Implementation of the Low complexity and High throughput LDPC Decoder

Dissertation submitted in partial fulfillment of the requirements

for the award of degree of

MASTER OF TECHNOLOGY

In

VLSI Design and CAD

Submitted By

Amrit Singh

Roll No. 601161002

Under guidance of

Dr. Sanjay Sharma

Professor, ECED

T.U, Patiala

Department of Electronics and Communication Engineering

THAPAR UNIVERSITY, PATIALA

June, 2013

DECLARATION

I hereby declare that the work which is being presented in the dissertation entitled, **"Implementation of Low complexity and High throughput LDPC decoder"** in partial fulfillment of the requirement for the award of degree of Master of Engineering in VLSI Design and CAD submitted in Electronics and Communication Engineering Department of Thapar University, Patiala, is an authentic record of my own work carried out under the supervision of **Dr. Sanjay Sharma, Professor, ECED** and refers other researcher's work which are duly listed in the reference section.

The matter presented in this dissertation has not been submitted in any other University/Institute for the award of degree.

Date: 14/06/13

Amrit Singh
AMRIT SINGH
Roll No: 601161002

It is certified that the above statement made by the student is correct to the best of my knowledge and belief.

Dr. Sanjay Sharma 14/6/2013
Professor
ECED, Thapar University

Countersigned by:

Head
ECED, Thapar University
Patiala-147004

Dean of Academic Affairs
Thapar University
Patiala-147004

ACKNOWLEDGEMENT

Words are often less to reveal one‘s deep regards. An understanding of the work like this is never the outcome of the efforts of a single person. I take this opportunity to express my profound sense of gratitude and respect to all those who helped me through the duration of this dissertation.

First, I would like to thank the Supreme Power, one who has always guided me to work on the right path of the life. Without his grace, this would never come to be today‘s reality.

I wish to express earnest acknowledgement, indebtness and gratitude to my respected Supervisor **Dr. Sanjay Sharma, Professor**, for his patient guidance and support throughout this dissertation. I am truly fortunate to have the opportunity to work with him. I found his guidance to be extremely valuable and his feedback and editorial comments quite generous and helpful for the writing this dissertation.

I am also thankful to our **Head of the Department, Dr. Rajesh Khanna, Professor** as well as **PG Coordinator, Dr. Kulbir Singh, Associate Professor,** and the entire faculty & staff of Electronics and Communication Engineering Department along with my friends who devoted their valuable time and helped me in all possible ways towards successful completion of this work. I thank all those who have contributed directly or indirectly to this work

Lastly, I would like to thank my parents for their years of unyielding love and encouragement. They have always wanted the best for me and I admire their determination and sacrifice

AMRIT SINGH

ABSTRACT

LDPC codes are appearing in an increasing number of applications. The main advantage of LDPC codes is that they provide a performance which is very close to the capacity for a lot of different channels. The near Shannon limit error correction capability has lead LDPC codes to become the coding technique of choice in many communication systems and storage systems since their introduction. Some of these applications include, among others, magnetic disc recording, deep space communication etc. LDPC is also used for 10Gbase-T Ethernet, which send 10 GB data on twisted pair cable. As these applications continue to evolve, the trend is shifting toward more and more strict requirements on power consumption, decoding throughput and hardware complexity as part of standard design practice, which has prompted for more efficient LDPC decoder implementations. Furthermore, the LDPC decoders are suited for implementations because that makes heavy use of parallelism. The feature of LDPC codes to perform near the Shannon limit of a channel exists only for large block lengths. The large block length results to increase in hardware complexity and decrease in throughput of decoder. The complexity of decoder multiplied as the length of codeword increase. The trade-off between the hardware complexity and the decoding throughput is a critical factor in the implementation of the practical decoder.

In this dissertation work, we proposed methods to decrease the hardware complexity and increase in decoding throughput. We have used partially parallel decoder architecture and reduced complexity algorithm to realize the LDPC decoder. Reduced complexity algorithm has greatly reduced the complexity of the decoder especially routing and check node complexity. The highly concurrent design of the decoder results into a maximum symbol throughput of 92.95 Mbps at maximum of 18 decoding iterations. In this work, we have implemented a 9216 bit, rate-1/2, (3,6) LDPC decoder on Xilinx XC3D3400A device from Spartan-3A DSP family and compare the hardware utilization and decoding throughput to the previously proposed (3,6) LDPC decoder. The design is simulated on ISim simulator and synthesized on Xilinx ISE design suite 13.1

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVATION

AWGN	Additive White Gaussian Noise
ALT	Approximate Lower Triangulation
AG	Address Generator
BSC	Binary Symmetric Channel
BEC	Binary Erasure Channel
BF	Bit Flipping
CV	Check-to-Variable
CNU	Check Node Unit
CNPU	Check Node Processing Unit
CPLD	Complex Programmable Logic Device
CNPE	Check Node Processing Elements
ECC	Error Correcting Coding
FPGA	Field Programmable Gate Array
IOB	Input Output Block
IEEE	Institute of Electrical and Electronics Engineers
LUT	Look-Up Table
LDPC	Low Density Parity Check
LLR	Log Likelihood Ratio
MSB	Most Significant Bit
MSA	Min-Sum Algorithm
PCM	Parity Check Matrix
PE	Processing Elements
QC – LDPC	Quasi-Cycle Low Density Parity Check
ROM	Read Only Memory
RTL	Register Transfer Level
RAM	Random Access Memory
SPA	Sum Product Algorithm
SALT	Systematic Approximate Lower Triangular
UMP	Uniformly Most Powerful
VNU	Variable Node Unit

VC	Variable-to-Check
VNPU	Variable Node Processing Unit
VHDL	Very High Speed Integrated Circuit Hardware Description Language
VLSI	Very Large Scale Integration

Chapter 1 INTRODUCTION

1.1 Overview

A communication system can transmit data from source to destination through the various channels such as: air, wires and optical fibers. The reliability of the data sent from the destination depends upon the channel used for communication and the external noise. The noise could distort the signal representing the data. In a digital transmission system, the nature of the channel largely affects the transmitted signal. Let we are transmitting the digital signal in '1' and '0' forms. In this, noise causes the distortion of the bits at the receiver end. So, elimination of noise is very much necessary.

A significant part of this advance in communication began some 60 years ago when Shannon published his paper "A Mathematical theory of Communication"[3]. In that paper, Shannon framed a fundamental question: how can we efficiently and reliably transmit information? Shannon also gave a basic answer: coding can do it. Channel coding is a process of signal transformation to improve communication performance by encoding signal in such a way that it can better withstand the effect of channel noise and fading. Channel coding is used to accomplish trade-offs between error performance and bandwidth or between power transmitted and error performance. Channel coding can be divided into two categories:

- Waveform Coding
- Structured Sequence

Here, we will discuss only structured sequence.

Structured Sequence: - These structure deals with transforming the data sequences into better sequences by introducing redundancy bits. These redundancy bits are used for error detection and correction. Encoding procedures provide the coded signals with better distance properties compared to original data. There are two ways by which we can use these redundancies to control errors:

- *Error detection and retransmission*: These techniques use redundancies to detect an error; If an error is detected, receive doesn't try to correct it rather send a request to

transmitter to retransmit the data. In this technique, a two way link is required between transmitter and receiver.

- *Forward Error Control*: In these techniques, only one way link is required for data transmission, since redundancies are used for both error detection and correction.

1.2 History and Background

The history of communications dates back to 3500 B.C when the Sumerians developed cuneiform writing and the Egyptians developed hieroglyphic writing. It was then that the human race started developing different ways of communicating their messages. The major breakthrough in telecommunications came when Samuel Morse invented the Morse code.

1.2.1 Timeline

The timeline of telecommunications is given below:

- 1793 - The Chappe brothers established the first commercial semaphore system between two locations near Paris.
- 1837 - Cooke and Wheatstone obtain a patent on telegraph. Morse publicly demonstrates his telegraph.
- 1843 - FAX invented by the Alexander Bain.
- 1844 - Electric telegraph was demonstrated by Samuel Morse. This was a major breakthrough in communications and marked the beginning of a new era.
- 1865 - James Clark Maxwell invents four simple electromagnetic equations that describe all known electromagnetic phenomena of the time.
- 1864 - James Clerk Maxwell proves that wireless telegraphy is possible.
- 1866 - The first Transatlantic telegraph cable is laid.
- 1876 - Alexander Graham Bell invents the telephone.
- 1895 - Guglielmo Marconi invented the radio.
- 1901 Marconi completes the first transatlantic radio transmission from Newfoundland.
- 1907 - The world's first transatlantic commercial wireless services is established by Marconi with stations at Clifden, Ireland and Glace Bay, Nova Scotia
- 1909 - Marconi gets a joint Nobel Prize in Physics, with Karl Ferdinand Braun because of their work in the development of wireless telegraphy.
- 1910 - Thomas Edison demonstrated the first talking motion picture.

- 1925 - John Logie Baird transmits the first experimental television signal.
- 1934 - Joseph Begun invents the first tape recorder for broadcasting first magnetic recording.
- 1940's - Spread Spectrum

Before 1948, communication was strictly an engineering discipline, with little scientific theory to back it up. Although a number of communication systems were in use at that time, communication engineering wasn't considered to be a hard science. Till 1948, there was no concept of 'information' and at that time, the transmission of audio signals and moving pictures was considered to be separate. In 1948, Claude E. Shannon known as, "The Father of Information Theory" published his ground breaking work and altered our basic think about communication. In his paper two major concepts were given:

1. The concept of information and the modeling of information sources
2. The concept of a channel and on the limits of reliable communication on unreliable channels

Given a communication channel, Shannon proved that there exists a number, called the capacity of the channel, such that reliable transmission is possible for rates arbitrarily close to the capacity, and reliable transmission is not possible for rates above capacity. He demonstrated that it is possible to achieve error free transmission on a noisy communication channel through coding. His work focuses on the problem of how best to encode the information a sender wants to transmit. This paper was the beginning of a new era in the field of error control coding. It was also proved in his paper that there exist codes which approach capacity, for which the probability of error at the decoder output goes to zero as the block length tends to infinity. Although capacity approaching codes are very good but the theorems given in the paper are non-constructive and do not offer any advice on how to find there codes. Soon after the publication of this paper, the researchers found out that random codes are capacity achieving. This spurred a lot of research activity in the field of error control coding and also gave the communication engineers the concepts of 'Information' and 'Channel'.

More than a decade ago in 1993, C. Berrou and A. Glavieux [10] presented a new scheme for channel codes decoding: the turbo codes, and their associated turbo decoding algorithm. Turbo codes made possible to get within a few tenth of dB away from the Shannon limit, for a bit error rate of 10^{-5}. This made other researchers realize the importance of iterative decoding and also

made them aware that other capacity approaching codes existed. Recently, LDPC codes have attracted much attention in the world of coding. They were originally invented in the 1960's and forgotten. They were again brought to life in 1995 by Mackay and Neal [2]. Unlike many other classes of codes LDPC codes are already equipped with very fast encoding and decoding algorithms. This makes LDPC codes not only attractive from a theoretical point of view, but also perfect for practical applications. They are already being implemented for satellite communication standards and 4G standards.

1.3 Necessity of coding.

- Reduce probability of error for finite block of symbol
- To approach toward the channel capacity

Consider the basic communication system as shown in figure-1.1. In this source produce R bits per second at fixed rate. The encoder is a device that prepares the data for transmission. We assume encoder makes the block of bits and each block has V bits. Encoder operates independently on each block. Encoder output then transmitted over channel and disturbed by the noise in channel. Decoder process the channels output and produce the delay version of the source bits [1].

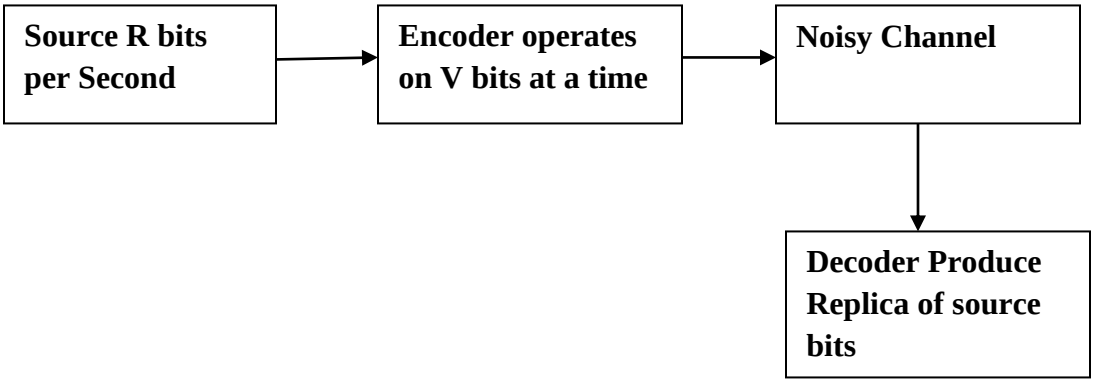

Figure-1.1: Block diagram of Communication System

If one element is sent on channel then it can be received in two possible ways, then good communication on channel does not occur. For this multiple element are sent at the place of one element. But if these elements are not correlated then it does not help us anymore. Due to this multiple correlated elements are sent.

Shannon theorem shows that, sequence of code, have capability of correcting all errors as the code length goes to infinity. Capacity of channel according to the Shannon is:-

$$C = W \log_2\left(1 + {}^{S}/_{N}\right)$$

OR

$$C = 1 - H(p)$$

Here, C = Channel capacity, it means one bit contain C part of information.

W = Bandwidth

S/N= Signal to Noise ratio

H(p) = entropy function of 'p' i.e. $[-p \log_2 p - (1 - p) \log_2(1 - p)]$

P = probability of error

So to increase the capacity, signal to noise ratio should be high i.e. noise should be less. In other words noise probability should be less.

According to noise channel coding theorem of the Shannon, the size of 'V' should be large. Due to this, coding schemes are used to increase the block length 'V' which in turn decrease the error probability but it is very difficult to implement practical encoder and decoder with very large block length.

1.4 Introduction to Error Correcting Coding and LDPC Codes

Error correcting coding (ECC) is a method to detect and possibly correct errors by introducing redundancy to the stream of bits to be sent to the channel. The Encoder will add bits to the message bits to be transmitted systematically. After passing through the channel, the decoder will detect and correct the errors. A simple example is repetition code which send '000' ('111' correspondingly) instead of sending only one '0' ('1' correspondingly) to the channel. Due to noise in the channel, the received bits may become '001'. But by majority logic decoding scheme, it will be decoded as '000' and therefore the message has been a '0'. As we increase the hamming distance of the code the possibility to detect correct logic is increase. In general the encoder will divides the input message bits into blocks of 'k' messages bits and replaces each 'k' message bits block into the codeword of 'n' bits by introducing (n-k) check bits to each message block.

ECC is an important part of modern communications systems, where it is used to detect and correct errors introduced during a transmission over a channel [9]. It relies on transmitting the

data in an encoded form, such that the redundancy introduced by the coding allows a decoding device at the receiver to detect and correct errors. In this way, no request for retransmission is required. In many applications, a substantial portion of the baseband signal processing is dedicated to ECC. The wide range of ECC applications includes space and satellite communications, data transmission, data storage and mobile communications [11]. NASA's space missions including Galileo, Odyssey, Rovers and Voyager would not have been possible without the use of ECC [12]. Odyssey, NASA's Mars spacecraft currently boasts the highest data transmission rate at 128,000 bits per second via a radio link. However, for future space missions NASA are planning to use optical communications via laser beams [13]. The new laser will beam back between one million and 30 million bits per second, depending on the distance between Mars and Earth. Projects like this provide great challenges to implement high-speed and low-power ECC systems with good error correcting performance in deep space.

In the past few years, LDPC codes have received much attention because of their excellent performance, and have been widely considered as the most promising ECC scheme for many applications in telecommunications and storage devices [14]. LDPC codes were first proposed by Gallager in 1962 [1].Due to the regular structure (uniform column and row weight) of Gallager's codes, they are now called regular LDPC codes. Gallager provided simulation results for codes with block lengths of the order of hundreds of bits. The results indicated that LDPC codes have very good potential for error correction. However, the high storage and computation requirements interrupted the research on LDPC codes.

1.5 Application areas of Error correcting codes

Main applications of Error correcting codes are:

- *Wireless and Mobile Communications*: Error correcting techniques are widely used in mobile communication systems to tackle with the problem caused by interference, noise, multi path fading, shadowing, propagation loss, etc. in the wireless channel. In GSM, Cyclic Redundancy Codes (CRC) are used for error detection, block and convolutional codes are applied for error correction. In CDMA2000, convolutional codes and turbo codes are used for error correction. In 3G, both convolutional and turbo codes are supported for error correction. Low-Density Parity-Check (LDPC) codes are the standard

error correcting codes for many wireless communication protocols such as WiMAX (802.16e) and WLAN (802.11).

- *Deep Space Communications*: Since space and the atmosphere is the channel for these communication systems, the space radiation is the most effective disturbance to the signal which can be modeled as Additive White Gaussian Noise Channel (AWGN). Although the noise sources are very limited, communication from outer space would be impossible because of the high propagation loss due to the distance. Thus, error correction is the only sensible way to communicate with signal powers as low as in space communication case. First of the codes that is used for deep space communications is Reed-Muller codes. After that, mostly convolutional codes and Reed-Solomon (RS) codes are used for space communications, with improvements in rate, gain, or time performance.
- *Satellite Communications*: For digital satellite TV, RS codes had been used for a long time, however currently replaced by modern codes such as LDPC and Turbo codes.
- *Military Communications*: In military communications, besides the natural interference and noise, the communication system also needs to deal with the intentional enemy interference. Therefore, the need for error correcting codes is obvious.
- *Data Storage*: RS codes are widely used for error correction in storage systems such as CDs, DVDs and hard discs. Single Error Correcting Double Error Detecting (SECDED) codes, which correct single errors and detect double errors, are widely used in many data storage units with RS codes or Hamming codes.

1.6 Channel models

One of the most important considerations in the design of any error-correcting coding scheme is the channel, or more precisely the channel model, error-correcting code is being designed for. It is important to design the coding scheme based on a channel model that accurately describes the statistics of the channel the code will eventually be used on. Firstly, we consider binary symmetric channel (BSC). Binary symmetric channel is consider as memory less channel i.e. if we give input to channel at particular time instant then we get delayed version of the signal at output and this output is independent of all other inputs or outputs. But there are certain probability that transmitted bit can be flipped as shown in figure-1.2 [1].

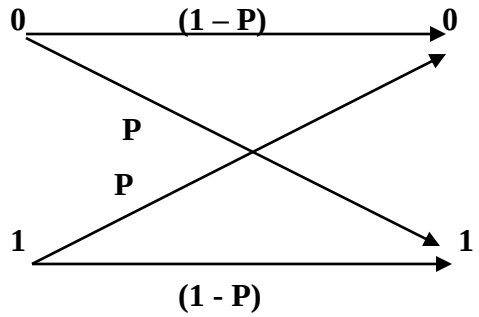

Figure-1.2: Simple model of BSC

Secondly, consider binary erasure channel (BEC). BEC is also a memory less channel. But there is certain probability that transmitted bit attains some random voltage level other than maximum or minimum voltage level as shown in figure-1.3.

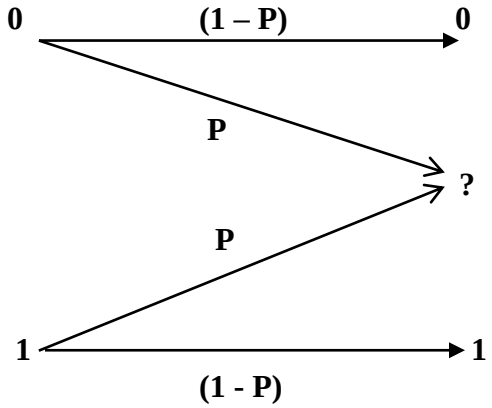

Figure-1.3: Simple model of BEC

1.7 Block codes

A binary linear block code is defined by three parameters: the block length ‘n’, the information length ‘m’, and the minimum hamming distance ‘d’ [15]. An (n, k) code is able to correct errors if its minimum distance satisfies the inequality $2t+1 \leq d$. The larger the minimum distance, better the performance of code. Finding a good code is an important issue in designing a communication system.

1.7.1 Simple Encoding

For the encoding, generator matrix (**G)** is required. Codeword is formed by matrix multiplication of message and generator matrix i.e. [5]

$$C = mG$$

Example-

$$C = mG = \begin{vmatrix} 1\,1\,0\,1 \\ 1\,0\,1\,1 \\ 1\,0\,0\,0 \\ 0\,1\,1\,1 \\ 0\,1\,0\,0 \\ 0\,0\,1\,0 \\ 0\,0\,0\,1 \end{vmatrix} \begin{vmatrix} 1 \\ 0 \\ 1 \\ 1 \end{vmatrix} = \begin{vmatrix} 0 \\ 1 \\ 1 \\ 0 \\ 0 \\ 1 \\ 1 \end{vmatrix}$$

Figure-1.4: Encoder matrix

1.7.2 Simple Decoding

For the decoding, parity-check matrix **(H)** is required. Original messag**e** signal is retrieved by matrix multiplication of received signal **(y)** and parity-check matrix. It uses Syndrome vector to determine error. If 'Z = 0' then reception is error free otherwise it define the position of error i.e. **Z = Hy** [5]

Example:-

$$Z = Hy = \begin{vmatrix} 1\,0\,1\,0\,1\,0\,1 \\ 0\,1\,1\,0\,0\,1\,1 \\ 0\,0\,0\,1\,1\,1\,1 \end{vmatrix} \begin{vmatrix} 0 \\ 1 \\ 1 \\ 0 \\ 0 \\ 1 \\ 1 \end{vmatrix} = \begin{vmatrix} 0 \\ 0 \\ 0 \end{vmatrix}$$

Figure-1.5: Decoder matrix

1.8 LDPC Codes

Low Density Parity Check (LDPC) codes are the class of block codes. LDPC were first introduced by Robert Gallager in 1962[1]. LDPC codes were ignored for many years because of

their high computational complexity for hardware technology at that time. Then LDPC codes were rediscovered by MacKay and Neal in 1990[2]. LDPC codes play a major role for the reliable communication system. LDPC codes are introduced as error correction codes that allow communication on the noisy channels near the Shannon capacity limits.

1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	1	1	1	1	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1
1	0	0	0	1	0	0	0	1	0	0	0	1	0	0	0	0	0	0	0
0	1	0	0	0	1	0	0	0	1	0	0	0	0	0	0	1	0	0	0
0	0	1	0	0	0	1	0	0	0	0	0	0	1	0	0	0	1	0	0
0	0	0	1	0	0	0	0	0	0	1	0	0	0	1	0	0	0	1	0
0	0	0	0	0	0	0	1	0	0	0	1	0	0	0	1	0	0	0	1
1	0	0	0	0	1	0	0	0	0	0	1	0	0	0	0	0	1	0	0
0	1	0	0	0	0	1	0	0	0	1	0	0	0	0	1	0	0	0	0
0	0	1	0	0	0	0	1	0	0	0	0	1	0	0	0	0	0	1	0
0	0	0	1	0	0	0	0	1	0	0	0	0	1	0	0	1	0	0	0
0	0	0	0	1	0	0	0	0	1	0	0	0	0	1	0	0	0	0	1

Figure-1.6: Robert Gallager matrix with j = 3 and k = 4

The name comes from the characteristic of the parity check matrix of LDPC codes i.e. there are low density of 1's number of 0's. Their main advantage is that they provide a performance which is very close to the capacity of channels. The LDPC matrix is represented in (n,j,k) form, where n represent the block length i.e. length of column, 'j' and 'k' are number of ones in each column and row respectively and else are all zeros. The code rate of the (j,k) matrix is 1 - j/k. Figure-1.6 shows the (20 × 15) LDPC parity check matrix presented by Robert Gallager. It has codelength of '20' and code rate is 1/4.

1.9 Representation of LDPC code

There are two ways by which we can represent the LDPC codes: -

1.9.1 Matrix representation

LDPC codes are defined by sparse parity check matrix. Below given the example of LDPC represent in the matrix form. Matrix is called low density matrix if two conditions is satisfied i.e. j << n and k << m as shown in figure-1.7 [5]. Here, 'j' is number of ones in each column and 'k'

is number of ones in each row. In order to meet these conditions, the low density parity check matrix should be large.

$$H = \begin{vmatrix} 0\,1\,0\,1\,1\,0\,0\,1 \\ 1\,1\,1\,0\,0\,1\,0\,0 \\ 0\,0\,1\,0\,0\,1\,1\,1 \\ 1\,0\,0\,1\,1\,0\,1\,0 \end{vmatrix}$$

Figure-1.7: Parity-check matrix

1.9.2 Graphical representation

LDPC can also represent with the connected set of graph called tanner graph. This graph is introduced by Tanner [5] [6] which effectively represent the LDPC code. Tanner graph directly maps the parity check matrix into nodes which is easy interpretable. This graph not only represents the LDPC code but also help to describe the decoding algorithm. There are two types of nodes in tanner graph called variable nodes and the check nodes. Variable nodes are equal to the number of columns and check nodes are equal to the number of rows in the parity check matrix. For codeword vectors of particular check node, sum of all vectors connected to particular check node is zero. Variable nodes and check nodes are connected to each other by undirected wire (edges) that is why it is also called as undirected bipartite graph. Number of edges connected to variable node is variable node degree and number of edges connected to check node is check node degree Figure-1.8 is tanner graph of parity-check matrix of figure-1.7. The graphical representation is analogous to matrix representation. The check node is connected to message node if the element of parity-check matrix **H** is 1.

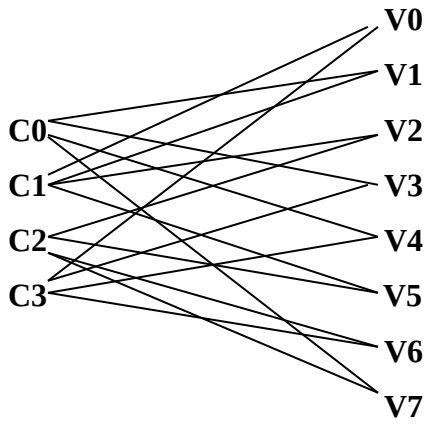

Figure-1.8: Tanner graph representation of LDPC parity check matrix

1.10 Regular and Irregular LDPC codes

LDPC codes can be Regular LDPC codes if the number of 1's in column (j) and row (k) is constant for every column and row. Example matrix is (3, 6) matrix. It's also possible to see the regularity of code while looking at graphical representation. There are same number of edges for every variable node and check node. LDPC codes can be regular if for given parity check matrix (n × m), code satisfy the condition, $k = j\ (^{n}/_{m})$. LDPC codes can be Irregular LDPC codes if the number of 1's in column (j) and row (k) is not constant. It means that degree of node for different variable and check nodes are different. The degree of variable node and check node are chosen according to some distribution.

1.11 Parameters for LDPC code design

Below are brief definition of the parameters and there effects on the performance and implementation.

- *Code Size*: This is used to specify the dimension of the parity check matrix. Generally a code is specified using its length and row-column weights in the form (N , j , k). It is shown that very long codes perform better than shorter ones [2] [16]. Long codes are therefore used where good performance is required. However, its implementation requires more resources like memory, processing nodes etc.
- *Code structure*: This define pattern of the connection between the rows and columns i.e. it define the connection between the check node and variable node in tanner graph. The pattern of the connection defines the hardware complexity of LDPC encoder and decoder. Random codes do not have any predefined connections between check and variable nodes but structured codes have known interconnections between check and variable nodes.
- *Code rate*: The rate of code, R, is defines the ratio of the length or number of bits containing information to the total length of codeword. It is expressed as 1 – j / k. High rate of code means less redundancy bits i.e. more number of information bit is transmitted resulting in higher throughput. However, less redundancy means less protection of bits and therefore less decoding performance and high error rate [17]. Low rate code has high redundancy bits i.e. less number of information bit is transmitted resulting in lower throughput. However, high redundancy means more protection of bits and therefore more decoding performance and low error rate.

- *Number of iterations*: It means the number of times received bit are estimated before a hard decision is made by decoding algorithm i.e. number of times the received bits passes through the pair of check nodes and variable nodes. Large number of iteration causes the increase in decoder performance but this also increases the decoder delay and the power consumption. For practical applications 20 to 30 iterations are commonly used. [19] - [21]
- *Minimum Hamming Distance*: Hamming distance of two codewords is the number of bits with which words differ from each other. Minimum hamming distance is smallest hamming distance between two codewords. Number of error corrected is represented through hamming distance i.e. $t \leq [(d - 1)/ 2]$. Here,'t' represents number of error corrected and 'd' represents the hamming distance. Therefore, larger the hamming distance better the performance of the code. A good code can be determined by the minimum hamming distance of the code. However, for randomly constructed code, it is difficult to find minimum hamming distance.

1.12 Construction of LDPC codes

LDPC codes construction defines a pattern of the connection between the rows and columns of parity check matrix i.e. it define the connection between the check node and variable node in tanner graph. Code is constructed with rate of code and code length kept in mind. The main objectives in code construction are good decoding performance and easier hardware implementation. However, making of codes for having low hardware complexity may degrade or limit decoding performance. It is difficult to obtain a wide range of codes in length and rate that have good decoding performance and are also easy to implement in hardware. Construction of LDPC codes can be following type

1.12.1 Random construction

Random means unstructured row-column connections. Random constructions have flexibility in design and construction but lack in row-column connections regularity, which increases decoder interconnection complexity. Random constructions connect rows and columns of a LDPC code matrix without any predefined connection pattern. They are actually pseudo-random connections. Constructions could be done in the Tanner graph by connecting check to variable nodes with edges or in the parity-check matrix by connecting rows to columns with '1' entries where all

other entries are ‘0’s. Randomly addition of edges to a Tanner graph or ‘1’ entries in the parity-check matrix could be optimized by putting constraints on the random choices. This constraint causes the construction of Random LDPC matrix of desired girth and rate. Random codes have good performance especially at long code lengths. Random construction methods are used to maximize performance [1] [16].

1.12.2 Structured Construction

Structured means row-column connections predefined in some way. Structured constructions have regular interconnection patterns but often structured codes are limited in rate, length and girth. The main objectives are to achieve good performance and at the same time have a connection pattern that is easier to implement in hardware. In this method, constructions could be done by placing the edges in the Tanner graph connecting check to variable nodes or ‘1’ entries in the parity-check matrix connecting rows to columns to some predefined pattern. There are many structured methods already developed, producing codes differing in row-column interconnection pattern, performance and hardware implementation complexity.

1.13 Quasi cycle LDPC codes

Simple LDPC codes do not have the same pattern of connections for all rows or columns. Hence the row-column interconnections pattern increases. Therefore, many interconnections patterns generally increase complexity of a decoder. So, Quasi-cycle (QC) LDPC is introduced in which rows or columns in a sub-matrix have similar and cyclic connections. Row-column connections in QC-LDPC codes are constructed by shifting identity sub- matrices. The complexity of QC-LDPC decoder is very much decreased because their decoder architectures require simple address generation mechanisms, less memory. Although row-column connections are constrained QC-LDPC codes can perform close to the capacity limit as demonstrated in [24].A QC-LDPC code can be formed by concatenation of circularly shifted sub-matrices with or without zero sub-matrices. The simple structure of QC-LDPC code is

$$H = [B_1, B_2, \dots\dots, B_k]$$

Where, B_k is a circulant matrix. As figure-1.9 shown, P_{xy} is L × L sub-matrix. This sub-matrix can be identity matrix or shifted identity matrix. Figure-1.10 has shown the matrix with zero sub-matrices whereas ‘0’ represent the zero sub-matrix of size L × L. A shifted identity sub-matrix is obtained by shifting each row of an identity sub-matrix to the right or left by some amount. In

figure-1.9 the number of sub-matrices in a row is equal to the matrix row-weight and equal to column weight in a column. Such structures have been shown to have a maximum girth of twelve. In figure-1.10 the number of sub-matrices is greater than row and column weights. This structure could be the result of irregular code designs or decoder architecture.

P_{11}	P_{12}	P_{13}	P_{14}
P_{21}	P_{22}	P_{23}	P_{24}
P_{31}	P_{32}	P_{33}	P_{34}

Figure-1.9: Quasi-cyclic code with all non-zero sub-matrices

P_{11}	0	P_{12}	0	P_{13}	0	P_{14}	0
0	P_{21}	0	P_{22}	0	P_{23}	0	P_{24}
P_{31}	P_{32}	0	0	0	P_{33}	P_{34}	0
0	0	P_{41}	0	P_{42}	0	P_{43}	P_{44}
0	P_{51}	0	P_{52}	0	0	P_{53}	P_{54}
P_{61}	0	P_{62}	P_{63}	0	P_{64}	0	0

Figure-1.10: Quasi-cyclic code with zero sub-matrices

Construction of the QC-LDPC codes can be random and structured. In the Random construction, the sub-matrices are randomly shifted to the right or left by some amount. But this may causes the reduction of performance due to existence of four-cycle. By putting some constraint on the shifting of the sub-matrix, this problem can be avoided. In the structured construction, the shifting amount for the sub-matrices is precisely calculated. The amount of shifting determines the code performance.

1.14 LDPC encoding

A new technique for efficient encoding of LDPC codes based on the known concept of approximate lower triangulation (ALT) is introduced. The greedy permutation algorithm is presented to transform parity-check matrices into an approximate lower triangular (ALT) form with minimum gap. Encoding with low complexity is not straightforward, as LDPC codes are defined by their Parity Check Matrix (PCM) and the generator matrix is generally unknown.

The conventional way is systematic encoding with the generator matrix derived from PCM by Gaussian elimination. The method is the same for any block code, thus it has high complexity. Rather than finding a generator matrix for H, an LDPC code can be encoded using the parity-check matrix directly by transforming it into upper triangular form and using back substitution. First, convert the parity check matrix into Lower Triangular form as in figure-1.11[7]

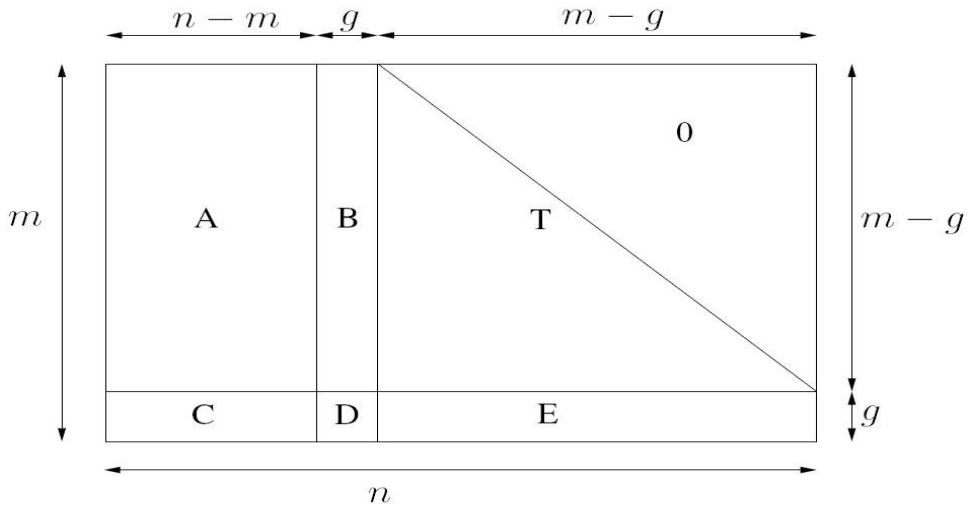

Figure-1.11: Lower triangular form

In a second step we keep the matrices A, B and T and we transform the matrix E into an all-zero matrix and the matrix D into an identity matrix. The resulting form is "Systematic Approximate Lower Triangular" (SALT) form. As shown in Figure-1.12 [7] [39].

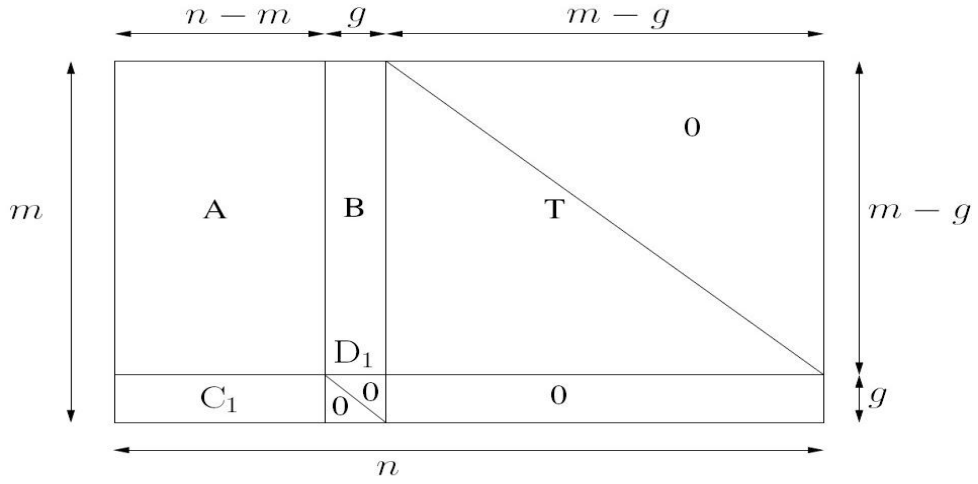

Figure-1.12: Systematic Approximate Lower Triangular form

Here,

$C_1 = - ET^{-1}A + C$

$D_1 = - ET^{-1}B + D$

For codeword we use this SALT form. In this E has been cleared P_1 depend only on the message bits, and so can be calculated independently from the parity bits in P_2. It is

$$P_1 = D_1^{-1} Cm$$

Here "m" is Message bits.

Once P_1 is known P_2 can be found from

$$P_2 = -T^{-1} (Am + BP_1)$$

1.15 LDPC decoding algorithm

Before description of algorithm, let we have **H** parity check matrix for LDPC codes. It has '**V**' variable nodes and '**C**' check nodes. Here, $H_{i,j}$ denotes the entry (i,j) at the **H** matrix. Let we have 'n' set of variable node which participate in m_{th} check node as $C(m) = \{n : H_{m,n}=1\}$, and 'm' set of check node which participate in n_{th} variable node as $V(n) = \{m : H_{m,n}=1\}$. C(m) \ n denote set of C(m) bits excluding 'n_{th}' and V(n) \ m denote set of V(n) bits excluding 'm_{th}'.

1.15.1 Bit-flipping decoding algorithm

For understanding the decoding, let we have error free code i.e. x = **[10010101]**. Now introduce error in this code i.e. x = **[11010101]**. Here, second bit is flipped and tanner graph for parity matrix of LDPC code is shown in figure-1.13

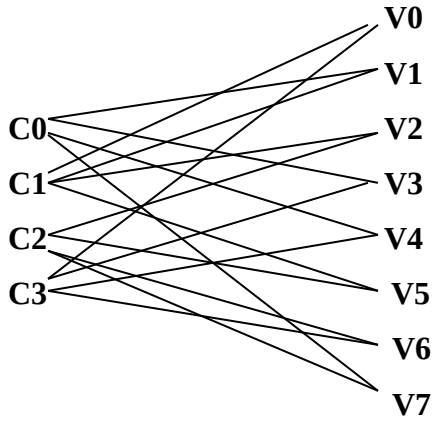

Figure-1.13: LDPC parity check matrix

The algorithm to find the error:-

Check – node operation: In this we have to compute check-to-variable messages ($CV_{m,n}$) for each check node.

$$CV_{m,n} = VC_{m,1} \text{ xor } VC_{m,2} \text{ xor } \dots\dots\dots \text{xor } VC_{m,k}, \qquad \text{where}, k \in C(m) / n$$

For given example, as in Table-1.1 check node C0 receive '1' from V3, '0' from V4, '1' from V7 then it send '0' to V1 which is EXOR of V3, V4 and V7. Similarly, check node C0 receive '1' from V1, '1' from V3, '1' from V7 then it send '1' to V4 which is EXOR of V1, V3 and V7. If all the equations are satisfied, it means we receive error free code. So the algorithm terminates and no need to go to the next step but if any of equation is not satisfied, it means there are errors in the received code and that is why go to next step of algorithm[8].

Check node	Activity				
C0	Receive	$VC_{1,1}$ (V1) = 1	$VC_{1,2}$ (V3) = 1	$VC_{1,3}$ (V4) = 0	$VC_{1,4}$ (V7) = 1
	Send	$CV_{1,1} = 0$	$CV_{1,2} = 0$	$CV_{1,3} = 1$	$CV_{1,4} = 0$
C1	Receive	$VC_{2,1}$ (V0) = 1	$VC_{2,2}$ (V1) = 1	$VC_{2,3}$ (V2) = 0	$VC_{2,4}$ (V5) = 1
	Send	$CV_{2,1} = 0$	$CV_{2,2} = 0$	$CV_{2,3} = 1$	$CV_{2,4} = 0$
C2	Receive	$VC_{3,1}$ (V2) = 0	$VC_{3,2}$ (V5) = 1	$VC_{3,3}$ (V6) = 0	$VC_{3,4}$ (V7) = 1
	Send	$CV_{3,1} = 0$	$CV_{3,2} = 1$	$CV_{3,3} = 0$	$CV_{3,4} = 1$
C3	Receive	$VC_{4,1}$ (V0) = 1	$VC_{4,1}$ (V3) = 1	$VC_{4,1}$ (V4) = 0	$VC_{4,1}$ (V6) = 0
	Send	$CV_{4,1} = 1$	$CV_{4,2} = 1$	$CV_{4,3} = 0$	$CV_{4,4} = 0$

Table-1.1: Check-to-variable message generation

Variable – node operation: In this operation, computation of the variable-to-check messages ($VC_{m,n}$) for each Variable node is done. The variable-to-check messages and decision bit are determined through majority rule.

$$VC_{m,n} = \begin{cases} 0 \text{ if, majority}(CV_{i,n}, x_n) = '0' \\ 1 \text{ if, majority}(CV_{i,n}, x_n) = '1' \\ VC_{m,n} \text{ otherwise,} \end{cases}$$

Where, $i \in V(n)$

Take a look at message node V1 in Table-1.2. It receives 2 0's from check nodes C0 and C1. Together with the input of decoder i.e. 1, it decides that its real value is 0. It then sends this information back to check nodes C0 and C1. Again repeat these operations until either exit occur or a certain number of iterations have been passed [8].

Message nodes	Output of channel	Messages from check nodes		Decision
V0	1	$CV_{2,1}$ (C1) = 0	$CV_{4,1}$ (C3) = 1	1
V1	1	$CV_{1,1}$ (C0) = 0	$CV_{2,2}$ (C1) = 0	0
V2	0	$CV_{2,3}$ (C1) = 1	$CV_{3,1}$ (C2) = 0	0
V3	1	$CV_{1,2}$ (C0)= 0	$CV_{4,2}$ (C3) = 1	1
V4	0	$CV_{1,3}$ (C0)= 1	$CV_{4,3}$ (C3) = 0	0
V5	1	$CV_{2,4}$ (C1)= 0	$CV_{3,2}$ (C2) = 1	1
V6	0	$CV_{3,3}$ (C2)= 0	$CV_{4,4}$ (C3) = 0	0
V7	1	$CV_{1,4}$ (C0)= 1	$CV_{3,4}$ (C2) = 1	1

Table-1.2: Hard-decision generation by majority logic

1.15.2 Soft decoding algorithm

This decoding is similar to Bit-flipping decoding, except it uses mathematical equations: -

- Let P^1_n is probability that x_n = '1' and for x_n = '0' probability is $P^0_n = (1 - P^1_n)$.
- Here, x_n are the n_{th} input bit and 'n' is number of variable nodes.

Initialization: At first, we compute variable-to-check messages ($VC_{m.n}$) for each variable node at starting of decoding. This message is:

$$VC_{m,n}(0) = P^0_n$$

And

$$VC_{m,n}(1) = P^1_n$$

Check – node operation: In this we have to compute check-to-variable messages ($CV_{m,n}$) for each check node. Every message contains a pair of $CV_{m,n}(0)$ and $CV_{m,n}(1)$ which means that probability of output of check node is 0 or 1 respectively. We also have $CV_{m,n}(0) + CV_{m,n}(1)$ = '1'. $CV_{m,n}(0)$ is also the probability that there are an even number of 1's on all variable-to-check message except the sending variable node.

First, let consider the probability that there are an even number of 1's from 2 variable nodes. Let $VC_{m,1}(1)$ be the probability that there is a '1' at variable node V1 and $VC_{m,2}(1)$ be the probability that there is a '1' at variable node V2.

$$\Pr[V1 \text{ Xor } V2 = 0] = VC_{m,1}(1).\, VC_{m,2}(1) + (1 - VC_{m,1}(1))\,(1 - VC_{m,2}(1))$$
$$= ½[\,1 + (1 - 2\,VC_{m,1}(1))\,(1 - 2\,VC_{m,2}(1))]$$

So we can write, the probability for even number of 1's on 'n' variable nodes is

$$\Pr[(V1 \text{ Xor } V2 \ldots\ldots \text{Xor } Vn)] = ½ + ½ \prod_{k=0}^{n}(1 - 2VC_{m.k}(1))$$

Now, message for check node to variable node

$$CV_{m,n}(0) = ½ + ½\prod_{k\epsilon C(m)\,\setminus\, n}(1 - 2VC_{m.k}(1))$$

And $$CV_{m,n}(1) = 1 - CV_{m,n}(0)$$

Variable – node operation: In this operation, computation of the variable-to-check messages ($VC_{m,n}$) and decision bit for each Variable node is done. Every message contains $VC_{m,n}(0)$ and $VC_{m,n}(1)$ pair which means probability that input is '0' or '1'. $VC_{m,n}(0)$ + $VC_{m,n}(1)$ = '1'. The variable-to-check message is:

$$VC_{m,n}(0) = Ki.(1 - P^1{}_n)\prod_{j\epsilon V(n)\,\setminus\, m} CV_{j,n}(0)$$

$$VC_{m,n}(1) = Ki.P^1{}_n\prod_{j\epsilon V(n)\,\setminus\, m} CV_{j,n}(1)$$

Here *Ki* is such that $VC_{m,n}(0)$ + $VC_{m,n}(1)$ = '1'

- If $VC_{m,n}(0) > VC_{m,n}(1)$ then output decision is '0' otherwise output decision is '1'

1.15.3 Log-Belief-propagation (BP) decoding algorithm

If we directly apply BP decoding algorithm then complexity is very high due to use of various multiplication unit. So to reduce the complexity, Log-BP decoding algorithm is introduced. It reduces the complex multiplication operation in simple addition operations [18] [23].

- Let $P^1{}_n$= Pr[x_n = 1] be the conditional probability that x_n is '1' and $P^0{}_n$= Pr[x_n = 1] be the conditional probability that x_n is '0'
- Here, x_n are the n_{th} input bit and 'n' is number of variable nodes.

Initialization: At first, we compute variable-to-check messages ($VC_{m.n}$) for each variable node with likelihood ratio (L_n) = $\ln\left(p_n^0 / p_n^1\right)$.

$$VC_{m.n} = \text{sign}(L_n)\ln\left(\frac{1 + e^{-|L_n|}}{1 - e^{-|L_n|}}\right)$$

Where, sign(L_n) is '+1' when $L_n \geq 0$

sign(L_n) is '-1' when $L_n < 0$

Check – node operation: In this we have to compute check-to-variable messages ($CV_{m,n}$) for each check node.

$$CV_{m,n} = \ln\left(\frac{1 + e^{-|\beta|}}{1 - e^{-|\beta|}}\right) \prod_{k \epsilon C(m) \setminus n} \text{sign}(VC_{m,k})$$

Where, β is $\sum_{k \epsilon C(m) \setminus n} |VC_{m,k}|$

Variable – node operation: In this we have to compute variable -to- check messages (VC_m) for each variable node.

$$VC_{m,n} = \text{sign}(L_{m,n})\ln\left(\frac{1 + e^{-|L_{m,n}|}}{1 - e^{-|L_{m,n}|}}\right)$$

Where, $L_{m,n} = L_n + \sum_{k \epsilon V(n) \setminus m} |CV_{k,n}|$

- Log-likelihood ratio (LLR) is update to

$$¥_n = L_n + \sum_{k \epsilon V(n)} CV_{k,n}$$

- Output decision is '0' if $¥ > 0$ and '1' if $¥ \leq 0$.

1.16 LDPC decoder architecture

LDPC decoding is done using hardware to speed up processing. The large size of LDPC codes, wide range of rates and unstructured interconnection patterns are some of the characteristics that make hardware implementation a challenge. The architecture of the check nodes and variable nodes varies from the algorithm used for implementing decoder. The connection between the check node and variable nodes are done by varies type of components such as memory, crossbar switches, buses etc. Decoding equations can be implemented in different forms, including approximations and look up tables. LDPC decoders are often classified according to the number of processing nodes (check and variable nodes). Three classifications based on this criterion are:

1.16.1 Fully Parallel architectures

A fully parallel architecture resembles the tanner graph of any code completely as shown in the figure-1.14. Each node of the Tanner graph is mapped into a processing unit. These nodes are connected by wires as per the check-variable node connection in the graph. The number of check processing nodes is equal to the number of rows and the number of variable processing nodes is equal to the number of columns in the parity check matrix. The check-to-variable and variable-to-check message passes to nodes through interconnection network.

This architecture provides the maximum number of computation nodes and communication wires so it is fast. There is no queuing or scheduling of messages for computation or communication. In this architecture, the number of decoding cycles is less than other architectures. With this architecture the throughput depends on the node and interconnection implementations. But the implementation of large codes the interconnection could be very complex and costly. Each '1' or row-column connection in the code matrix represents an interconnection between processing elements i.e. variable and check nodes.

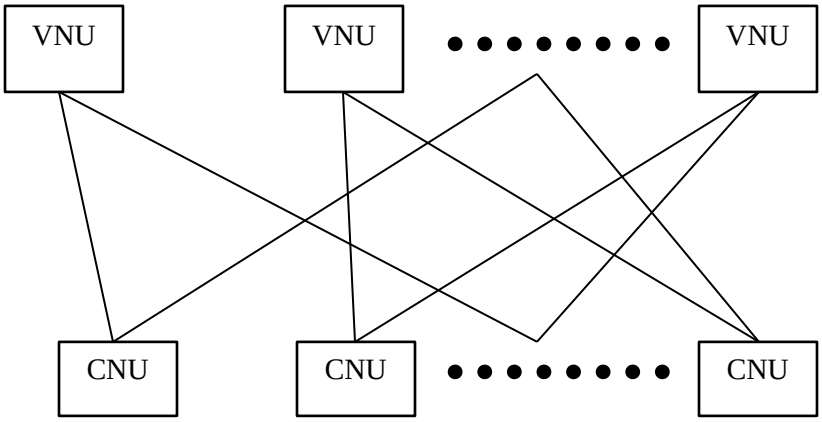

Figure-1.14: Fully parallel LDPC decoder architecture

The routing complexity between variable and check nodes is further increased if we random code construction method. In this architecture, interconnect occupied half of the chip area as compared to whole decoder. Complexity of decoder is also depends upon the size of the code. As the size of the code increases the complexity of the architecture also increases.

1.16.2 Serial architectures

As the size of code increases fully parallel architecture has maximum hardware utilization. So, it is impossible to use fully parallel architecture where minimum hardware is required. Serial

architectures, as shown in figure-1.15, have one check and one variable node computation units. The check node unit processes one row at a time and the variable node process one column at a time. As in fully parallel architectures the interconnection network could be bidirectional or unidirectional

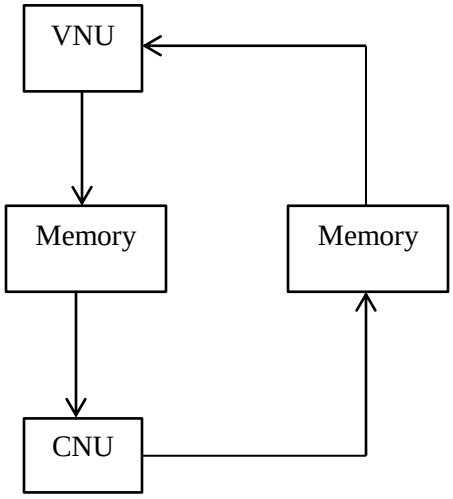

Figure-1.15: Serial LDPC decoder architecture

Figure-1.15 shows two unidirectional connections. Serial architectures are very flexible and less costly compared to fully parallel architecture. But, it's main disadvantage is that it has very low throughput and it requires many decoding cycles to decode one iteration.

1.16.3 Partially parallel architectures

Partially parallel architecture made a trade-off between complexity and throughput obtain from fully parallel and serial architecture. It has more throughput than serial architectures and less complexity than fully parallel architectures. Several processing nodes are implemented with an interconnection network as shown in Figure-1.16. The number of variable node and check node units is much smaller than the number of rows and columns in parity check matrix. Hence, several columns or rows are mapped onto a single node. Partially parallel architectures are often based on structured codes but can be used for random codes. The choice of interconnect network and scheduling of messages depends on the structure parity check matrix of code. Partially parallel architectures have reduced cost and complexity compared to fully parallel architectures as there are fewer nodes but have less throughput. Partially parallel architecture has more hardware complexity than serial architecture as there are more nodes but have high throughput. Throughput can be traded with hardware cost to suit target application.

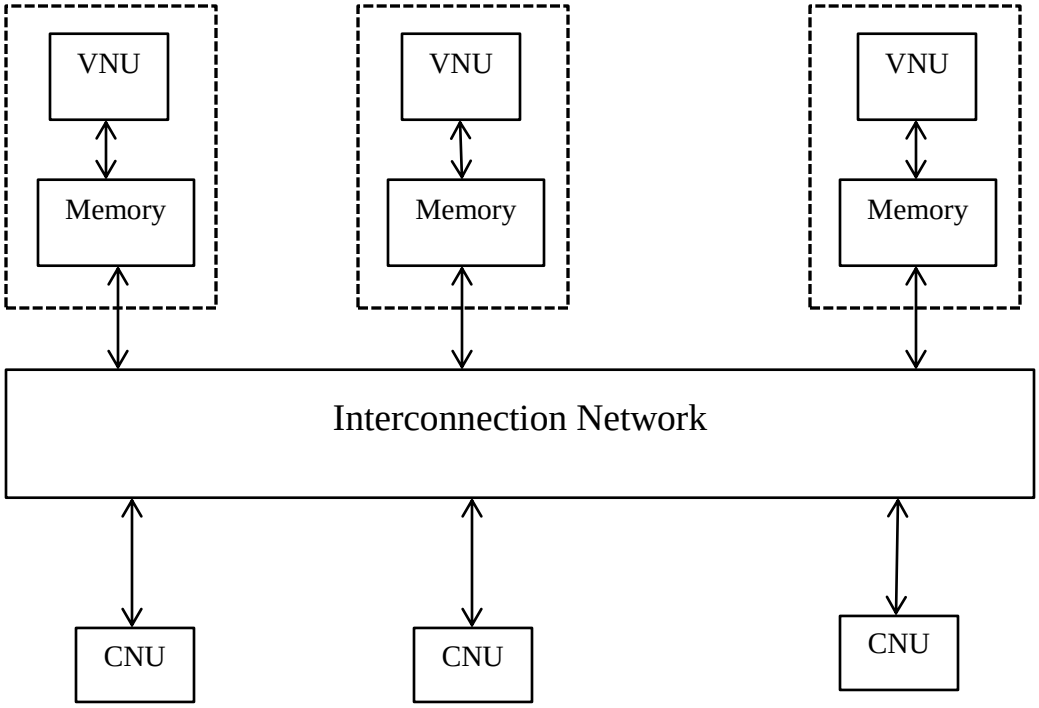

Figure-1.16: Partially Parallel LDPC decoder architecture

Comparison of different LDPC decoder architecture is shown in Table- 1.3

Feature	Fully parallel architecture	Serial Architecture	Partially parallel architecture
Hardware complexity of processing node	High	Lowest	Low
Throughput	High	Lowest	Low
Hardware complexity of control logic	Low	Highest	High
Hardware complexity of memory requirement	None	Highest	High
Main advantage	High throughput	Lowest hardware complexity	Low hardware complexity
Main drawback	Interconnection complexity	Throughput and memory requirement	Memory requirement

Table-1.3: Architectural Comparison

1.17 FPGA (Field Programmable Gate Array)

A field programmable gate array is a semiconductor device containing programmable logic components and programmable interconnects. The programmable logic components can be programmed to duplicate the functionality of basic logic gates such as AND, OR, XOR, NOT or more complex logic functions such as decoders or simple mathematical functions. In most FPGAs, these programmable logic components (or logic blocks, in FPGA parlance) also include memory elements, which may be simple flip-flops or more complete blocks of memories.

Field-Programmable Gate Arrays (FPGAs) have long been used for implementing the logic on slices. More recently, they are being used for many real-life applications including communications, encryption, video image processing, medical imaging, network security and numerical computations. FPGA arrived in 1984 as an alternative to programmable logic devices (PLDs) and ASICs. FPGA offers the significant benefits of being readily programmable. FPGA can be programmed again and again.

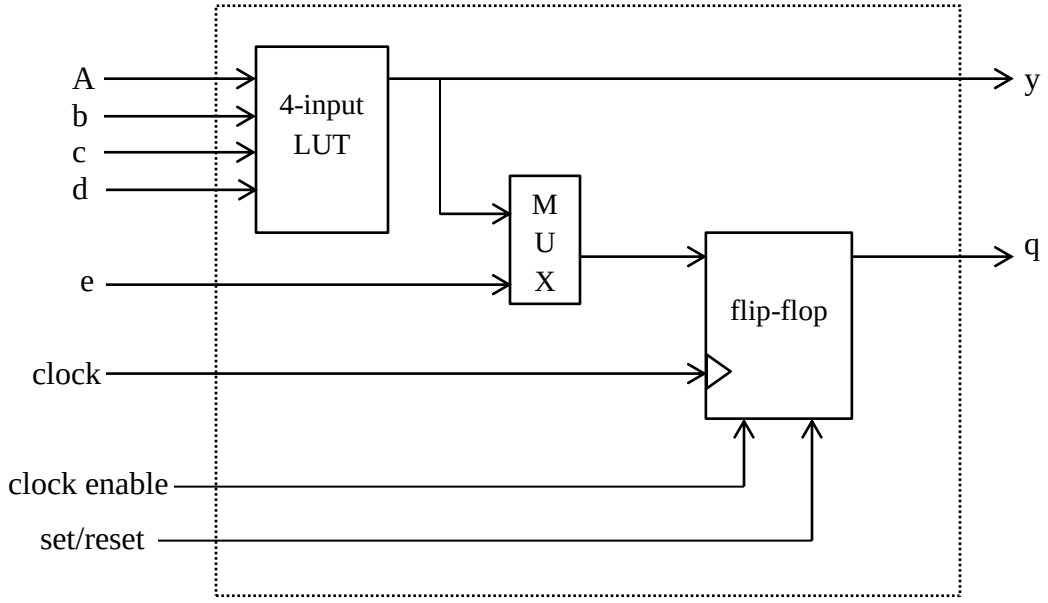

Figure-1.17: Simplified logic of Xilinx logic cell

In recent years, the size of FPGAs has followed Moore's law: the number of logic gate doubles every 18 months. FPGAs can exploit improvements following Moore's law better than microprocessors because of their simpler and more regular structure. The fundamental building block of Xilinx FPGAs is the logic cell. A logic cell comprises a 4-input look-up table (which can also act as a 16 × 1 RAM or a 16-bit shift register), a multiplexer and a register. A simplified view of a logic cell is shown in figure-1.17 [46]. Two logic cells are paired together in an

element called a slice. A Slice contains additional resources such as multiplexers and logic to increase the efficiency of the architecture. These extra resources are equivalent to having more logic cells, and therefore a slice is counted as being equivalent of 2.25 logic cells.

The architecture of FPGA is shown in Figure-1.18. In general, an FPGA will have an array of configurable logic blocks (which contain two or four slices depending on the FPGA family), programmable wires, and programmable switches to realize any function out of the logic blocks and implement any type of interconnection. In addition to logic blocks, some FPGAs contain embedded hardware elements for memory, multiplication, multiply-and-add etc. Clock management is facilitated by on-chip PLL (phase-locked loop) or DLL (delay-locked loop) circuitry. Dedicated memory blocks can be configured as basic single-port or dual port RAMs, ROMs, FIFOs. Now days FPGAs are system building resource as such as high-speed serial input/output, arithmetic modules, embedded processors, and large amount of memory.

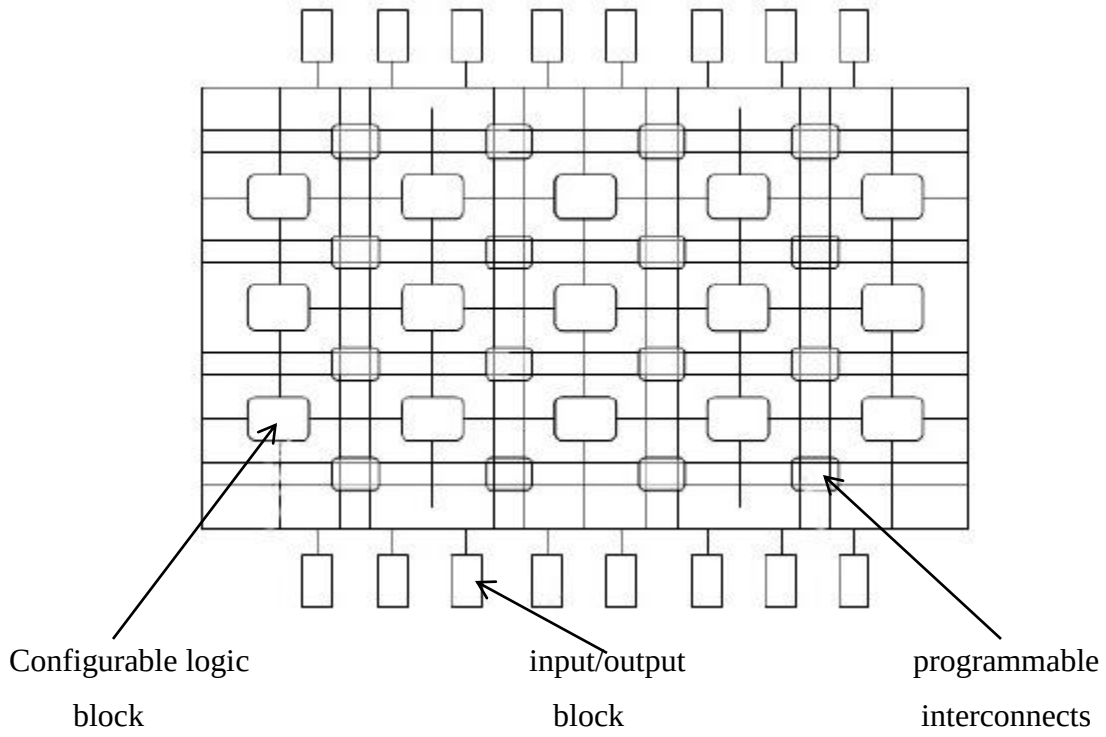

Figure-1.18: Architecture of FPGA

The long IC fabrication time is completely eliminated for these devices and design realization times are only a few hours. The idea of user-programmability is very exciting, most ASIC users now prefer FPGAs. Also, from a marketing point of view, the FPGA technology allows quick product announcements, which is commercially attractive. The two major FPGA vendors are Altera and Xilinx.

1.18 LDPC application

LDPC codes have been applied to applications in communication and storage systems. LDPC code design, construction and implementation are depends upon the target applications. LDPC code required for storage systems are very high rate i.e. $^{8}/_{9}$ and higher, low SNRs i.e. 7 to 12dB. These high data rates cause the system to be faster and throughput is in range of Giga-bits per second range (Gbps). LDPC code design for storage system, column-weight of codes is carefully selected because of their low complexity of decoder i.e. few edge connections. Low complexity codes such as quasi-cyclic codes are important as disk storages are also sensitive to cost of VLSI. Communications standards such as digital video broadcasting version 2 (DVB-2) recommend a variety of rates, $^{1}/4$ to $^{8}/9$, and very long code lengths of 3200 and 64800. LDPC codes are also recommended for other communication environments such as Gigabit Ethernet, wireless broadband and optical communications.

1.19 Problem Formulation

Large codes require more hardware in terms of memory communication network and processing nodes. LDPC decoder memory required depends on the structure of the code, implementation architecture and decoding algorithm. Interconnection complexity between nodes and large memory are the main difficulties in hardware decoder implementation especially for fully parallel and random codes. Partially-parallel decoder architectures are often used to implement structured and random codes. They offer a better tradeoff between critical factor like decoding throughput, hardware cost and hardware complexity. Various algorithms are also proposed to trade-off between these critical factors like min-sum algorithm, iterative log-BP algorithm etc.

1.20 Objective

The main objectives are:

- Reducing hardware utilization of the LDPC decoder
- Increasing the decoding throughput of decoder
- Maximum utilization of source available

1.21 Dissertation Outline

The dissertation is organized as follow

- *Chapter 1* describes some background of the LDPC codes, hardware resources, FPGAs, decoding algorithms, architectures of decoders etc.
- *Chapter* 2 is dedicated to the literature survey. In this, work done by many earlier researchers has been presented and discussed the main ideas contained in them.
- *Chapter 3* describes the various simulation and synthesis tools used to implement LDPC decoder. This chapter also defines the flow of design for implementation.
- *Chapter 4* present the implemented algorithm and architecture for (3,6)-LDPC decoder. This chapter define the each elements used in the implementation.
- *Chapter 5* defines the meaningful results, FPGA implementation of decoder, maximum resource utilization, maximum symbol throughput and comparing the results with other architecture.
- *Chapter 6* concludes this dissertation, summarizing the major results and offering suggestions for further work on this topic.

In the end, a list of references to this dissertation is given, without which this work could not have been possible.

Chapter 2 LITERATURE SURVEY

In this chapter the work done by many earlier researchers has been presented and discussed the main ideas contained in them.

2.1[25] Tong Zhang et. al. has described A 54 MBPS (3,6)-regular FPGA LDPC decoder

Based on the joint code and decoder design methodology, in this work a 9216-bit, rate- $^1/_2$ (3,6)-regular LDPC decoder is developed and this decoder is implemented on Xilinx FPGA device. Partly parallel decoder design is used to achieve appropriate trade-offs between hardware complexity and decoding throughput. In this work, a novel concatenated scheme is used to realize the random connectivity. This scheme uses two concatenated routing networks, which causes significantly reduce the random interconnections. Log belief-propagation (BP) algorithm is used to realizes the (3,6)-regular LDPC decoder. Partially-parallel decoder architecture for (3,6)-regular LDPC decoder is designed with the VHDL language and SYNOPSYS FPGA express is used to synthesize the decoder architecture implemented in VHDL. The synthesized design is place and route by the Xilinx Development System tool suite for the target XCV2600E device with the speed option -7. The number of slices utilized by the design is 11,762 and slices utilized by all the CNU (check node unit) and VNU (variable node unit) is 74% of the total utilized slices i.e. 8691 slices. Maximum clock frequency utilized, determined by the post-routing static timing analysis is 56 Mhz. If *s* decoding iterations is required to decode one frame, then total clock cycles for decoding one frame will be $2s \cdot L + L$. Here the extra L clock cycles is due to the initialization process. The maximum symbol throughput will be $56 \cdot k^2 \cdot L / (2s.L+L) = 56 \cdot 36 / (2s+1) = 54$ Mbps at maximum decoding iteration is 18.

2.2[26] Yijun Li et. al. has described Power efficient architecture for (3,6)-regular low density parity check code decoder

Low-Density Parity-Check (LDPC) code have widely used in telecommunications and magnetic storage because of its good error performance and hardware implementation. Critical factor for the design of LDPC decoder architecture is to increase throughput and reducing hardware complexity. In most of the decoder architecture power analysis is almost neglected. In fact,

power consumption is also a critical issue in computing, portable and mobile platforms, because of battery life and power dissipation. So in designing LDPC decoder architecture there must be a trade-off among throughput, power consumption and hardware complexity. This work, analyses the power consumption of the (3,k)-regular LDPC decoder architecture. In the partially parallel decoder architecture the 95% of the power consumption is consumed in accessing the memory i.e. in the memory read/write operations. In this work, a new architecture for (3,6)-Regular LDPC Decoder is proposed. This architecture has less interconnection complexity and also reduces the read/write operation from memory which causes the reduction in power consumption. This architecture reduces the power consumption by 14% and also has lower hardware complexity.

2.3[27] Thirupathi M et. al. has described VLSI Implementation of Very High Speed LDPC Decoder

High throughput is a key factor that affects the architecture of the LDPC decoder. For high throughput applications, the decoding parallelism is usually very high. Hence, a complex interconnection network is used which consumes a significant amount of silicon area and power. The QC-LDPC codes are very suitable for the long codewords because it has excellent error correction performance, regularity in their parity check matrices and interconnection complexity of QC-LDPC decoder is very less than any another structured or random code. In this work, high throughput architecture for the QC-LDPC code is proposed. An approximate layered decoding technique increase clock speed and hence increases the decoding throughput. LDPC Decoder computes the check-to-variable messages, variable-to-check messages, and LLR messages corresponding to an entire row of 'H' matrix in one clock cycle. So, there are three stages of pipelining is introduce in this that causes increase in the decoding throughput.

2.4[28] Luca Fanucci et. al. has described A throughput/complexity analysis for the VLSI implementation of LDPC decoder

The complexity and throughput is the critical factor which affects the design of LDPC decoder. The effect of these factors is different for the different LDPC decoder architectures. In this work, different architectures are compared and determine architectural issues, hardware complexity, power consumption and throughput. For comparison between different architectures (fully and

partially parallel architecture) a ½ rate, (3,6) regular LDPC code with a 2048 codeword is used and area complexity estimations is done in 0.18 um CMOS technology.

2.5[29] Wen-Yao Chen et. al. has described the Error Correction Capability of Bit-Flipping Algorithm for LDPC Codes

The error correction capability of bit-flipping decoding algorithm for low density parity-check (LDPC) codes is studied by Tanner graphs of LDPC codes. The bit-flipping (BF) algorithm is a simple and easy to implement. Bit-flipping (BF) algorithm is a iterative hard-decision algorithm for low density parity-check (LDPC) codes over binary symmetric channels. In each iteration, the decoder computes the check sums for each variable node and a variable node is said to satisfy the flipping criterion if it has more than or equal to a threshold number of unsatisfied check sums. This paper shows that for code having column weight '¥' and girth g = 8, than the weight of error pattern is than or equal to "¥ - 1" can be corrected.

2.6[30] Yukui Pei et. al. has described Design of Irregular LDPC Codec on a Single Chip FPGA

In this paper implementation of irregular Low density parity check (LDPC) codec on a single chip Xilinx FPGA is given. Partially-parallel architecture is used for encoder and decoder design. The structure of encoder and decoder, memory management, and computation units of variable and check node are depends on the parity-check matrix used. The Error correcting capability of the proposed scheme is kept the same as that by random generation method. The proposed design approach for irregular LDPC code is very good for real applications because proposed scheme provides high speed LDPC codes with powerful error-correcting capability on a single chip FPGA.

2.7[31] David Haley et. al. has described Reversible Low-Density Parity-Check Codes

In this we discuss the concept of using decoder circuit again as encoder of LDPC code. It's mainly has following benefits: -

- Reduction in area requirement for the use of encoder and decoder
- Increase in probability of detecting an implementation error during testing
- Help in verification that the generated codeword is correct or not

Hard-decision decoding and Soft- decision decoding are performed. The main benefit of Hard-decision decoding is that it is reversible i.e. we can use it has encoder again. But in case of soft-decision decoding it is not possible. By decoder reuse idea, they developed a reversible LDPC codes. It means that same factor graph can be used for the encoder purpose. So, iterative matrix inversion method is proposed. Reuse the decoder architecture cause to reduce the size of CODEC. In this reuse wire routing occurs. Routing reuse cause the consumption of same wire for encoder and decoder purpose. It also describes the parallel implementation of the circuit, which decrease the latency.

2.8[32] Marc P. C. Fossorier, et. al. has described Reduced Complexity Iterative Decoding of Low-Density Parity Check Codes Based on Belief Propagation

Iterative decoding means the number of times received bit are estimated before a hard decision is made by decoding algorithm i.e. number of times the received bits passes through the pair of check nodes and variable nodes. It should also require that algorithm have low complexity and high speed. In this work, two simplified versions of the belief propagation algorithm are proposed. These algorithms have fast iterative decoding of low-density parity check codes. These algorithms are more suitable for the additive white Gaussian noise channel. These two algorithms do not require any prior knowledge about the channel characteristics. Both algorithms have good trade-off between the performance and complexity. These algorithms can be implemented in software as well as in hardware. These two algorithms are:

- UMP APP-based decoding algorithm
- UMP BP-based decoding algorithm

Here, UMP means Uniformly Most Powerful

2.9[33] Hao Zhong et. al. has described Joint Code-Encoder-Decoder design for LDPC coding system VLSI implementation

Joint construction of code, encoder and decoder hardware causes the suitable implementation of code. In this work, approach for low-density parity-check (LDPC) coding system hardware implementation is given by jointly conceiving irregular LDPC code construction and VLSI implementations of encoder and decoder. The main reason of jointly constructing code is to construct irregular LDPC codes with good error performance and to ensure the effective

implementation of LDPC encoder and decoder hardware. This proposed approach provides a complete systematic solution for LDPC coding system hardware implementation

2.10[34] Abdessalam.Ait madi et. al. has described Design Simulation and Hardware implementation of Low Density Parity Check Decoders using Min-Sum Algorithm

In Min-Sum Algorithm (MSA) decoding of Low Density Parity Check (LDPC), Variable Node Processing Unit (VNPU) and a Check Node Processing Unit (CNPU) are most important processing elements. The architecture of these units is fully parallel. In this work, VNPU and CNPU have been designed and implemented in software using Simulink tool. Then, VNPU and CNPU were designed and simulated in Very High Speed integrated circuits Hardware Description Language (VHDL). Same test vectors are applied to the VNPU and CNPU, designed in VHDL via a test bench written in VHDL and Simulink tool. Comparison between these two implementations is done that shows high level modular approach (Matlab, Simulink and Modelsim) can be used to test and validate digital circuits (VNPU and CNPU) before being implemented on desired Field Programmable Gate Array (FPGA) device.

2.11[35] Christian Spagnol, William Marnane, Emanuel Popovici, et. al. has described Reduced complexity FPGA implementation of quasi-cyclic LDPC decoder

Quasi-cyclic (QC) LDPC codes are codes in which rows or columns in a sub-matrix have similar and cyclic connections. In this work, FPGA implementation of particular quasi-cyclic LDPC code decoder is done and careful design of partially-parallel decoder architecture causes to takes full advantage of the structure of the code and the hardware resources of FPGA. In partially-parallel architecture the memory access is most important operation. In this work, α^2 architecture is proposed which reduced the complexity of QC-LDPC decoder architecture. In this architecture, efficient management of the memory access is done by a simple memory controller that reduces the complex memory control to a small number of adders and multiplexes. The α^2 architecture allows the reduction of the area and of the power consumed by the FPGA implementation of the decoder. This architecture causes reduction in Slices, 4 input LUT, block RAM, Minimum period of FPGA and increase in throughput as compared to other architectures such as serial architecture, partially-parallel architecture.

2.12[36] Emmanuel Boutillon et. al. has described Decoder-first code design

As we going to design any error correction system then, first we construct a code for this system and after that we define the hardware structure of the decoder for this code. But this not a good method for develop the system because there are very little chance that code is suited for a hardware implementation. In this work, firstly an efficient hardware structure is chosen and secondly, a code is constructed that adequately fits this structure. Such a methodology is tested on LDPC codes of 4096 bits and rate of code is ½. This work results that determining the hardware architecture prior or generation of code jointly with the code, leads to make an efficient high speed and low complexity hardware decoders for LDPC codes. This "decoder first code design" approach causes the excellent performances of the LDPC decoder.

2.13[37] Andrew J. Blanksby et. al. has described A 690-mW 1-Gb/s 1024-bits, Rate-1/2 Low-Density Parity-Check Code Decoder

In this work, implementation of 1024-bits, rate-1/2, soft decision low-density parity check (LDPC) code decoder is given which has performance equivalent turbo codes. The parallel architecture of the decoder supports maximum throughput of 1 Gbps at maximum of 64 decoder iterations. In this work proposed decoding algorithm causes low switching activity of decoder that results in power dissipation of only 690 mW from a 1.5 V supply. Belief propagation algorithm is used to implement the LDPC decoder. This decoder was designed in a 0.16 um, 1.5V CMOS process with five levels of metal. The variable and check node were synthesized from a VHDL description. In this proposed architecture, 1024 variable nodes were grouped into 16 variable node shift register groups, each containing 64 variable nodes. Each of the shift register groups is used to give input to the 64 variable nodes. The total gate count of the decoder was 1750K gates with the variable node and shift register logic contributing 986K gates and the check node logic requiring 686K gates

2.14[38] Houshmand Shirani-Mehr et. al. has described A Reduced Routing Network Architecture for Partial Parallel LDPC Decoders

In this paper, partial parallel decoding method is purposed which has low routing complexity of interconnections. This method is based on the LDPC matrix proposed in IEEE 802.15.3c and IEEE 802.11ad standards. This proposed method results in improvements in area, speed and

power, with an identical error correction performance because it causes complete elimination of logic gates on the routing network. Decoder of (672,588) LDPC matrix is implemented which is adopted in the IEEE 802.15.3c is implemented in a 65 nm CMOS technology. The proposed method causes reduction in 96% of gates on the routing network, that causes 1.26× improvement in area and 1.3× improvement in throughput.

2.15[39] T. Richardson et. al. has described Efficient Encoding of Low-Density Parity-Check Codes

In this paper, the authors have shown the method of exploiting the sparseness of the parity check matrix to obtain efficient encoders. As it is known that the complexity of encoding is essentially quadratic in the block length. In this method, the associated coefficients are made very small so that the encoding of codes of length 100,000 is practically possible. It is also shown in this paper that optimized codes actually admit linear time coding.

2.16[40] Vikram Arkalgud Chandrasetty et. al. has described FPGA Implementation of a LDPC Decoder using a Reduced Complexity Message Passing Algorithm

In this work, a simplified message passing algorithm is proposed for reduce the implementation complexity of (LDPC) decoder. This algorithm uses simple hard-decision decoding techniques which also utilizing the soft information to be processed for improvement in decoder performance. The results show that the simplified message passing algorithm algorithm can achieve improvement in bit error rate (BER) performance and decoding iterations compared to fully hard-decision based decoding algorithms such as BFA. The behavior of LDPC decoder is tested with Wireless Local Area Network (WLAN – IEEE 802.11n) standard. With significantly reduction in hardware resources, the implemented decoder can achieve high throughput.

2.17[41] Keshab K. Parhi, et. al. has described VLSI implementation-oriented (3,k) – regular low-density parity-check codes.

There are tremendous efforts are done to increase the error-correcting performance of Low-Density Parity-Check (LDPC) codes. However, very less effort is given to hardware implementation of practical LDPC decoder. If we directly apply fully parallel LDPC decoder architecture than hardware complexity is too high that is unsuitable for many applications.

Therefore, partly parallel decoder architecture is required for such applications. However, LDPC codes are mostly randomly constructed, so it is extremely difficult nearly impossible, to develop a direct transformation of fully parallel architecture to partly parallel architecture. Here, joint construction of LDPC code and decoder is proposed and this causes the reduction of hardware complexity and improving performance of LDPC systems. Joint (3,k) - regular LDPC code and decoder design causes construct a class of (3,k)-regular LDPC codes that not only have very good performance but also have high speed partly parallel decoder.

2.18[42] Zhongfeng Wang et. al. has described on finite precision implementation of low density parity check codes decoder

In this paper, the effects of finite precision i.e. quantization on LDPC decoding performance is analyzed. Various quantization schemes is investigated for Log-BP based LDPC codes decoder and the best choice considering the tradeoff between the hardware complexity and the performance were addressed in this paper. The most optimal word length proposed is 4 bits and 6 bits for representing received data and extrinsic information This paper shows that purposed quantization scheme for the Log-BP based decoder is effective in approximating the infinite precision implementation.

2.19[43] Zhiqiang Cui, et. al. has described Studies on Practical Low Complexity Decoding of Low-Density Parity-Check Codes

It is very necessary that LDPC decoder has low complexity i.e. low hardware requirement. In this paper, firstly LDPC decoder is implemented using Weighted Bit Flipping (WBF) based decoding algorithm with VLSI implementation. Then LDPC decoder is implemented using an optimized 2-bit soft decoding approach. Comparison between the two decoder hardware and decoding performance is done. This shown that purposed optimized 2-bit soft decoding approach has comparable hardware complexity with WBF-based algorithms while it has significantly better decoding performance. So, the proposed 2-bit soft decoding scheme is more suited for practical low complexity VLSI implementation of LDPC decoders.

2.20[44] In-Cheol Park et. al. has described Scheduling Algorithm for Partially Parallel Architecture of LDPC Decoder by Matrix Permutation

LDPC decoding by fully parallel architecture has high decoding throughput, but it also has large hardware complexity. The most practical solution for this drawback of fully-parallel architecture is to use the partially-parallel architecture. In partially-parallel architecture the variable node units and check node units are shared by several rows or columns that also cause reduction in complex interconnections. In this work, an efficient scheduling algorithm is given that can be applied to general LDPC codes. This algorithm minimize the overall processing time by overlapping the decoding operations. Scheduling algorithm causes to reduce the latency, because a check node operation can start before the variable node operation is finished, and vice versa. The overlapped processing of check node units and variable node units results in a reduced number of cycles for an iteration.

Chapter 3 SIMULATION AND SYNTHESIS TOOLS

3.1 Introduction

Tools required for implementation of (3, 6) LDPC decoder on FPGA are:

- XILINX ISE web pack 13.1 for design, synthesis and implementation.
- ISim simulator for simulation.

3.2 Simulation tools

ISim simulator is used to simulate the design written in VHDL language. ISim tool is used fast simulation of the design. ISim tool enables development and verification of the design completely. VHDL is a language is used for describing digital electronic system. Its full form is VHSIC Hardware Description Language. It arose out of the United States government's Very High Speed Integrated Circuit (VHSIC) program in 1980. It is used for the Gate level implementation of the circuits.

3.2.1 Structure of VHDL Code

- Every VHDL design description consists of at least one Entity/ Architecture pair. (In VHDL, this combination of an entity and its corresponding architecture is sometimes referred to as a design entity)
- In a large design, you will typically write many Entity/ Architecture pairs and connect them together to form a complete circuit
- An entity declaration describes the circuits as it appears from the "outside" i.e. from the perspective of its input and output interface.
- The second part of VHDL design description is the architecture declaration. Every entity in a VHDL design description must be bound with a corresponding architecture
- Architecture describes the actual function or contents of the entity to which it is bound.

3.2.2 Different Modeling Styles in VHDL

The internal working of entity can be defined using different modeling styles inside architecture body. These are defined as follow:

- *Behavioral Style*: This description of the circuit will describe the circuit in term of its operation over time. In this type of modeling, the internal working of an entity can be implemented using- Process statements, Sequential Statements etc.
- *Data flow Style*: This gives the view of data as flowing through a design, from input to output. In this abstraction you describe your circuit in term of how data moves through system. At heart of most digital systems are gates, so in this we can describe how information is passed between different gates.
- *Structural Style*: In this we can describe the function how it composed of sub-modules. Each of sub-modules is an instance of some entity, and the ports of the instances are connected using signals.

3.2.3 Advantage of using VHDL to design FPGAs

- *Top-Down Approach*: HDLs are used to create complex designs. The top-down approach to system design supported by HDLs is advantageous for large projects that require many designers working together.
- *Functional Simulation Early in the Design Flow*: One can verify the functionality of your design early in the design flow by simulating the HDL description.
- *Synthesis of HDL Code to Gates*: One can synthesize your hardware description to a design implemented with gates. This step decreases design time by eliminating the need to define every gate.
- *Early Testing of Various Design Implementations*: HDL allows one to test different implementations of your design early in the design flow. One can then use the synthesis tool to perform the logic synthesis and optimization into gates.
- *Reuse of RTL* Code: One can retarget RTL code to new FPGA architectures with a minimum of recoding.

3.3 Synthesis tools

Xilinx ISE design suite 13.1 is used for synthesis the implemented design.

3.3.1 Xilinx ISE design suite 13.1

For this dissertation, we use Xilinx ISE design suite 13.1. We can generate synthesizable design in VHDL and Verilog languages. The Xilinx ISE design suite has several elements inbuilt such as counters, adders, distributed ram, block ram etc. These elements are fast and have low hardware requirements i.e. operation of these elements are fast and number of slices utilizes is less. Xilinx ISE design suite 13.1 enables the user to know about the Slice utilized by the synthesizable design, maximum clock frequency supported by design, worst path delay in design etc. We can also analyze the RTL level design generated by the VHDL and Verilog languages design.

Xilinx ISE design suite gives us the ability to the design environment to our chosen PLDs as well as the preferred design flow. In general, the design flow for FPGAs and CPLDs is identical. We can choose whether to enter the design in schematic form or in HDL, such as VHDL, Verilog. The design can also comprise of a mixture of schematic diagrams and embedded HDL symbols. There is also a facility to create state machines.

Xilinx ISE design suite 13.1 incorporates a Xilinx version of the ISim simulator. This powerful simulator is capable of simulating functional VHDL before synthesis, or simulating after the implementation process for timing verification. Xilinx ISE software offers an easy to- use GUI to visually create a test pattern. A test bench is then generated and compiled to test and understand the behavior of the design. This XILINX release has been used for synthesis and implementation of our design. We can also download design or place and route the design on the desired FPGA.

3.4 Flow of design

This section examines the design flow for an FPGA or a CPLD. This is the entire process is for designing and implementation. The design flow consists of the steps

Step 1: Writing a Specification

As specification allows each engineer to understand the entire design and his or her piece of it. It allows the engineer to design the correct interface to the rest of the pieces of the chip. It also saves time and misunderstanding. There is no excuse for not having a specification.

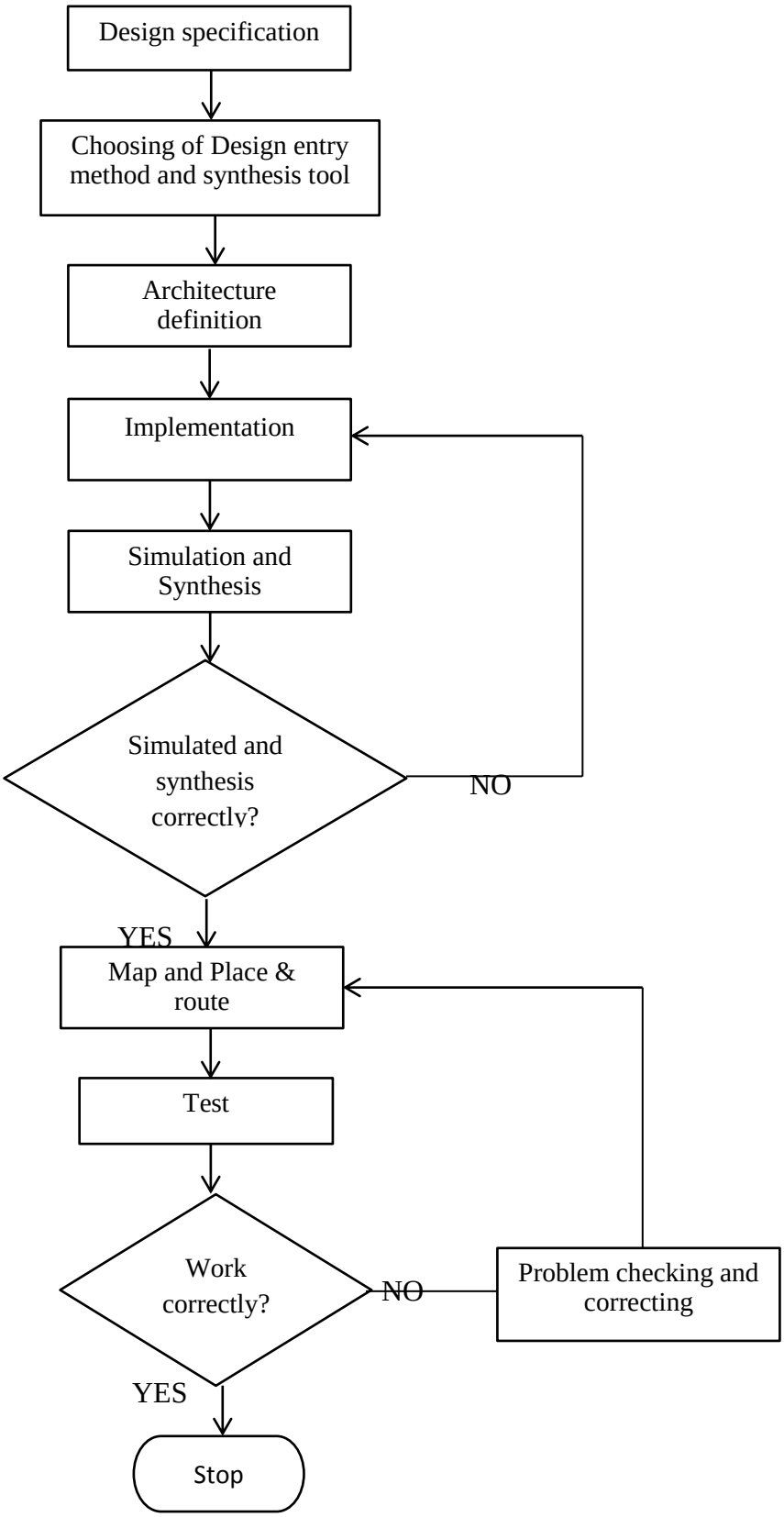

Figure-3.1: Design flow

A specification should include the following information:

- An external block diagram showing how the chip fits into the system.
- An internal block diagram showing each major functional section.
- A description of the I/O pins including
- Output drive capability
- Input threshold level
- Timing estimates including
- Setup and hold times for input pins
- Propagation times for output pins
- Clock cycle time
- Estimated gate count
- Target power consumption
- Target price
- Design throughput

Step 2: Choosing a Design Entry Method

You must decide at this point which design entry method you prefer. For smaller chips, schematic entry is often the method of choice, especially if the design engineer is already familiar with the tools. For larger designs, however, a hardware description language (HDL) such as Verilog or VHDL is used because of its portability, flexibility, and readability. When using a high level language, synthesis software will be required to synthesize the design. This means that the software creates low level gates from the high level description. VHDL used as the design entry method in this dissertation.

Step 3: Choosing a Synthesis Tool

You must decide at this point which synthesis software you will be using if you plan to design the FPGA with an HDL. This is important since each synthesis tool has recommended or mandatory methods of designing hardware so that it can correctly perform synthesis. It will be necessary to know the methods of design before so that sections of the chip will not need to be redesigned later on. In this dissertation, Xilinx ISE design suite 13.1 is used as synthesis tool.

Step 4: Defining architecture of designing

It is very important to follow good design practices.

- Top-down design
- Use logic that fits well with the architecture of the device you have chosen
- Use of Macros
- Synchronous design
- Routing of connections
- Computation unit definition
- Definition of element such as memory counters etc. (if any)

Step 5: Implementing of design

At this point, design is implemented in the design entry method use selected. This method can be use of VHDL language, Verilog log language and any other method. VHDL used for implementing the design in this dissertation.

Step 6: Simulating - design review

Simulation is an ongoing process while the design is being done. Small sections of the design should be simulated separately before hooking them up to larger sections. There will be much iteration of design and simulation in order to get the correct functionality. Once design and simulation are finished, another design review must take place so that the design can be checked. It is important to get others to look over the simulations and make sure that nothing was missed and that no improper assumption was made. In this dissertation, ISim is used for the simulation of the design.

Step 7: Synthesis

If the design was entered using an HDL, the next step is to synthesize the chip. Synthesis is the general term that describes the process of transformation of the model of a design in HDL, from one level of Behavioral abstraction to a lower, more detailed level. This involves optimally translate the register transfer level (RTL) design into a gate level design that can be mapped to logic blocks in the FPGA. This may involve specifying switches and optimization criteria in the

HDL code, or playing with parameters of the synthesis software in order to insure good timing and utilization.

With reference to VHDL, synthesis is an automatic method of converting a higher level of abstraction to a lower level of abstraction. The preparation of a synthesizable model requires the knowledge about features. It is important here to note that not all features of VHDL can be synthesized; therefore, one must consult Xilinx Simulation and Synthesis Guide for a list of synthesizable features

Step 8: Mapping

Map optimizes the gates and removes unused logic. This step also maps the designs logic resources.

Step 9: Place and root

The next step is to lay out the chip, resulting in a real physical design for a real chip. This involves using the vendor‘s software tools to optimize the programming of the chip to implement the design. Then the design is programmed into the chip. The Place and Route process places each macro from the synthesis list into an available on the FPGA and connects the macros using routing resources available on the FPGA. The job of the place and route tool is to create the programming files that will be used to specify the logic function of the logic macros in the logic areas and the switch programming of the wires used to connect the macros together. The placement algorithm also tries to place logical gates on the critical path close to each other so that local interconnect can used to connect the gates.

Step 10: Resimulating - final review and Testing

The design must be resimulated with the new timing numbers produced by the actual layout. If everything has gone well up to this point, the new simulation results will agree with the predicted results. If the problems encountered here are significant than sections of the FPGA may need to be redesigned. If there are simply some marginal timing paths or the design is slightly larger than the FPGA, it may be necessary to perform another synthesis with better constraints or simply another place and route with better constraints. If you have followed the procedure up to this point, chances are very good that your system will perform correctly with only minor problems.

System testing is necessary at this point to insure that all parts of the system work correctly together.

Step 11: Generating of programing file

This feature generates the bit file to be downloaded on to the target device i.e. FPGA using the downloading cable.

Chapter 4 IMPLEMENTATION OF (3,6) LDPC DECODER

4.1 Introduction

LDPC (Low Density Parity Check) codes have excellent performance in the noisy channel. The trade-off between the hardware complexity, decoding throughput and performance is a critical factor in the implementation of the practical decoder. Large number of LDPC algorithms is introduced till now with varying complexity and performance. However, trade-off between decoder complexity and decoder performance (like throughput, number of iterations, BER) is still a problem. Iterative log Belief-propagation (BP) algorithm, based on soft-decision decoding, has high decoding capability but high decoding complexity. In contract, Bit-Flip algorithm, based on hard-decision decoding, has low decoding complexity but poor performance. Decoder architecture is another important factor that determines the complexity and throughput of the decoder. Fully parallel architecture has high throughput but has very high decoder complexity. As the code length increases complexity also increases. So complexity limits the maximum code length to be decoded. Serial decoder architecture has low decoder complexity but also has low throughput. Its throughput decreases with increase in the code length. So, partially parallel decoder was introduced to trade-off between throughput and complexity.

4.2 Construction of parity check matrix

In this section, description of construction of the (3,6) LDPC parity check matrix is given with girth of 12 [45]. The construction of parity matrix is divided in two parts:

4.2.1 Regular and fixed matrix: As shown in figure-4.1 matrix H1 and H2 is regular and fixed matrix. Sizes of these matrixes are defined as $L.k^2 \times L.k$ matrix. Here, L defined the size of sub matrix and k is degree of check node i.e. (3,k) LDPC codes. $I_{x,y}$ block in H1 matrix is $L \times L$ identity matrix. $P_{x,y}$ block of H2 matrix is obtained by right cyclic shift of an $L \times L$ identity matrix. Let we have $T^u(I)$ right cyclic shift factor. This factor represent right cyclic shift of matrix 'I' by 'u' columns. Then $P_{x,y} = T^u(I)$ where $u = ((x - 1). y) \bmod L$. Let L = 5, x = 3 and value of y =4, then we have value of $u = ((3 -1). 4) \bmod 5 = 8 \bmod 5$ which is equal to 3, then

$$P_{3,4} = T^3(I) = \begin{vmatrix} 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \end{vmatrix}$$

Both H1 and H2 contains 'k' ones in each row and each column contain single one, that make (2,k) parity check matrix with $L.k^2$ variable nodes and $2L.k$ check nodes.

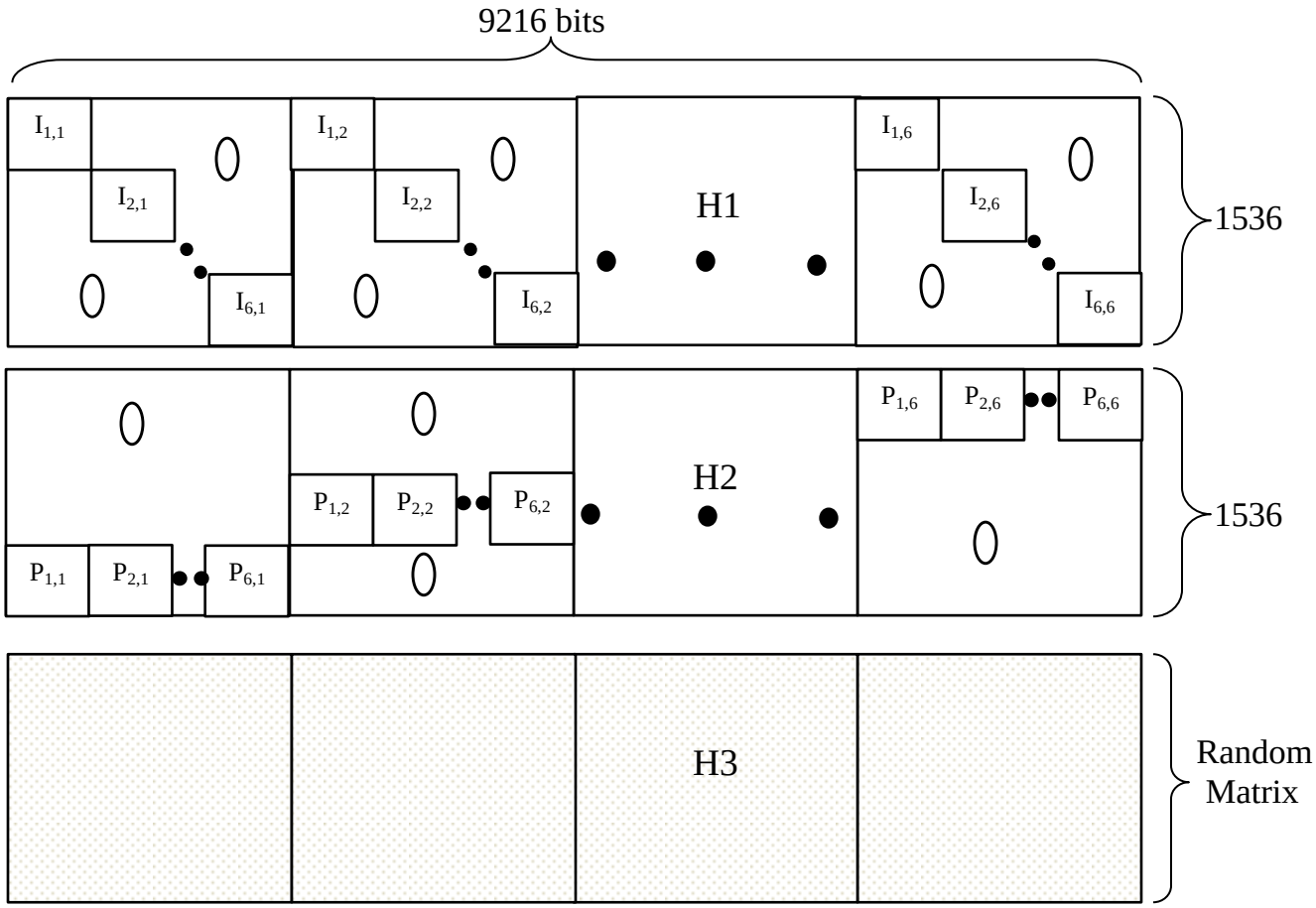

Figure-4.1: (3,6) parity check matrix

4.2.2 Random matrix: Matrix H3 is random matrix as shown in figure-4.1. The size of matrix is defined as $L.k^2 \times L.k$. In this positions of 1's are randomly chosen and the error positions change with iterations in a seemingly random fashion. In H_3 matrix there should be 'k' ones in each row and single one in each column. The construction of H_3 sub-matrix is done jointly with the decoder architecture.

Now, the overall matrix is the combination of both these sub-matrixes. This leads to the (3,k) parity matrix. So, total matrix is $\mathbf{H = [H1^T, H2^T, H3^T]^T}$. This overall matrix has $L.k^2$ variable nodes and 3L.k check nodes in tanner graph.

4.3 Reduced complexity decoding algorithm

Before description of algorithm, let we have **H** parity check matrix for LDPC codes. It has '**V**' variable nodes and '**C**' check nodes. Here, $H_{i,j}$ denotes the entry (i,j) at the **H** matrix. Let we have 'n' set of variable node which participate in m_{th} check node as $C(m) = \{n : H_{m,n}=1\}$, and 'm' set of check node which participate in n_{th} variable node as $V(n) = \{m : H_{m,n}=1\}$. C(m) \ n denote set of C(m) bits excluding 'n_{th}' and V(n) \ m denote set of V(n) bits excluding 'm_{th}'.
Here, reduced complexity decoding algorithm is used for trade-off between low decoding complexity and decoder performance.

- Let $P^1_n = Pr[x_n = 1]$ be the conditional probability that x_n is '1' and $P^0_n = Pr[x_n = 1]$ be the conditional probability that x_n is '0'
- Here, x_n are the n_{th} input bit and 'n' is number of variable nodes.

Initialization: At first, we compute Likelihood ratio (L_n) for each input variable node and assign this to one bit variable-to-check message ($VC_{m,n}$). In this, LLR value will update after the each iteration that is why initial LLR value is used only once.

$$\text{Likelihood ratio } (L_n) = \ln\left(p_n^0 / p_n^1\right)$$

$$VC_{m,n} = MSB(L_n)$$

Where, 'n' denote n_{th} variable node

$MSB(L_n)$ denotes most significant bit of L_n i.e. sign bit

Check – node operation: The complexity of the check node depends on the quantization of the variable-to-check and check-to-variable message. That is why check node complexity increase with increase in the quantization bit. However, in reduced complexity algorithm complexity is very much decreased by using only one bit variable-to-check and check-to-variable message. In this, only XOR gates are used for check node implementation, similar to BFA.

- In this one bit check-to-variable information ($CV_{m,n}$) for each check node is computed as:

$$CV_{m,n} = VC_{m,1} \text{ xor } VC_{m,2} \text{ xor } \dots\dots\dots \text{xor } VC_{m,k}, \qquad \text{where}, k \in C(m) \setminus n$$

Variable – node operation: It perform the following operations

1. compute new updated LLR value
2. compute variable-to-check messages ($VC_{m,n}$)

- Variable node also performs the soft-operation on the hard-decision bits received from the check nodes. Variable node converts the received check node bits into constant integer value called weight 'W'. Value of weight is positive if received check bit is '0' and weight is negative if received check bit is '1'.
- In this one bit variable-to-check message ($VC_{m,n}$) is computed as:

$$Y_{m,n} = +W \quad \text{if} \quad CV_{m,n} = 0$$

$$Y_{m,n} = -W \quad \text{if} \quad CV_{m,n} = 1$$

Where, $Y_{m,n}$ is temporary integer check-to-variable message, according to original hard-decision check-to-variable message bit.

$$g_{m,n} = L_n + \sum_{k \epsilon V(n) \setminus m} Y_{k,n}$$

$$VC_{m,n} = MSB(g_{m,n})$$

Where, $g_{m,n}$ is temporary integer variable-to-check message.

- Log-likelihood ratio (LLR) is update to

$$L_n = L_n + \sum_{k \epsilon V(n)} Y_{k,n}$$

- Output decision is '0' if $L_n \geq 0$ and '1' if $L_n < 0$.

4.4 Proposed partial-parallel decoder architecture

In this partial parallel architecture of 9216-bit at code rate of ½ is developed. It should be required that, architecture should have high-speed and low complexity. This architecture is for (3,k) parity check matrix. Here, value of k = 6 i.e. this partially-parallel decoder architecture is for (3,6) LDPC codes.

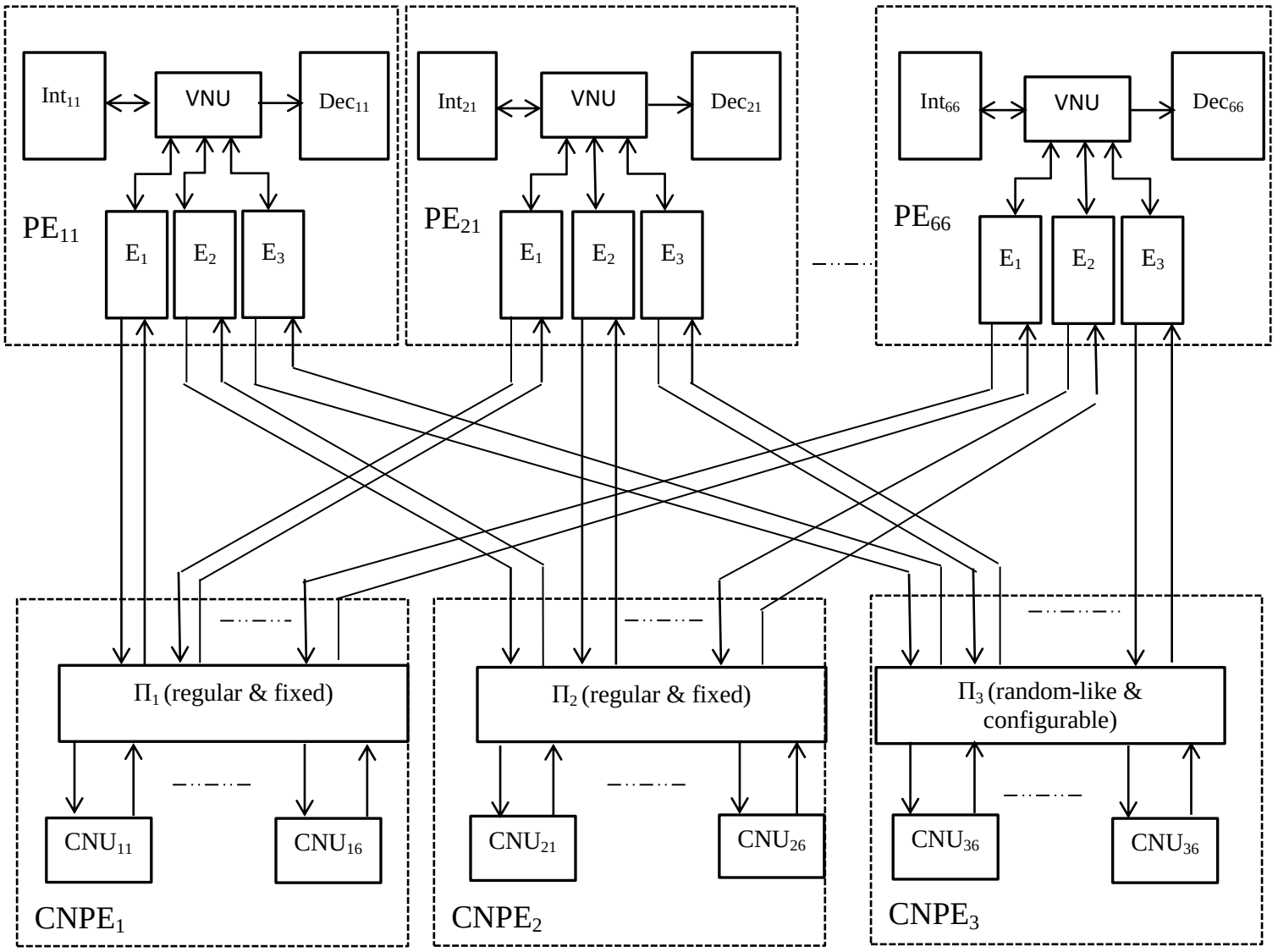

Figure-4.2: Principal partial-parallel decoder architecture

This architecture contains: k^2 i.e. '36' variable node and 3k i.e. '18' check nodes. The architecture has 2-D network for implementation of random-like shuffle network with low routing complexity and the folding factor of VNU and CNU is 'L' i.e. each VNU done the operation of 'L' variable nodes and CNU done the operation of 'L' check nodes. Decoder computes each iteration in only 2(L + 2) clock cycles. In which CNU woks for first (L + 2) cycles and VNU woks for second (L + 2) cycles. Here, extra '2' cycles are due to pipelining stages and the value of L is '256'.

Figure-4.2 shows principal partial-parallel decoder architecture. It consist of k^2 i.e. '36' PE (processing elements) and '3' CNPE (check node processing elements). Each CNPE consist k i.e. '6' check node and '1' shuffle network. That is why, decoder contain '3' bi-directional

shuffle network, each of these represent H1, H2 and H3 matrix. Each decoder PE block consists of three rams i.e. E1, E2 and E3. Each of these rams stores decoding information with respect to its 'L' i.e. '256' variable node in the variable node group. This, variable node group is sub-matrix of 'H' consisting of 'L' columns, in that sub-matrix of 'H' we go through '$I_{x,y}$' and '$P_{x,y}$'. Exchange of information between variable node and check node is through the bi-directional shuffle network. During variable-to-check message, shuffling is done and during check-to-variable message unshuffling is done.

This decoder has two modes in the each decoding iteration. During 1_{st} (L + 2) cycles it works in CNP (check node processing) mode and during 2_{nd} (L + 2) cycles it works in VNP (variable node processing) mode. In the CNP mode: computing of check-to-variable messages and information exchange between variable node is done. In the VNP mode: decoder only computes the variable-to-check messages. In this check-to-variable messages and variable-to-check messages are of '1' bit only. In this each of the E1, E2 and E3 ram called extrinsic ram is of (256 × 1) bits, each of Int_{xy} ram called intrinsic ram is of (256 × 8) bits and each of Dec_{xy} ram called decision ram is of (256 × 1) bit. In the Intrinsic ram, same address is used for storing variable-to-check message and check-to-variable message given for $\{H_{i,j} = 1\}$ where, $H_{i,j}$ denote the entry (i,j) at the **H** matrix and Decision ram is used for storing hard-decisions. This technique of architecture is greatly simplifies the generation of the control logic.

4.5 Check-node processing (CNP)

In the check node processing, computing of check-to-variable messages and information exchange between variable nodes is done. To achieve higher throughput, pipelining is used in CNP. To realize pipelining scheme, bi-directional connection in three shuffle networks is done by two distinct opposite direction unidirectional wires. In decoder, CNP perform following functions: 1. *Read*: In this, decoder reads a 1-bit variable-to-check message from the rams E1, E2 and E3 (Extrinsic rams); 2. *Shuffle*: It shuffles the input message and sent it to the respective CNU. This shuffle network can be fixed or random; 3. *Modify*: In this parity check is perform on input variable-to-check messages and convert it into check-to-variable messages; 4. *Unshuffle*: This works opposite to the shuffle network. In this check-to-variable messages are sent to their respective variable nodes.;5. *Write*: In this check-to-variable message are written to same address on E1, E2 and E3 rams from where read operation is done before. Due to this, write address for

each check-to-variable message, is obtained via delaying the read address by number of pipelining stages.

In this we use three stages of pipelining, because as we see, the architecture of CNU is not very complex so we can realize shuffle, CNU and Unshuffle in one clock cycle. Figure-4.4 shows three stage pipelining of CNP, in first cycle we have read operation, in second cycle we have shuffle, CNU and Unshuffle operation, in third cycle we have write operation.

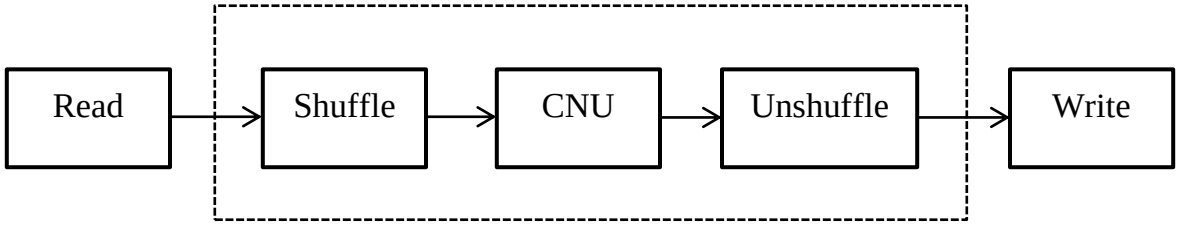

Figure-4.4: Three-stage pipelining of CNP

4.5.1 Implementation of Check node unit (CNU)

Each CNU perform operation of one check node i.e. computation of check-to-variable messages. Figure-4.5 shows the architecture of CNU, which is less than architecture of Log-BP algorithm architecture. It consists of only '12' xor gates. Input to CNU is k i.e. 6-bit variable-to-check message coming from '6' PE (processing elements) and the output is also of k i.e. 6-bit sending to various PE. This architecture realizes the "check node computation" of the reduced complexity algorithm completely.

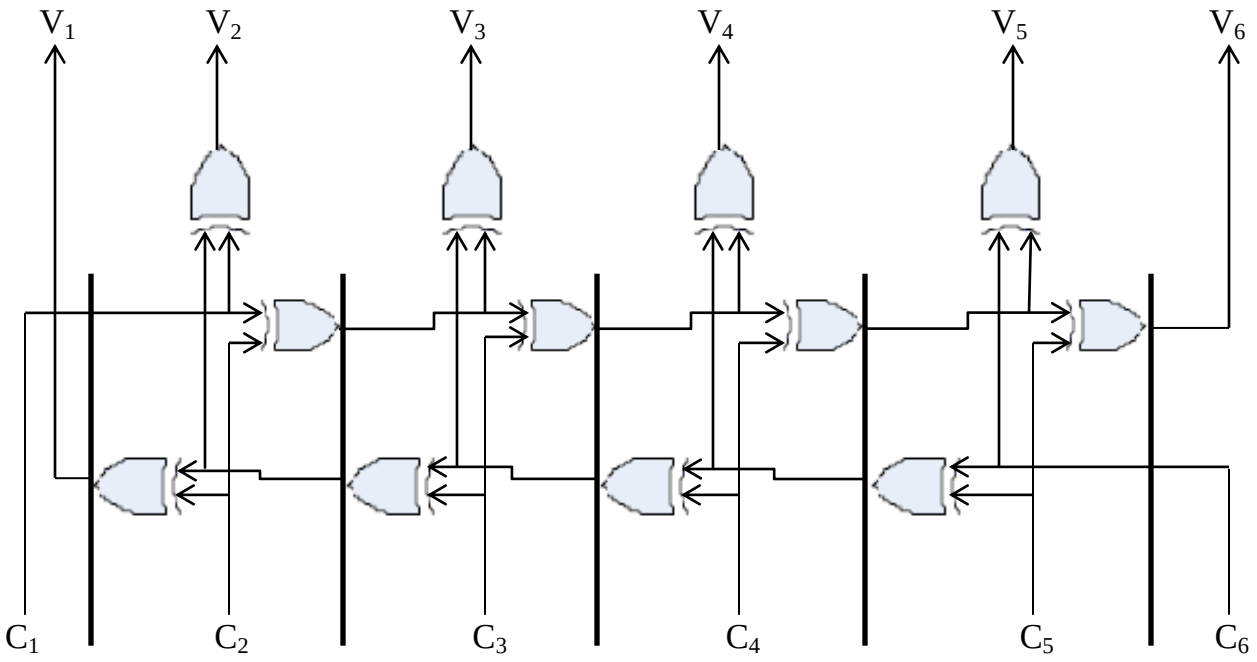

Figure-4.5: Architecture of check node unit (CNU)

4.5.2 Implementation of Address Generator (AG)

There are '3' AG is used in each PE block. Each address generator is corresponding to each of these rams E1, E2 and E3 (Extrinsic rams) as shown in figure-4.3 i.e. AG^{i}_{xy} associates with $ramE_i$. Address generator provides the read address for variable-to-check messages and write address for check-to-variable messages, which is obtained via delaying the read address by number of pipelining stage. Three address generators are:

Address generator-1(AG^{1}_{xy}): Each AG^{1}_{xy} is a $[\log_2 L]$ - bit binary counter i.e. $\log_2 256 = 8$ bits binary counter. It counts from '0' to [L - 1] i.e. (0-to-255). Its initialization at beginning of:

- CNP mode is $\text{Intialize}_{xy} = 0$
- VNP mode is $\text{Intialize}_{xy} = 0$

Address generator-2(AG^{2}_{xy}): Each AG^{2}_{xy} is a $[\log_2 L]$ - bit binary counter i.e. $\log_2 256 = 8$ bits binary counter. It counts from '0' to [L - 1] i.e. (0-to-255). Its initialization at beginning of:

- CNP mode is $\text{Intialize}_{xy} = ((x-1).\ y) \bmod L$
- VNP mode is $\text{Intialize}_{xy} = 0$

Address generator-3(AG^{3}_{xy}): This address generator is for realization for random matrix that is why, its initialization is randomly generated. Each AG^{3}_{xy} is a $[\log_2 L]$ - bit binary counter i.e. $\log_2 256 = 8$ bits binary counter. It counts from '0' to [L - 1] i.e. (0-to-255). Its initialization at beginning of:

- CNP mode is $\text{Intialize}_{xy} = t_{xy}$
- VNP mode is $\text{Intialize}_{xy} = 0$

For the generation of '4-cycle free' tanner graph it should be required that following conditions for t_{xy} must be satisfied.

- For given x, we have $t_{x,y1} \neq t_{x,y2}$, where $y_1, y_2 \in \{1, \dots, k\}$
- For given y, we have $t_{x1,y} - t_{x2,y} \not\equiv ((x_1 - x_2).y) \bmod L$,where $x_1, x_2 \in \{1, \dots, k\}$

4.5.3 Implementation of shuffle network

Shuffle network is used for the connection between variable node and check node. It defines this connectivity jointly with address generator. There are three shuffle network used in this decoder. Each sub matrix H1, H2 and H3 has shuffle network Π_1, Π_2 and Π_3 respectively .Three shuffle networks are:

Shuffle network for H1 i.e. Π_1: This shuffle networks connect PE_{xy} to the $CNPE_1$. This connects k i.e. '6' PE elements with the CNU which has same x-index.

Shuffle network for H2 i.e. Π_2: This shuffle networks connect PE_{xy} to the $CNPE_2$. This connects k i.e. '6' PE elements with the CNU which has same y-index.

Shuffle network for H3 i.e. Π_3: This shuffle networks connect PE_{xy} to the $CNPE_3$. It should be required that the matrix H3 should be random to some content. So, implementation of Π_3 should be of random type. Here, we use 2-D random shuffle network implementation technique for realization of the random network.

Figure-4.6 shows the shuffle network forward path. In this, input is get from $PE_{x,y}$ and output is sent to $CNU_{3,k}$ for parity check. It realizes two types of shuffle stage i.e. intra-row shuffle and intra-column shuffle. At start, $r_{x,y}$ data block coming from the $PE_{x,y,}$ sent to intra-row shuffle. In this, each row shuffle network 'α_k' shuffles the k i.e. '6' input data $r_{x,y}$ to $C_{x,y}$ for $1 \leq y < k$ and same value of 'x'. Each shuffle 'α_k' is controlled by one bit signal 'Sr_k'. If $Sr_k = 1$ then network perform shuffling otherwise identity permutation. The k i.e. '6' bit control signal 'Sr' changed every clock pulse and its value for each L is stored in ROM1 so this is fixed random permutation. Due to the size of ROM1 is (256 × 6). In next shuffle stage data block $C_{x,y}$ suffer from intra-column shuffle. In this, each shuffle network 'β_k' are controlled by one bit signal 'Sc_k'. If $Sc_k = 1$ then network perform shuffling otherwise identity permutation. This, each column shuffle network 'β_k' shuffles the k i.e. '6' input data $C_{x,y}$ to $V_{x,y}$ for given $1 \leq x < k$ and same value of 'y'. The k i.e. '6' bit control signal 'Sc' changed every clock pulse and its value for each L is stored in ROM2. Due to the size of ROM2 is (256 × 6). Now for implementing bi-directional shuffle network, we also require backward path of shuffle network from $CNU_{3,k}$ to $PE_{x,y}$. For this we should configure shuffle network 'α_k' and 'β_k' as backward network.

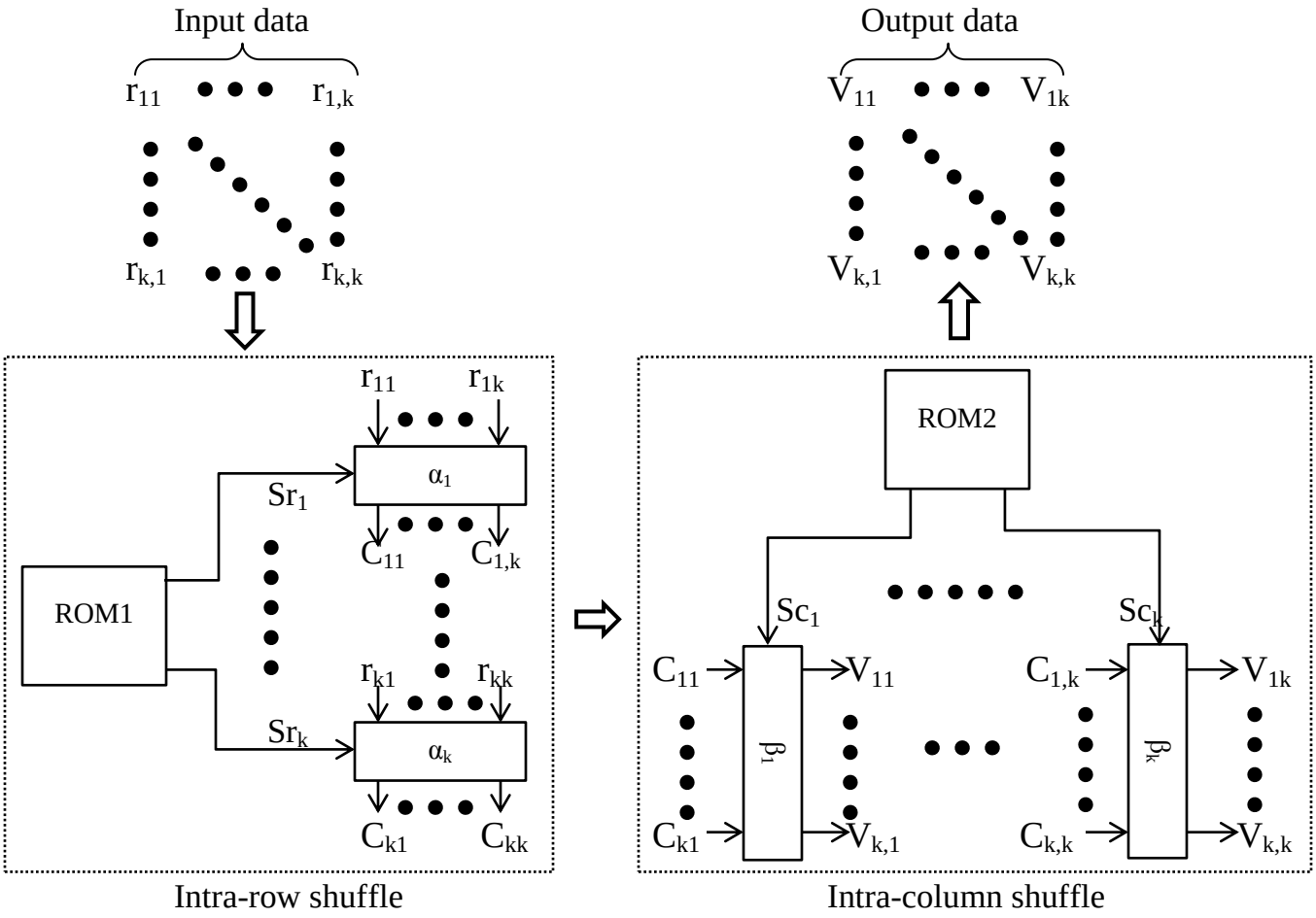

Figure-4.6: shuffle network for Π_3

4.6 Variable node processing (VNP)

In the variable node processing, computing of variable-to-check messages is only performed. In this we use three stages of pipelining. In VNP decoder perform following functions in each clock cycle: 1. *Read*: In this decoder reads '1'-bit check -to- variable messages from the rams E1, E2 and E3 (Extrinsic rams); 2. *Modify*: In this, based upon input check -to- variable messages and intrinsic message, each VNU compute 1-bit hard decision, three 1-bit variable-to-check messages and update value of LLR;3. *Write*: In this variable-to-check message are written to same address on E1, E2 and E3 rams from where read operation is done before, also hard-decision is stored in Decision ram and updated LLR message is stored in intrinsic ram. Due to this, write address for variable-to-check message, is obtained via delaying the read address by number of pipelining stages.

Figure-4.7 shows three stage pipelining of VNP, in first cycle we have read operation, in second cycle we have modify operation done VNU, in third cycle we have write operation.

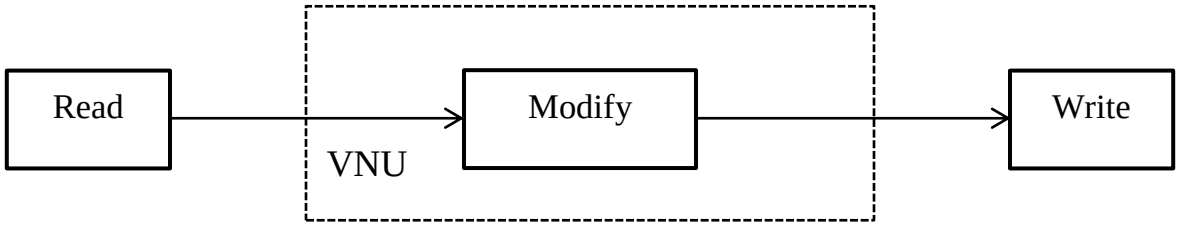

Figure-4.7: Three-stage pipelining of VNP

Variable node unit (VNU): Each VNU generates: hard-decision and stored it in 'Dec' ram (Decision ram), intrinsic message and stored it in 'Int' ram (Intrinsic ram), variable-to-check messages and stored it in E1, E2 and E3 ram (Extrinsic ram). In the input, it has input '5-bit' intrinsic message during initialization and '8-bit' intrinsic message during decoding iteration. Here, intrinsic message is nothing but 'LLR' value. It also has three '1-bit' check-to-variable messages as input. Its output is one '8-bit' intrinsic message, three '1-bit' variable-to- check messages and one '1-bit' hard-decision .

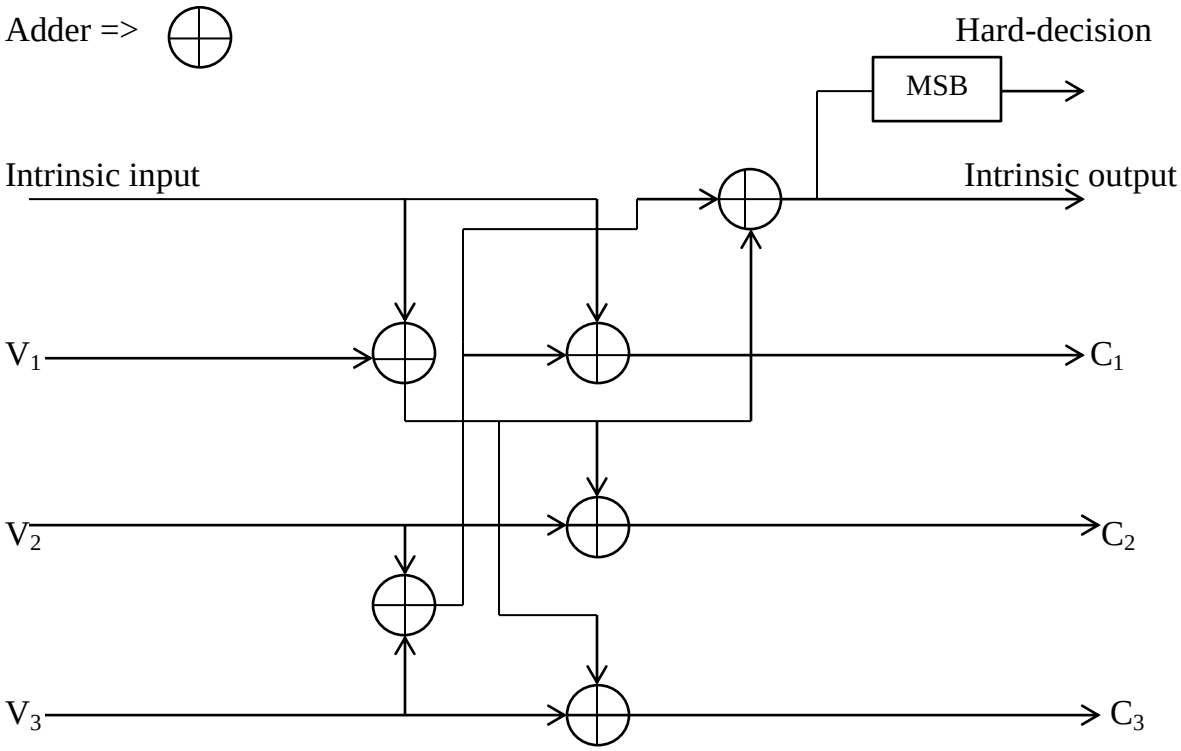

Figure-4.8: Architecture of VNU

Figure-4.8 shows the architecture of the VNU. In this, intrinsic message input are get from 'Int' ram and three '1-bit' check-to-variable messages are get from Extrinsic rams i.e. E1, E2 and E3. The size of intrinsic input is 8-bits and check-to-variable message of '1' bit is converted into 8-bit message by constant weight of 8-bit. Intrinsic message output is stored in the 'Int' ram, three '1-bit' variable-to-check messages are stored in the rams E1, E2 and E3 and hard-decision is stored in 'Dec' ram. The size of VNU output is 8-bit, variable-to-check messages is converted in '1' bit messages by taking MSB of 8-bit messages. This architecture completely realizes the variable node of reduced complexity algorithm. All the input and output message is in the two's complement format. In this architecture, there are six '8-bit' binary adders are used and MSB of intrinsic output i.e. updated LLR value is used as hard-decision bit.

4.7 Pipelining of the consecutive code frames

Partially-parallel decoder works simultaneously on three consecutive code frames. In this decoder, when one frame is read out from the decoder, then next frame is iteratively decoded and third frame is loaded into the decoder. So, decoder works on three frames simultaneously. To realize this pipelining, two 'Int' rams and two 'Dec' rams are used. The size of first 'int' ram is (256 × 5) ram because we use 5-bit quantization of LLR i.e. initial value of intrinsic message and the second 'Int' ram is of (256 × 8) because we update the value of LLR to 8-bit quantization for further processing of iteratively decoding. Input frame is loaded into the first 'Int' ram and second 'Int' ram is used for storing the intrinsic message during iteratively decoding. We also used two 'Dec' rams, Output frame is read out from the first 'Dec' ram of decoder and second 'Dec' ram is used for storing the hard-decision during iteratively decoding. Data of one ram is transferred to another ram during initialization phase for further processing.

Chapter 5 DATA INPUT/OUTPUT, BEHAVORIAL SIMULATION FPGA IMPLEMENTATION AND RESULTS

5.1 Structure of LDPC decoder for data input/output

Figure-5.1 shows the Simple block diagram of LDPC decoder. It consists of global 'clock' for synchronous operation of decoder with other circuits. The architecture of LDPC decoder is "positive edge triggered of clock". In this architecture, we use 5-bit quantization for LLR (intrinsic) input and LLR input is loaded 'one' symbol per positive edge of clock pulse. The LLRs are fed into the decoder using the 'Load' control signal i.e. LLR input is also synchronous with 'Load' control signal. When 'load' signal is high then input is serially loaded in decoder in next positive edge of clock. After decoding of input, 1-bit output is obtained at 'decoded data' with synchronous with 'dataout_ready' and 'clock'. Output is available only in next positive edge of clock when 'dataout_ready' is high. In this, architecture the number of input and output pins is decrease than architecture in [25]. In this, output is obtained by only '1' pin rather than '36' pin in architecture [25]. This architecture does not need '14' pins for loading the address in the each 'Int' memory of '36' variable nodes.

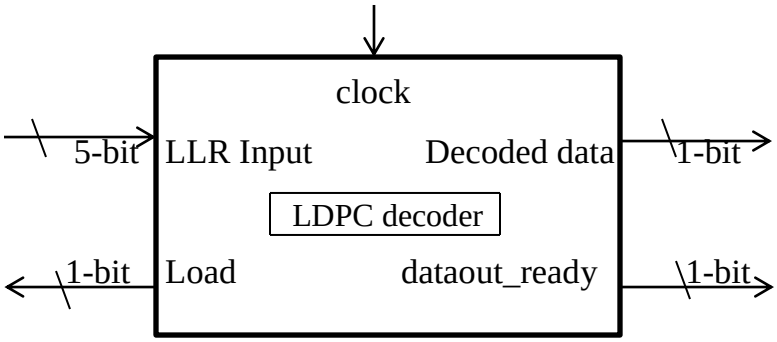

Figure-5.1: Simple block diagram of LDPC decoder

5.2 Behavior simulation with ISim simulator

In this section, we verify the result of Behavior simulation in ISim simulator and simulate the decoder input passes through three stage pipelined structure. Here, we use clock signal 'clk' for

the synchronous operation and decoder works on positive edge of the clock. First of all, we input the 5-bit LLR information through ‘din’ port. We apply this LLR input on the next positive edge when ‘load’ signal is high .As shown in figure-5.2, ‘l1’ signal shows the memory address of ‘Int’ ram, in which input LLR information is loaded and ‘c’ signal shows the VNU in which LLR information is loaded out of ‘36’ VNU i.e. form (0-to-35). It takes 9,474 total clock cycles to input intrinsic information i.e. 258 for initialization and 9216 for input.

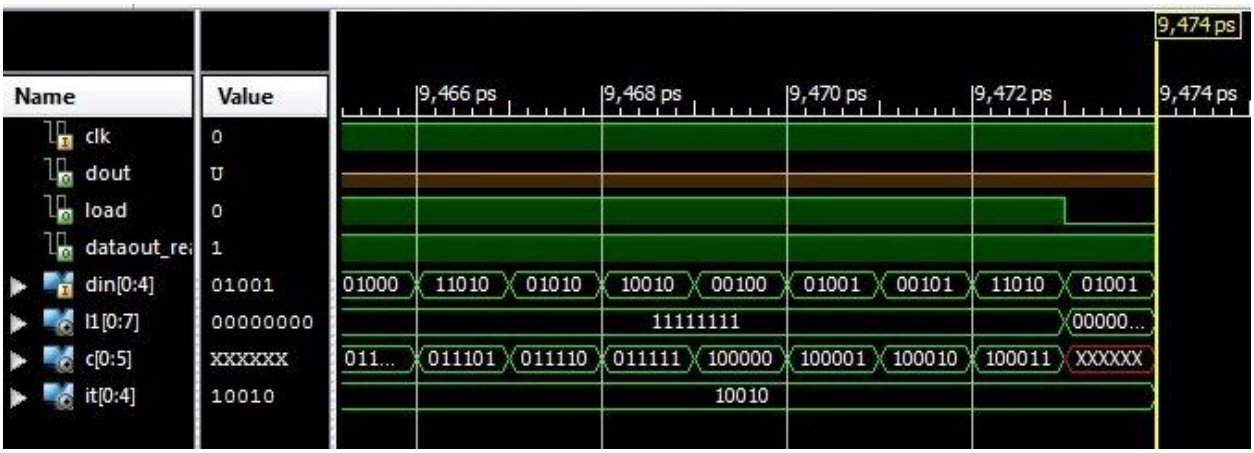

Figure-5.2: Decoder has loaded all the LLR information

5.3 FPGA Implementation

We apply partially parallel architecture for (3,6)-regular LDPC code and L = 256 on Xilinx XC3D3400A device from Spartan-3A DSP family.

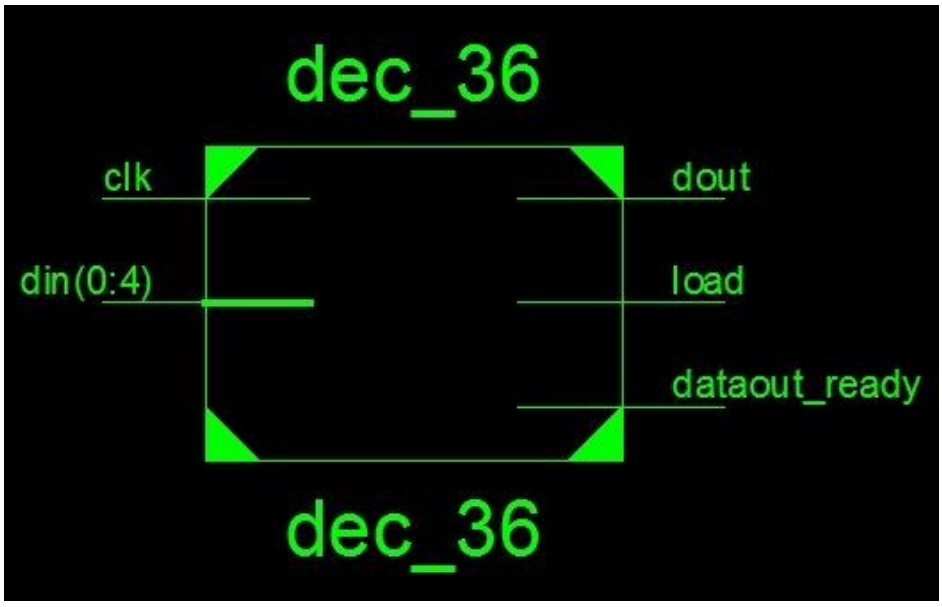

Figure-5.5: Pin configurations

Figure-5.5 shows the pin configuration of the (3,6)-regular LDPC decoder. In the left hand side of the block there are input pins and right hand side there are output pins. In this implementation of decoder, we configure:

Placed and Route of the decoder using Xilinx FPGA editor tool is shown in figure-5.9. In this figure-5.9 shows the clocked region and design implementation

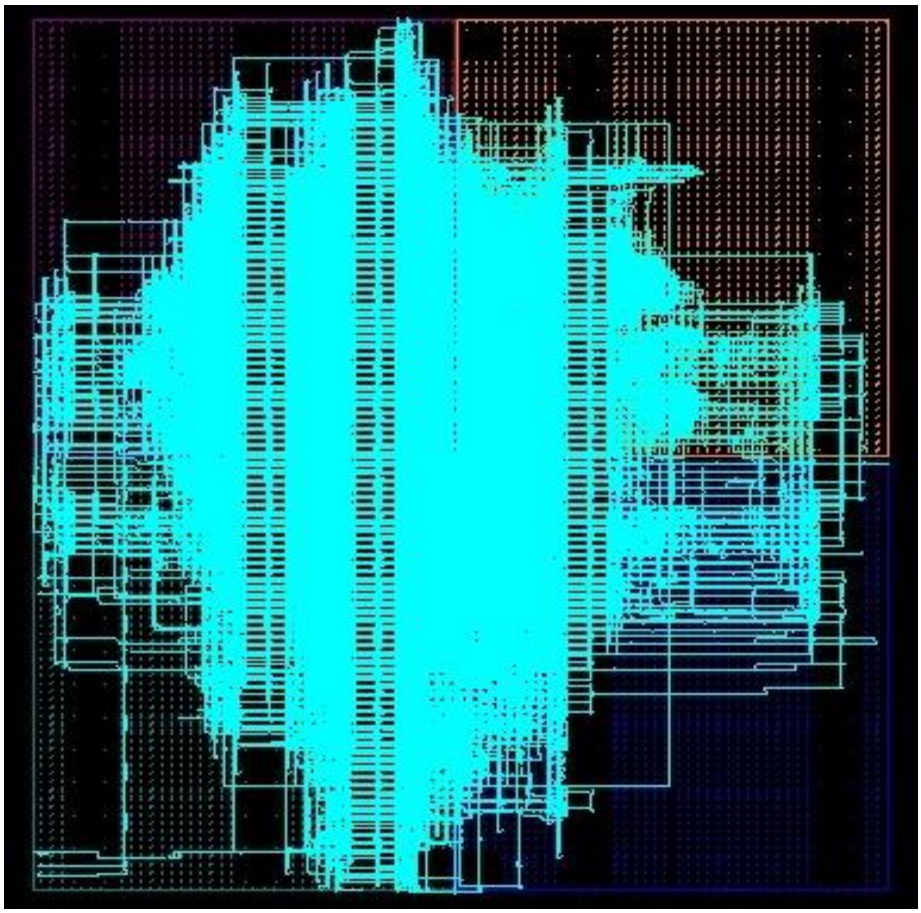

Figure-5.9: Placed and Route decoder implementation

Chapter 6 CONCLUSION AND FUTURE WORK

6.1 Conclusion

The trade-off between throughput and hardware implementation of LDPC codes is critical factor for the design of LDPC decoder architecture. The objective was to implement in hardware of (3,6) LDPC code having high decoding throughput and less area consumption. We achieved our objectives by developing LDPC decoder using reduced complexity algorithm. We implemented the 9216-bits, rate-1/2 regular LDPC matrix on the Xilinx XC3D3400A FPGA device. This decoder has maximum decoding iteration is '18', maximum decoding throughput is '92.95' Mbps and Maximum slice utilization is '7021'. Reduced complexity algorithm used in this dissertation greatly reduced the resources utilization by BP-algorithm. Reduced complexity algorithm has performance is in between Soft SPA algorithm and Hard BFA algorithm. In this algorithm, the variable node operation is soft operation which improves the performance and check node operation is hard operation which improves the complexity. The quantization of variable-to-check message and check-to-variable message is reduced to one bit only which reduce the complexity of decoder hardware. The pipelining system used in the (3,6) LDPC decoder architecture cause in increase in decoding throughput. In this (3,6) LDPC decoder architecture, hardware resource utilization result obtained by the partially-parallel decoder is very less than fully-parallel decoder architecture. Partially-parallel decoder has variable node unit very less than columns in the parity check matrix and check node unit is very less than rows in the parity check matrix. Here, we use only '36'variable node instead of '9216' columns in LDPC matrix and '18' check nodes instead of '4608' rows in matrix.

6.2 Future work

Following are the number of areas for future research:

- Balanced trade-off between decoding throughput, hardware complexity and performance of the LDPC decoder.

- Increase in the girth of code without significantly increasing the computational complexity of the algorithm because large girth causes the improvement in code performance.
- Extending the work from [25] and explore more method to realizes the architecture satisfying characteristic such as high girth, low complexity, high decoding throughput etc.
- Hardware implementation of capacity efficient LDPC codes. So that maximum utilization of the capacity of the channel.
- Achieving the high energy efficient LDPC decoder.
- Further analysis of code in term of the minimum distance required.

Paper under Review

Paper “FPGA implementation of (3,6) LDPC decoder using reduced complexity algorithm” with Manuscript Number COMPELECENG-D-13-00431 in http://ees.elsevier.com/compeleceng/

REFERENCES

[1] R.G. Gallager, “Low-density parity-check codes”, IEEE Transactions on Information Theory, 8:21-28, 1962.

[2] D. J. C. MacKay and R. M. Neal, “Near Shannon limit performance of low density parity check codes,” Electronics Letters, vol. 32, no. 18, pp. 1645-1646, Aug. 1996.

[3] C. E. Shannon, “A mathematical theory of communication”, Bell System Technical Journal, vol. vol. 27, pp. 379.423 and 623.656, July and October 1948.

[4] K. K. Parhi, “VLSI Digital Signal Processing Systems: Design and Implementation”, JohnWiley & Sons, 1999.

[5] B. M. J. Leiner, “LDPC Codes – a brief Tutorial,” April 2005 available at http://bernh.net/media/download/papers/ldpc.pdf

[6] R. M. Tanner, “A recursive approach to low complexity codes,” IEEE Transaction on Information Theory, pp. 533-547, September 1981.

[7] Sarah J. Johnson, “Introducing Low-Density Parity-Check Codes” available at http://materias.fi.uba.ar/6624/index_files/outline_archivos/SJohnsonLDPCintro.pdf

[8] Tuan Ta, “A Tutorial on Low Density Parity-Check Codes” at www.ece.umd.edu/~tta/

[9] M. Bossert, “Channel Coding for Telecommunications”, John Wiley & Sons, 1999

[10] C. Berrou, A. Glavieux, and P. Thitimajshima, “Near Shannon limit error-correcting coding and decoding: Turbo-codes”, IEEE International Conference on Telecommunications, Vol. 2, Issue 23-26, pp. 1064-1070, May 1993.

[11] D.J. Costello, J. Hagenauer, H. Imai, and S.B. Wicker, “Applications of error control and coding”, IEEE Transactions on Information Theory, 44(6):2531-2560, 1998.

[12] Jet Propulsion Laboratory, “Basics of Space Flight, 2004”, http://www2. jpl.nasa.gov/basics.

[13] H. Hemmati, “Overview of laser communication research”, In Proceedings of SPIE (The Search for Extraterrestrial Intelligence) in the Optical Spectrum III, volume 4273, 2001.

[14] V. Bhagavatula, H. Song and J. Liu, “Low-density parity-check (LDPC) codes for optical data storage”, In Proceedings of IEEE International Symposium on Optical Memory and Optical Data Storage Topical Meeting, pages 371-373, 2002.

[15] R. Blahut,“Theory and Practice of Error Control Codes”, Addison-Weseley Publishing Co in 1984.

[16] S. Chung, G.D. Forney, J.J. Richardson, and R. Urbanke, “On the Design of Low-Density Parity-Check Codes within 0.0045dB of the Shannon Limit”, IEEE Communication Letters, 5:58–60, February 2001.

[17] D. MacKay and M.C. Davey, “Evaluation of Gallager Codes for Short Block Length and High Rate Applications”, Proceedings of the IMA Workshop on Codes, Systems and Graphical Models, 123:113–130, 1999.

[18] Tong Zhang and Keshab K. Parhi, “An FPGA implementation of (3,6)-regular low-density parity-check code decoder,” EURASIP Journal on Applied Signal Processing, special issue on Rapid Prototyping of DSP Systems, vol. 2003, no. 6, pp. 530–542, May 2003

[19] X.Y. Hu, E. Elecftheriou, D.M. Arnold, and A. Dholakia, "Efficient Implementation of the Sum-Product Algorithm for Decoding LDPC Codes", Proceeding of IEEE Globecom'01, pages 1036–1036E, November 2001.

[20]E. Elefthriou, T. Mittelholzer, and A. Dholakia, "Reduced-complexity Decoding Algorithm for Low-Density Parity-Check Codes", IEEE Electronics Letters, 37:102–104, January 2001.

[21] Li Ping and W.K. Leung, "Decoding Low Density Parity Check Codes with Finite Quantization Bits", IEEE Communications Letters, 4(2):62–64, 2000.

[22] D. MacKay and R. Neal, "Good codes based on very sparse matrices," Proc. IMA Conf. Cryptography, Coding, vol. 1025, pp. 100-111, 1995.

[23] S.B. Wicker and S. Kim, "Fundamentals of Codes, Graphs and Iterative Decoding", Kluver Intenational Series in Engineering and Computer Science, 2003

[24] L. Chen, J. Xu, I. Djurdjevic, and S. Lin, "Near Shannon Limit Quasi-Cyclic Low- Density Parity-Check Codes", IEEE Transactions on Communications, 52(7):1038–1042, July 2004

[25]Tong Zhang and Keshab K. Parhi, "A 54 MBPS (3,6)-regular FPGA LDPC decoder", Signal Processing Systems, 2002 (SIPS '02), IEEE Workshop on 16-18 Oct. 2002, pp. 127 – 132

[26] Yijun Li, Mahmoud Elassal and Magdy Bayoumi, "Power efficient architecture for (3,6) Low Density Parity Check Code Decoder", Circuits and Systems, 2004. ISCAS '04. Proceedings of the 2004 International Symposium, Vol.4, pp. IV - 81-4, 23-26 May 2004

[27] Thirupathi M and Ganesan S, "VLSI Implementation of Very High Speed LDPC Decoder", International Journal of Computer Communication and Information System (IJCCIS) Vol2. No1. ISSN: 0976–1349 July – Dec 2010

[28] Luca Fanucci , Francesco Ross, “A Throughput / Complexity Analysis for the VLSI Implementation of LDPC Decoder”Signal Processing and Information Technology, 2004. Proceedings of the Fourth IEEE International Symposium, Pages: 409 – 412, 18-21 Dec., 2004

[29] Wen-Yao Chen and Chung-Chin Lu, "On Error Correction Capability of Bit-Flipping Algorithm for LDPC Codes," Information Theory Proceedings, 2011 IEEE International Symposium, July 31 2011-August 5 2011.

[30] Yuhi Pei, Liuguo Yin, and Jianhua Lu, “Design of Irregular LDPC Codec on a Single Chip FPGA”, IEEE 6th CAS Symp. on Emerging Technologies: Mobile and Wireless Communication Shanghai, China, May 31 - june 2, 2004

[31] David Haley, Member, IEEE, and Alex Grant, Senior Member, IEEE “Reversible Low-Density Parity-Check Codes” IEEE Transaction on information theory, vol. 55, no. 5, pp.2016-2036, may 2009.

[32] Marc P. C. Fossorier, Miodrag Mihaljevic, and Hideki Imai, “Reduced Complexity Iterative Decoding of Low-Density Parity Check Codes Based on Belief Propagation”, IEEE Transactions on Communication, Vol. 47, NO. 5, MAY 1999

[33] Hao Zhong and Tong Zhang, “Joint code-encoder-decoder design for LDPC coding system VLSI implementation” Circuits and Systems, 2004. ISCAS’04. Proceedings of the international Symposium in 2004, Vol. 2, Pages: 389 – 2, 23-26 May, 2004

[34] Abdessalam.Ait madi, Anas.Mansouri1, Ali.Ahaitouf, “Design, Simulation and Hardware implementation of Low Density Parity Check Decoders using Min-Sum Algorithm”, IJCSI International Journal of Computer Science Issues, Vol. 9, Issue 2, No 3, March 2012.

[35] Christian Spagnol, William Marnane, Emanuel Popovici,“Reduced complexity, FPGA implementation of quasi-cyclic LDPC decoder” Circuit Theory and Design, 2005. Proceedings of the European Conference in 2005, Vol. 1, Pages: 289 – 292, 28 Aug.- 2 Sept, 2005

[36] E. Boutillon, J. Castura, and F. R. Kschischang, “Decoder-first code design”, in Proceedings of the 2nd International Symposium on Turbo Codes and Related Topics, pp. 459–462, Brest, France, Sept. 2000. available at http://lester.univ-ubs.fr:8080/˜boutillon/publications.html.

[37] A. J. Blanksby and C. J. Howland, “A 690-mW 1-Gb/s 1024- b, rate-1/2 low-density parity-check code decoder,” IEEE Journal of Solid-State Circuits, vol. 37, no. 3, pp. 404–412, March 2002.

[38] Houshmand Shirani-Mehr, Tinoosh Mohsenin and Bevan Baas, “A Reduced Routing Network Architecture for Partial Parallel LDPC Decoders” Signals, System and Computers (ASLOMAR), 2011 Conference Record of the Forty Fifth Asilomar Conference, Pages: 2192 – 2196, 6 – 9 Nov. 2011,

[39] Thomas J. Richardson and Rudiger L. Urbanke, “Efficient Encoding of Low-Density Parity-Check Codes” IEEE Transaction on Information theory, Vol. 47, NO. 2, FEBRUARY 2001.

[40] Vikram Arkalgud Chandrasetty and Syed Mahfuzul Aziz, “FPGA Implementation of a LDPC Decoder using a Reduced Complexity Message Passing Algorithm” Journal of Networks, vol. 6, NO. 1, January 2011

[41] T. Zhang and K. K. Parhi, “VLSI implementation-oriented (3,k)-regular low-density parity-check codes”, pp. 25–36, IEEE Workshop on Signal Processing Systems (SiPS), Sept. 2001. available at http://www.ecse.rpi.edu/homepages/tzhang/

[42] Z. Zhang, T. Wang and K. K. Parhi,“On finite precision implementation of low density parity-check codes decoder”, in Proceding of 2001 IEEE International Symposium on Circuits and Systems, Sydney, May 2001. available at http://www.ece.umn.edu/groups/ddp/turbo/.

[43] Zhiqiang Cui and Zhongfeng Wang, “Studies on Practical Low Complexity Decoding of Low-Density Parity-Check Codes” Signal Processing Systems, 2007 IEEE workshop on 17-19, Oct. 2007, Pages: 216 - 221

[44] In-Cheol Park and Se-Hyeon Kang,“Scheduling Algorithm for Partially Parallel Architecture of LDPC Decoder by Matrix Permutation” Circuit Systems, 2005. ICAS 2005. IEEE International Symposium, Vol. 6, Page: 5778 – 5781, 23-26, May 2005

[45] T. Zhang and K. K. Parhi, “Joint code and decoder design for implementation-oriented (3,k)-regular ldpc codes,” in Proc. of IEEE Asilomar Conference, Nov. 2001, pp. 1232–1236

[46] C. Maxfield, “The Design Warrior's Guide to FPGAs”, Newnes, 2004

[47] R.H. Morelos-Zaragoza, “The Art of Error Correcting Coding”, John Wiley & Sons, 2002