8	544	88.56	76.22	455 , 476
16	1088	79.80	137.38	772 , 814

Table (2) Decoder performance Results

5.3 Hardware and Software Used

5.3.1 Hardware Components

Following are the hardware / electronic components used:

5.3.1.1 Spartan-6 FPGA SP605

The Spartan®-6 FPGA SP605 Evaluation Kit includes all of the necessary hardware, design tools, IP, and reference designs to get you up and running quickly. This package contains a versatile system design environment, as well as pre-verified reference designs and examples of how to use

technologies like high-speed serial transceivers, PCI Express®, DVI, and/or DDR3. PC

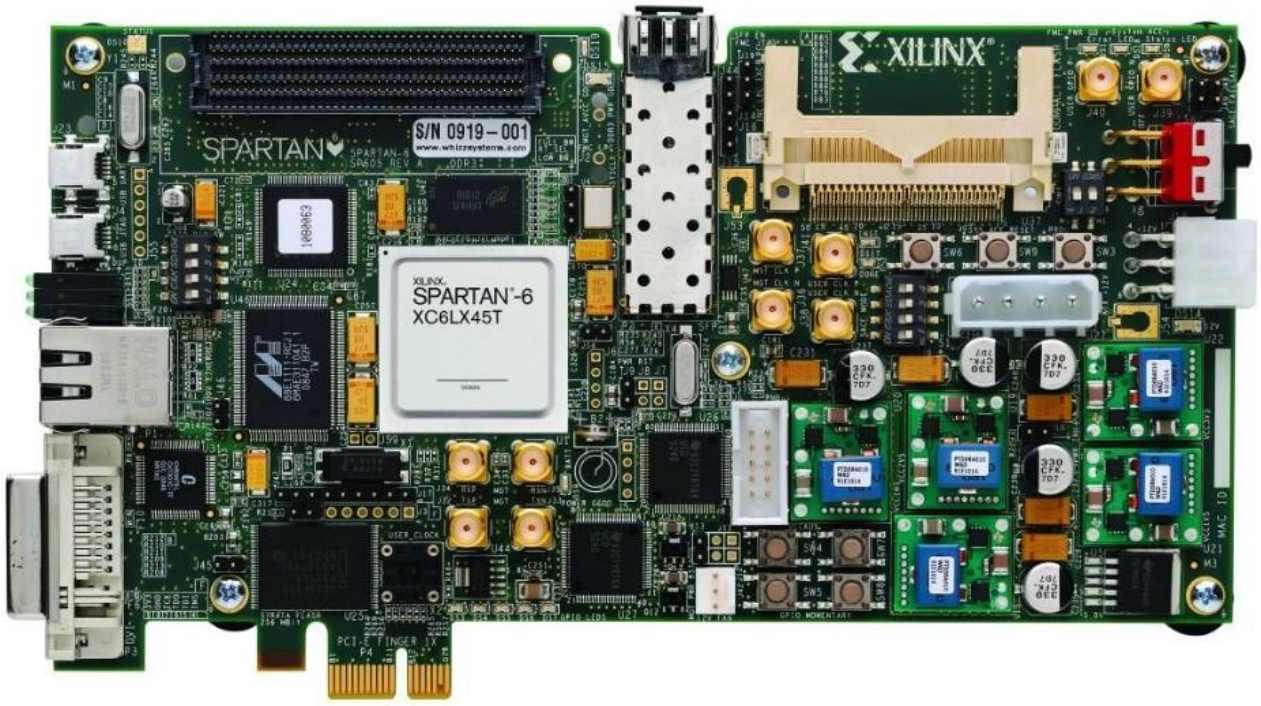

Figure 10 SPARTAN®-6 FPGA 605

Featuring the **Spartan 6 XC6SLX45T-FGG484-3C FPGA**

Logic Cells	43,661
Memory (Kb)	2,088
DSP Slices	58
3.2 Gb/s Transceivers	4
Maximum I/O	296

Table (3) Spartan-6 Features

Figure 11 XILINX® Logo

5.3.2 Software Tools Used

The software tools used were primarily for coding LDPC encoder and decoder and programming the hardware. Following are the software tools used:

5.3.2.1 Matlab

MathWorks' Matlab is a multi-paradigm programming language and quantitative calculation environment. Matlab provides matrix manipulation, function and data visualisation, algorithm implementation, user interface design, and interaction with other languages' programmes. At the core of MATLAB is the MATLAB language, a matrix-based language that allows for the most natural description of computer mathematics.

Figure 12 Matlab Logo

5.3.2.2 Visual Studio Code

Microsoft Visual Studio is an integrated development environment from Microsoft. It is employed in the development of computer programmes, websites, web apps, web services, and mobile applications. An integrated development environment is a feature-rich tool that encompasses

multiple aspects of software development (IDE). Before publishing an app, use the Visual Studio IDE as a creative starting point to change, debug, and create code.

Figure 13 Visual Studio Logo

5.3.2.3 Xilinx ISE Design Suite

Xilinx ISE is a Xilinx software tool for HDL synthesis and analysis that is primarily used to create embedded firmware for the Xilinx FPGA and CPLD integrated circuit families. Generally, the Xilinx ISE is used for circuit design and synthesis, whereas ISIM or the ModelSim logic simulator is utilised for system-level testing.

Figure 14 Xilinx ISE Logo

5.3.2.4 Digilent

Digilent Adept is a unique and powerful method for interfacing with Digilent system boards and a variety of logic devices. JTAG configuration and data transfer, board verification, and I/O expansion are all features of Adept. The procedure of connecting and opening devices is completely automated and has never been easier. These functions are accessible via a command-line interface in Adept Utilities.

Figure 15 Digilent Logo

5.4 Summary

This chapter showed the results and simulations performed for LDPC implementation for both MATLAB and Verilog.

The implementation of LDPC codes on MATLAB helps get an overview of the performance of LDPC codes and the following results show the performance of LDPC decoder when iterated for 10,000 blocks each of size 4352 bits. The LLRs have been quantized to 8 bits and number of iterations performed for a single block is equal to 8. The numbers used here are the same as that are used for implementation in Verilog.

This chapter also includes details of hardware and software tools used tools used

Chapter 6: Conclusion and Future Prospects

6.1 Conclusion

The Project was completed with all its goals achieved. A very sophisticated design for LDPC implementation has been produced achieving high throughputs and at low latency. Implementation of encoder and decoder on FPGA along with hardware acceleration also gave the advantage for the modules to be portable, adaptable, improvable and erasable. The digital design is generic and can also be used to develop ASICs.

6.2 Future Prospects

Although our identified work is done there are many aspects of hardware design and LDPC codes that can be improved upon and implemented using other techniques that are still advancing.

One of the most promising aspects is to implement multiple instances of the given modules on same FPGA. The modules given above have a very low resource utilization and during the calculations I/O ports are free. The free I/O ports can be used to distribute incoming data to different instances introducing module level parallelization.

For example, the encoder module given above takes 330 clock cycles to compute parity bits. During only 22 of these clock cycles input ports are used and output ports are used at max in 46 clock cycles. In this case a maximum of 330/46 ~ 7 instances can be implemented on same FPGA to take advantage of instance level parallelization.

References

[1] Survey of turbo, LDPC, and polar decoder ASIC implementations. Shao, Shuai, et al. "Survey of turbo, LDPC, and polar decoder ASIC implementations." *IEEE Communications Surveys & Tutorials* 21.3 (2019): 2309-2333.

[2] High Performance LDPC Decoder design using FPGA S. Pawankar and N. Mohota, "High Performance LDPC Decoder design using FPGA," *2019 9th International Conference on Emerging Trends in Engineering and Technology - Signal and Information Processing (ICETET-SIP-19)*, 2019.

[3] Design of LDPC Decoder Using FPGA Panigrahi, A. K., & Panda, A. K. (2012). Design of LDPC Decoder Using FPGA: Review of Flexibility. *International Journal of Engineering and Science*, *1*(7), 36-41.