PERFORMANCE ANALYSIS OF LDPC DECODING ALGORITHMS FOR NEXT GENERATION WIRELESS COMMUNICATIONS

A Project report submitted in partial fulfilment of the requirements for the award of the degree of

BACHELOR OF TECHNOLOGY
IN
ELECTRONICS AND COMMUNICATION ENGINEERING

Submitted by

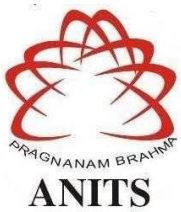
K. Pavani	(318126512085)
R. Aasha	(318126512096)
K. Rumosh	(318126512084)
CH.Karthik	(319126512L09)

Under the Esteemed Guidance of

Mr.R.CHANDRA SEKHAR

M.Tech, (Ph.D)

Assistant Professor, Department of E.C.E



DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING

ANIL NEERUKONDA INSTITUTE OF TECHNOLOGY AND SCIENCES

(UGC-AUTONOMOUS)

(*Permanently Affiliated to AU, Approved by AICTE and Accredited by NBA & NAAC with 'A' Grade*)

Sangivalasa, Bheemli mandal, Visakhapatnam district, 531162, (A.P)

2021-2022

PERFORMANCE ANALYSIS OF LDPC DECODING ALGORITHMS FOR NEXT GENERATION WIRELESS COMMUNICATIONS

Project report submitted in partial fulfilment of the requirements forthe award of the degree of

BACHELOR OF TECHNOLOGY
IN
ELECTRONICS AND COMMUNICATION ENGINEERING

Submitted by

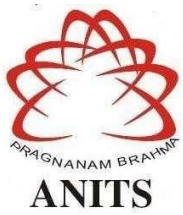
K. Pavani	(318126512085)
R. Aasha	(318126512096)
K. Rumosh	(318126512084)
CH.Karthik	(319126512L09)

Under the Esteemed Guidance of

Mr.R.CHANDRA SEKHAR

M.Tech, (Ph.D)

Assistant Professor Department of E.C.E



DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING

ANIL NEERUKONDA INSTITUTE OF TECHNOLOGY AND SCIENCES

(UGC-AUTONOMOUS)

(*Permanently Affiliated to AU, Approved by AICTE and Accredited by NBA & NAAC with 'A' Grade*)

Sangivalasa, Bheemli mandal, Visakhapatnam district, 531162, (A.P)

2021-2022

DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING
ANIL NEERUKONDA INSTITUTE OF TECHNOLOGY AND SCIENCES

(UGC-AUTONOMOUS)

(Permanently Affiliated to AU, Approved by AICTE and Accredited by NBA & NAAC with 'A' Grade)

Sangivalasa, Bheemili Mandal, Visakhapatnam District-531162, (A.P).

ANITS

CERTIFICATE

This is to certify that the project report entitled **"PERFORMANCE ANALYSIS OF LDPC DECODING ALGORITHMS FOR NEXT GENERATION WIRELESS COMMUNICATIONS"** submitted by K.Pavani (31812651285), R.Aasha (31812651296), K.Rumosh (31812651284), CH.Karthik (319126512L09) in partial fulfillments of the requirements for the award of the degree of ***Bachelor of Technology in Electronics & Communication Engineering*** of Andhra University, Vishakhapatnam is a record of bonafide work carried out under my guidance and supervision.

Project Guide

R. Chandra Sekhar

Mr.R.ChandraSekhar
M.Tech, (Ph.D)
Assistant Professor
Department of E.C.E
ANITS.

Assistant Professor
Department of E.C.E.
Anil Neerukonda
Institute of Technology & Sciences
Sangivalasa, Visakhapatnam-531 162

Head of the Department

Dr. V.RajyaLakshmi
M.E, Ph.D, MIEEE, MIE, MIETE
Professor & H.O.D
Department of E.C.E
ANITS.

Head of the Department
Department of E C E
Anil Neerukonda Institute of Technology & Sciences
Sangivalasa-531 162

ACKNOWLEDGEMENT

We would like to express our deep gratitude to our project guide **Mr.R.Chandra Sekhar** Assistant Professor, Department of Electronics and Communication Engineering (ANITS), for his guidance with unsurpassed knowledge and immense encouragement. We are grateful to **Dr.V.RajyaLakshmi,** Professor and Head of the Department, Electronics and Communication Engineering, for providing us with the required facilities for the completion of the project work.

We are very much thankful to the **Principal and Management, ANITS, Sangivalasa,** for their encouragement and cooperation to carry out this work.

We express our thanks to all **teaching faculty** of Department of ECE, whose suggestions during reviews helped us in accomplishment of ourproject. We would like to thank **all non-teaching staff** of the Department ofECE, ANITS for providing great assistance in accomplishment of our project.

We would like to thank our parents, friends, and classmates for their encouragement throughout our project period. At last but not the least, we thank everyone for supporting us directly or indirectly in completing this project successfully.

PROJECT STUDENTS

K.Pavani (31812651285),

R.Aasha (31812651296),

K.Rumosh (31812651284),

CH.Karthik (319126512L09).

ABSTRACT

Reliability (or) Accuracy of data is the most important thing in communication. There are different components that influence the nature of information when it is transferred over a communication channel like noise, fading etc. In a digital communication system we have to transmit more data via a noisy channel which results in vast no. of errors. To conquer these impacts channel coding schemes are introduced. Channel encoding and decoding techniques are helpful in error control and improves the reliability of the system and reduces the transmit power to achieve a given target BER and prevents re-transmission.

The traditional block codes and conventional codes are commonly used in digital communication. To approach the theoretical limit for Shannon's channel capacity, the length of a linear block code or constant lengths of convolutional codes have to beincreased, which in turn makes the decoder complexity to become high and may render it physically unrealizable. The powerful turbo, LDPC and Polar codes approach the theoretical limit for Shannon's channel capacity with feasible complexity for decoding. Fornext generation wireless communication Low Density Parity Check (LDPC) Codes are keychannel coding techniques for improving BER in significant way.

The major goal of this work is to examine the Bit-Error Rate (BER) and Frame Error Rate (FER) of several LDPC decoding algorithms such as Min-Sum, Normal Min-sum, Mod Min-Sum, Bit flip, Log-Sum Product. Based on simulation results, the Mod Min-Sum decoding algorithm performed better in terms of BER while the Log Sum-Product algorithm performed better in terms of FER. By varying the number of iterations (10,20) and expansion factor (2,32,64) the bit error rate and frame error rate of LDPC decoding methods were improved.

CONTENTS

LIST OF FIGURES

Figures **Page No.**

LIST OF TABLES

Tables **Page No.**

LIST OF ABBREVATIONS

SNR	Signal to Noise Ratio
NR-LDPC	New Radio - Low Density Parity Check
BER	Bit Error Rate
QC LDPC	Quasi Cyclic Low Density Parity Check codes
AWGN	Additive White Gaussian Noise
LLR	Log Likelihood Ratio
SPA	Sum Product Algorithm
CN	Check Nodes
VN	Variable Nodes

CHAPTER-1

INTRODUCTION

Channel coding is the heart of digital communications and data storage. It is often used in digital communication systems to protect the digital information from noiseand interference and reduce the number of bit errors. Channel coding is mostly accomplished by selectively introducing redundant bits into the transmitted information stream. These additional bits will allow detection and correction of bit errors in the received data stream and provide more reliable information transmission.

Earlier the traditional block codes and conventional codes are commonly used in digital communications. To approach the theoretical limit for Shannon's channel capacity, the length of a linear block code or constant lengths of convolutional codes have to be increased, which in turn makes the decoder complexity to become high and make it physically unrealizable. Later on the powerful Reed Solomon (RS), Bose Chaudhuri Hocquenghem (BCH) codes, Turbo codes approach the theoretical limit for Shannon's channel capacity with feasible complexity for decoding [1,2].

Now (4G-5G), Low Density Parity Check (LDPC) and Polar Codes are key channel coding techniques for improving BER in a significant way. LDPC codes are also known as Gallager codes, in honor of Robert G.Gallager, who developed the LDPC concept in his doctoral dissertation at the Massachusetts Institute of Technology in 1960. LDPC code is a linear error correcting code, a method of transmitting a message over a noisy transmission channel. LDPC codes are finding increasing use in applications requiringreliable and highly efficient information transfer over bandwidth-constrained or return-channel-constrained links in the presence of corrupting noise. Implementation of LDPC codes has lagged behind that of other codes, notably turbo codes.

5G is the coming fifth-generation wireless broadband technology based on the IEEE 802.11ac standard. 5G will provide better speeds and coverage than the current 4G. It operates with a 5Ghz signal and is set to offer speeds of up to 1 Gb/s for tens of connections. Commonly accepted use cases for 5G networks are eMBB (Enhanced Mobile Broadband), Massive IoT (Internet of Things) and URLLC (Ultra Reliable and Low Latency Communications). eMBB covers Internet access with high data rates to enable rich media applications, cloud storage and applications, and augmented reality for entertainment [3,4].

1.1 MOTIVATION:

Low-density parity-check codes (LDPC codes) are efficient channel coding codes that allow transmission errors to be corrected.

LDPC codes are finding increasing use in applications requiring reliable and highly efficient information transfer over bandwidth-constrained or return-channel-constrained links in the presence of corrupting noise

Earlier, turbo codes were introduced with iterative decoding algorithms. Utilizing iterative coding and decoding algorithms, it becomes possible to obtain performance within a few tenths of a dB to the Shannon limit for a bit error rate. Later LDPC were introduced. LDPC are comparable to turbo codes which are known to have channel capacity close to the Shannon limit. LDPC codes perform better than the turbo codes, as this scheme has better forward error correction properties and higher detection of incorrect decoding. LDPC codes got a lot of attention in recent years because of several properties. These codes have been chosen for real-time and high-throughput communications. Firstly, the LDPC based scheme is capacity approaching; secondly, these codes can be efficiently decoded by parallel iterative decoding algorithms with low latency. LDPC has got the attention of researchers to evaluate its performances for the development of applications. LDPC codes provide certain advantages over other codes. As compared to turbo codes, LDPC codes are not only simple in code structure but also have a fully parallelizable decoding implementation. LDPC codes, by using message passing algorithms and decoding algorithms have achieved excellentperformance [5, 6].

The LDPC and turbo codes are comparable on design complexity. On one side, by defining the shape of the parity check matrix, it is possible to generate LDPC codes with different rate and block length, whereas, the rate of turbo codes is monitored largely by a puncturing schedule, so flexibility in rate is obtained only through considerable effort while designing. On the other side, LDPC codes have higher encoding complexity than turbo codes, being generically quadratic in the code dimension, although this can be reduced to some extent.

LDPC codes have been proved of closely approaching the channel capacity. In particular using random coding arguments. Hence, we focus the work on five decoding algorithms to compare the bit error rate and frame error rate characteristics between the existing algorithms in LDPC.

1.2 PROJECT OBJECTIVE:

The main objective of this project is to analyze the BER performance fordifferent LDPC decoding techniques like Min-Sum, Normal Min-sum, Mod Min-Sum, Bit flip, Log-Sum Product for next generation wireless communication systems.

The BER and FER performance is analyzed considering the parameters like number ofiterations, expansion factor, SNR values and number of blocks.

1.3 PROJECT OUTLINE:

This document is presented over the four remaining chapters. Chapter 2 gives an overview of channel coding techniques for improving performance of communication system. Chapter 3 describes briefly about the LDPC encoding and decoding techniques used in this project. Chapter 4 describes features and uses of MATLAB. Chapter 5 portrays the simulation results of performance of Bit Error Rate and Frame Error Rate for different SNR values with respect to different decoding algorithms.

CHAPTER-2

CHANNEL CODING TECHNIQUES

2.1 Channel coding techniques for improving performance of communication system

2.1.1 Shannon's Noisy Channel Coding Theorem

Any channel affected by noise possesses a specific "channel capacity" C a rate of conveying information that can never be exceeded without error, but in principle, an error-correcting code always exists such that information can be transmitted at rates lessthan C with an arbitrarily low BER.

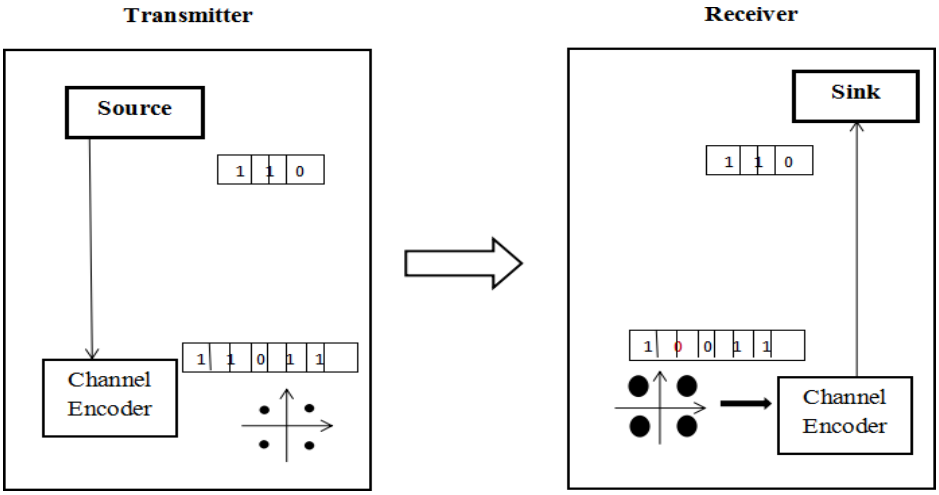


Fig: 2.1 Illustration of channel coding principle

2.1.2 Channel Coding Principle

The channel coding principle is to add redundancy to minimize error rate as illustrated in the below figure.

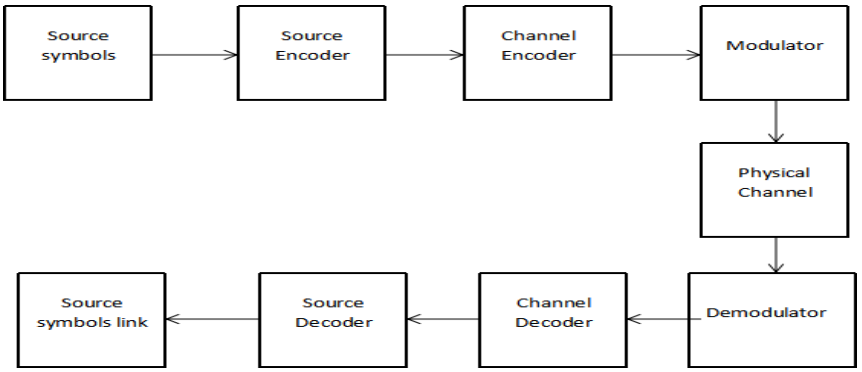


Figure: 2.2 Digital Communication system with coding

2.1.3 Channel Coding Gain

The bit error rate (BER) is the probability that a binary digit transmitted from the source received erroneously by the user. For required BER, the difference between the powers required for without and with coding is called the coding gain. A typical plot of BERversus Eb/N0 (bit energy to noise spectral density ratio) with and without channel coding is shown in Fig. 2.3. It can be seen that coding can arrive at the same value of the BER atlower Eb/N0 than without coding. Thus, the channel coding yields coding gain which is usually measured in dB. Also, the coding gain usually increases with a decrease in BER.

2.2 Types of Channel Codes

There are many types of channel codes, namely linear block codes, convolutional codes, turbo codes, LDPC codes and polar codes.

2.2.1 Block Codes

The data stream is broken into blocks of k bits and each k-bit block is encoded into a block of n bits with n > k bits as illustrated in Fig.2.3. The n-bit block of the channel block encoder is called the code word. The code word is formed by adding (n − k) parity check bits derived from the k message bits.

Some important properties of block codes are defined as

- Block code rate

The block code rate (R) is defined as the ratio of k message bits and length of the code word n.

$$R = k/n \quad \ldots (2.1)$$

- Code word weight

The weight of a code word or error pattern is the number of nonzero bits in the code word or error pattern. For example, the weight of a code word c = (1, 0, 0, 1, 1, 0, 1, 0) is 4.

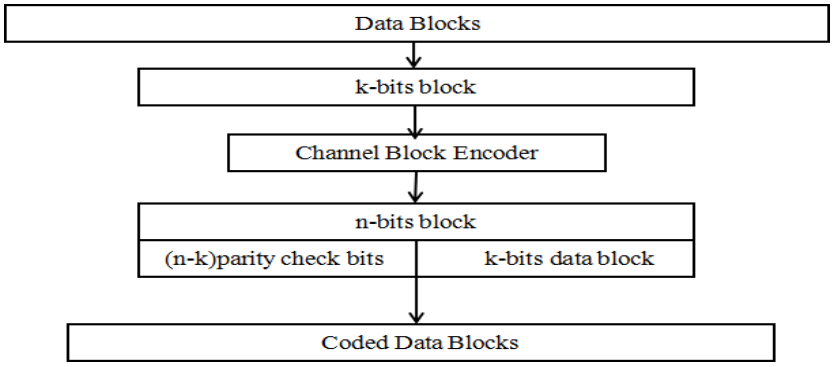


Figure: 2.3 Coded data stream

2.2.2 Linear Block Codes

A block code *C* consisting of *n*-tuples {(c0, c1, … , cn-1)} of symbols from GF(2) is said to be binary linear block code if and only if *C* forms a vector subspace over GF(2). The code word is said to be systematic linear code word if each of the 2*k* code words is represented as linear combination of *k* linearly independent code words.

2.2.3 Hamming Codes

Hamming code is a linear block code capable of correcting single errors having a minimum distance d_{min} = 3. It is very easy to construct Hamming codes. The party check matrix H must be chosen so that no row in H^T is zero and the first $(n-k)$rows of H^T form an identity matrix and all the rows are distinct.

We can select $2^{n-k}-1$ distinct rows of H. Since the matrix H^T has $n$rows, for all of them to be distinct, the following inequality should be satisfied

$2^{n-k}-1 \geq n$ …. (2.2)

Implying that

$$(n-k) \geq \log_2(n+1) \ldots (2.3)$$

$$n \geq k + \log_2(n+1) \quad \ldots (2.4)$$

Hence, the minimum size *n* for the code words can be determined.

2.2.4 Cyclic Codes

An *(n, k)* linear block code *C* is said to be a cyclic code if for every code word *c= (c*0*, c*1*, ... , cn*−2*, cn*−1*) C,* there is also a code word *c*1 *=(cn*−1*, c*0*, c*1*, ..., cn*−2*)* obtained by shifting *c* cyclically one place to the right is also code word in *C.*

2.2.5 Convolutional Codes

In the convolutional coding, the message bits come in serially instead of large blocks. The name convolutional codes are due to the fact that the redundant bits are generated by the use of modulo-2 convolutions in a convolutional encoder. Theconvolutional encoder can be considered as finite-state machine consisting of an M-stage shift register, modulo-2 adders and multiplexers. The rate of a convolutional encoder with *k* inputs and *n* outputs is *k/n*. Often the manufacturers of convolutional code chips specify the code by parameters *(n, k, L).* The quantity *L* is called the constraint length of the code that

represents the maximum number of bits in a single output stream that can be affected by any input bit.

2.2.6 Turbo Codes

The groundbreaking codes called turbo codes. The best known convolution codes are mostly non-systematic. However, in turbo encodes, systematic convolutionalcodes are used. Turbo codes are generated by using the parallel concatenation of two recursive systematic convolutional (RSC) encoders. This chapter discusses turbo encoding, iterative turbo decoding, and performance analysis of turbo codes.

2.2.7 Low Density Parity Check Codes

Low density parity check (LDPC) codes are forward error-correction codes, invented by Robert Gallager in his MIT Ph.D. dissertation, 1960. The LDPC codes are ignored for long time due to their high computational complexity and domination of highly structured algebraic block and convolutional codes for forward error correction. A number of researchers produced new irregular LDPC codes which are known as new generalizations of Gallager's LDPC codes that outperform the best turbo codes with certain practicaladvantages. LDPC codes have already been adopted in satellite based digital video broadcasting and long-haul optical communication standards. This chapter discusses LDPC Code Properties, construction of parity check matrix for regular and irregular LDPC codes, efficient Encoding and Decoding of LDPC Codes, performance analysis of LDPC Codes.

2.2.8 Polar Codes

A new channel coding has flourished known as polar coding, and it is a channel coding scheme that was invented by Erdal Arıkan at Bilkent University (Ankara, Turkey). Polar codes are said to achieve channel capacity in a given binary discrete memory less channel. This can be achieved only when the block size is large enough. The complexity of encoding and decoding is less and these codes can be successfully decoded. The main idea behind the polar codes invented by Arikan is the channel polarization. The channels can be categorized into good and bad by channel polarization. The recursive application of the polarization transformation makes the synthesized channels having better reliability for good channels and worse reliability for bad channels. The channels can be distinctly polarized with increase in code length and the information bits are transmitted over good channels, whereas the frozen bits are transmitted over bad channels.

CHAPTER 3

LOW DENSITY PARITY CHECK CODES

Low density parity check (LDPC) codes are forward error-correction codes, invented by Robert Gallager in his MIT Ph.D. dissertation, 1960. The LDPC codes are ignored for long time due to their high computational complexity and domination of highly structured algebraic block and convolutional codes for forward error correction. A number of researchers produced new irregular LDPC codes which are known as new generalizations of Gallager's LDPC codes that outperform the best turbo codes with certain practicaladvantages. LDPC codes have already been adopted in satellite based digital video broadcasting and long-haul optical communication standards. This chapter discusses LDPC Code Properties, construction of parity check matrix for regular and irregular LDPC codes, efficient Encoding and Decoding of LDPC Codes, performance analysis of LDPC Codes.

3.1 LDPC Code Properties

Low Density Parity Check (LDPC) code is a linear error-correcting code that has a parity check matrix *H*, which is sparse i.e. with less nonzero elements in each row and column. LDPC codes can be categorized into regular and irregular LDPC codes. When the parity-check matrix $H_{(n-k)\times n}$ has the same number w_c of ones in each column and the same number w_r of once in each row, the code is a regular (w_c, w_r). The original Gallager codes are regular binary LDPC codes. The size of *H* is usually very late, but the density of nonzero element is very low. LDPC code of length *n*, or denoted as an (n, w_c, w_r)LDPC code. Thus, each information bit is involved with w_c parity checks, and each parity-check bit is involved with w_r information bits. For a regular code, we have thus$w_c < w_r$. If all rows are linearly independent, the code rate is $\frac{(w_r - w_c)}{w_c}$, otherwise it is *k/n*. Typically, $w_c \geq 3$. A parity check matrix with minimum column weight w_c will have a minimum distance$d_{min} \geq w_c + 1$. When $w_c \geq 3$, there is atleast one LDPC code whose minimum distance d_{min} grows linearly with the block length *n* [1]; thus a longer code length yields a better coding gain. Most regular LDPC codes are constructed with w_c and w_r on the order of 3 or 4.

3.2 Construction of Parity Check Matrix *H*

3.2.1 Gallager Method for Random Construction of H for Regular Codes

In this method, the transpose of regular (n, w_c, w_r) parity check matrix *H* has the form

$$H^T = [H_1^T, H_2^T, \ldots, H_{w_c}^T] \ldots (3.1)$$

The matrix H_1has n columns and n/w_r rows. The H_1 contains a single 1 in each column and contains 1s in its ith row from column $(i-1)w_r + 1$ to column$i\, w_r$. Permuting randomly the columns of H_1 with equal probability, the matrices H_2 to $H_{w_c}$are obtained.

The parity check matrix for (n = 20,w_c = 3,w_r= 4)code constructed by Gallager is given as

$$H = \begin{bmatrix}
1&1&1&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&1&1&1&1&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&1&1&1&1&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&1&1&1&1&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&1&1&1\\
1&0&0&0&1&0&0&0&1&0&0&0&1&0&0&0&0&0&0&0\\
0&1&0&0&0&1&0&0&0&1&0&0&0&0&0&1&0&0&0&0\\
0&0&1&0&0&0&0&0&1&0&0&0&1&0&0&0&1&0&0&0\\
0&0&0&1&0&0&0&0&0&1&0&0&0&1&0&0&0&1&0&0\\
0&0&0&0&0&0&0&1&0&0&0&1&0&0&0&1&0&0&0&1\\
1&0&0&0&0&1&0&0&0&0&0&1&0&0&0&0&0&1&0&0\\
0&1&0&0&0&0&1&0&0&0&1&0&0&0&0&1&0&0&0&0\\
0&0&1&0&0&0&0&1&0&0&0&1&0&0&0&0&0&1&0&0\\
0&0&0&1&0&0&0&0&1&0&0&0&1&0&0&1&0&0&0&0\\
0&0&0&0&1&0&0&0&0&1&0&0&0&1&0&0&0&0&1&1
\end{bmatrix} \ldots(3.2)$$

The following MATLAB program can be used to generate Gallager regular parity check matrix H with different code rates.

3.2.2 Algebraic Construction of H for Regular Codes

The construction of the parity check matrix H using algebraic construction as follows. Consider an identity matrix where $a > (w_c - 1)(w_r - 1)$ and obtain the following matrix by cyclically shifting the rows of the identity matrix $I_a$by one position to the right.

$$A = \begin{bmatrix}
0&1&0&0&\ldots&0\\
0&0&1&0&\ldots&0\\
0&0&0&1&\ldots&0\\
0&0&0&0&\ldots&1\\
1&0&0&0&\ldots&0
\end{bmatrix} \ldots(3.3)$$

Defining $A^0 = I_a$the parity check matrix H can be constructed as

$$H = \begin{bmatrix}
A^0 & A^0 & A^0 & \cdots & A^0\\
A^0 & A^1 & A^2 & \cdots & A^{(w_r-1)}\\
A^0 & A^2 & A^4 & \cdots & A^{2(w_r-1)}\\
A^0 & A^{(w_r-1)} & A^{2(w_r-1)} & \cdots & A^{(w_r-1)(w_r-1)}
\end{bmatrix} \ldots (3.4)$$

The constructed *H* matrix has $w_c a$rows and $w_r a$columns, and it is of a regular $(w_r a, w_c, w_r)$ having the same number of w_r ones in each row and the same number of w_c ones in each column. It is four cycle free construction. The algebraic LDPC codes are easier for decoding than random codes. For intermediate *n*, well designed algebraic codes yields a low BER.

3.2.3 Random Construction of H for Irregular Codes

In the random construction of the parity check matrix H, the matrix is filled with ones and zeros randomly satisfying LDPC properties.

An example of parity check matrix for irregular LDPC code is

$$H = \begin{bmatrix} 1\,1\,0\,1\,1\,0\,0\,1\,0\,0 \\ 0\,1\,1\,0\,1\,1\,1\,0\,0\,0 \\ 0\,0\,0\,1\,0\,0\,0\,1\,1\,1 \\ 1\,1\,0\,0\,0\,1\,1\,0\,1\,0 \\ 0\,0\,1\,0\,0\,1\,0\,1\,0\,1 \end{bmatrix} \ldots (3.5)$$

3.3 Representation of Parity Check Matrix Using Tanner Graphs

The Tanner graph of the parity check matrix *H* is a bipartite graph. if variable *i* participates in the j^{th} parity-check constraint, then check node *j* is connected to variable node i.

3.3.1 Cycles of Tanner Graph

Consider the following parity-check matrix

$$H = \begin{bmatrix} 1\;\;1\,0\,1\,1\,0 \\ 1\,0\,0\,0\,1\,0 \\ 0\,0\,1\,0\,0\,1 \\ 0\,1\,1\,1\,0\,1 \end{bmatrix}$$

The Tanner graph of the *H* matrix is shown in Fig.3.2. A sequence of connected nodes starting and ending at the same node with no node more than once is a cycle of a Tanner graph. The number of edges in a cycle is called cycle length and the smallest size of the cycle in a graph represents the girth of the graph. Cycles of length 4 situations arise where pairs of rows share 1s in a particular pair of columns of the above *H* matrix. A cycle of length 4 is shown in bold in Fig.3.2. The minimum lower bound distance for four cycle free (w_c, w_r) regular LDPC code parity check matrix with girth g, is given by

$$d_{min} \begin{cases} 1 + w_c + w_c(w_c - 1) + w_c(w_c - 1)^2 + \cdots + w_c(w_c - 1)^{\frac{g-4}{4}} \; for\; odd\; g/2 \\ 1 + w_c + w_c(w_c - 1) + w_c(w_c - 1)^2 + \cdots + w_c(w_c - 1)^{\frac{g-8}{4}} \; otherwise \end{cases} \ldots (3.7)$$

Thus the minimum distance can be increased by increasing the girth or the column weight.

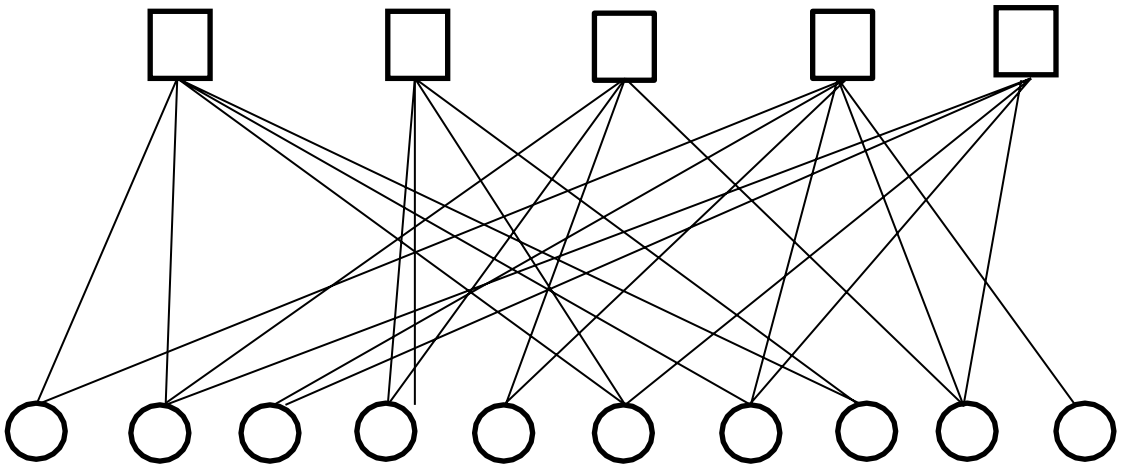


Fig: 3.1 Tanner graph of H matrix of example: 3.2

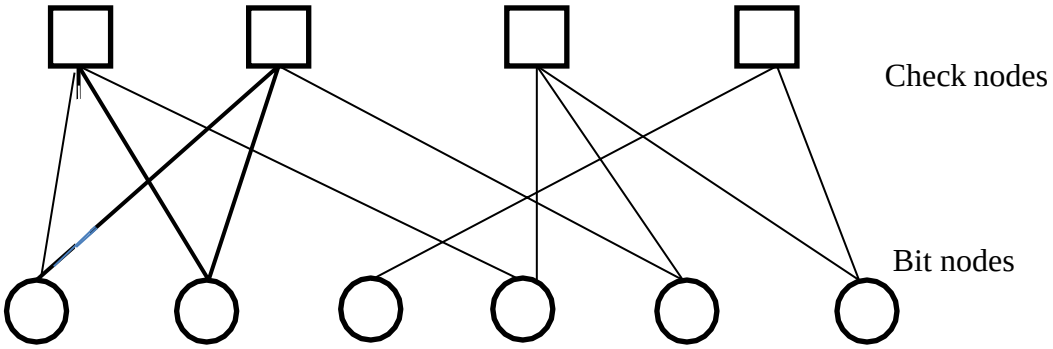


Fig: 3.2 A tanner graph with a cycle of length 4

3.3.2 Detection and Removal of Girth 4 of a Parity Check Matrix

If the Tanner graph of a parity check matrix contains no loops then this decoding is quickly computable. Unfortunately, LDPCs have loopy graphs, and so the algorithm needs to be repeatedly iterated until it converges to a solution. The effect of girth on the performance of LDPC codes can be reduced by choosing the codes having Tanner graphs with longer girths. However, longer girths are not helpful for finite length codes. A girth of 6 is sufficient and hence the removal of girth 4 is a required. A lemma states that the *H* matrix has no girth 4, if and only if all the entries of the matrix $[H^T H]$ are 1s except the diagonal line. A standard approach is to search the parity-check matrix *H* forming a rectangle of four 1s in the matrix. Eliminating the rectangle by reshuffling some elements around while preserving the other relevant properties of the matrix is equivalent to removing a girth 4 from the Tanner graph.

3.4 LDPC Encoding

3.4.1 Reprocessing Method

For coding purposes, we may derive a generator matrix *G* from the parity check matrix *H* for LDPC codes by means of Gaussian elimination in modulo-2 arithmetic. Since the matrix *G* is generated once for a parity check matrix, it is usable in all encoding of messages. As such this method can be viewed as the pre-processing method.

1-by-*n* code vector *c* is first partitioned as

$$C = [b : m] \ldots (3.8)$$

Where *m* is *k* by 1 message vector, and *b* is the *n*−*k* by 1 parity vector correspondingly, the parity check matrix *H* is partitioned as

$$H^T = \begin{bmatrix} H_1 \\ \ldots \\ H_2 \end{bmatrix} \ldots (3.9)$$

Where *H*1 is a square matrix of dimensions $(n-k) \times (n-k)$, H_2 and is a rectangular matrix of dimensions $k \times (n-k)$ transposition symbolized by the superscript T is used in the partitioning of matrix *H* or convenience of representation.
Imposing the constraint $CH^T = 0$.
We may write

$$[b: m] \begin{bmatrix} H_1 \\ \ldots \\ H_2 \end{bmatrix} = 0 \quad \ldots (3.10)$$

Or equivalently,

$$bH_1 + mH_2 = 0 \ \ldots \ (3.11)$$

The vectors m and b are related by

$$b = mP \ \ldots (3.12)$$

Where, *P* is the coefficient matrix. For any nonzero message vector *m*, the coefficient matrix of LDPC codes satisfies the condition

$$PH_1 + H_2 = 0 \ \ldots (3.13)$$

Which holds for all nonzero message vectors and, in particular, in the form [0 ... 0 1 0 ... 0] that will isolate individual rows of the generator matrix. Solving
Eq. (8.13) for matrix P, we get

$$P = H_2 \, H_1^{-1} \ \ldots (3.14)$$

Where H_1^{-1} is the inverse matrix of H_1, which is naturally defined in modulo-2 arithmetic. Finally, the generator matrix of LDPC codes is defined by

$$G = [P : I_k] = [H_2 \, H_1^{-1} : \ I_k] \ \ldots (3.15)$$

Where I_k is the k by k identity matrix. The code word can be generated as

$$C = mG \ \ldots (3.16)$$

3.5 Efficient Encoding of LDPC Codes

The pre-processing method discussed in Sect. 3.4.1 for finding a generator matrix *G* for a given *H* can be used for encoding any arbitrary message bits vector of size 1 × *m*. However, it has a complexity of (n^2) . LDPC code can be encoded using the parity-check matrix directly by using the efficient encoding method which has a complexity of *O(n)*. The stepwise procedure of efficient coding of LDPC coding is as follows:

Step 1: By performing row and column permutations, the non-singular parity check matrix *H* is to be brought into a lower triangular form indicated in Fig. 3.3 More precisely, the *H* matrix is brought into the form

$$H_t = \begin{bmatrix} A & B & T \\ C & D & E \end{bmatrix} \ldots (3.17)$$

with a gap length *g* as small as possible.

Where *A* is *(m − g) × (n − m)* matrix, *B* is *(m − g) × g* matrix, *T* is *(m − g) × (m − g)* matrix, *C* is *g × (n − m)* matrix, *D* is *g × g* matrix and *E* is *g × (m − g)* matrix. All of these matrices are sparse and *T* is lower triangular with ones along the diagonal.

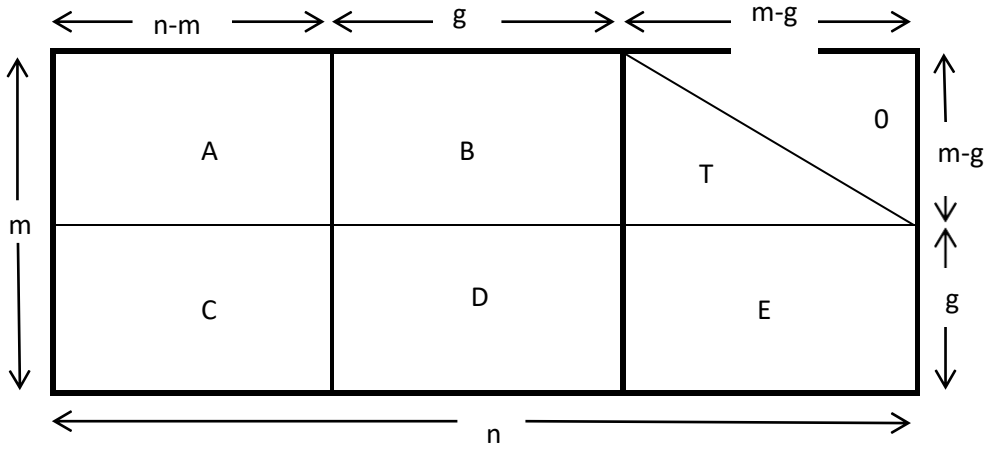


Fig: 3.3 The parity-check matrix in approximate lower triangular form

Step 2: Premultiply H_t by $\begin{bmatrix} I_{m-g} & 0 \\ -ET^{-1} & I_g \end{bmatrix}$

In order to check that $-ET^{-1}B + D$is non-singular. It is to be ensured by performing column permutations further.

Step 3: ObtainP_1 using the following

$$P^T{}_1 = \emptyset^{-1}(-ET^{-1}A + C)s^T \ \ldots (3.19)$$

Where

$$\emptyset = -ET^{-1}B + D \text{and s is message vector.}$$

Step 4: ObtainP_2 using the following

$$2\mathrm{T} \ = \ -\mathrm{T} - 1(\mathrm{AsT} \ + \ \mathrm{BP1T})$$

Step 5: Form the code vector c as

$$c = [s \ \ p_1 \qquad p_2] \ldots (3.21)$$

$p_1$holds the first g parity and p_2 contains the remaining parity bits.

3.6 LDPC Decoding

In the LDPC decoding, the notation B_j is used to represent the set of bits in the parity check equation of H, and the notation A_i is used to represent the parity check equations for the i th bit of the code. Consider the following parity check matrix

$$H = \begin{bmatrix} 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 \end{bmatrix} \ldots (3.22)$$

For the above parity check matrix, we get

$B_1 = \{1,2,3\}, \qquad B_2 = \{1,4,5\}, \qquad B_3 = \{2,4,6\}, \qquad B_4 = \{3,5,6\},$

$A_1 = \{1,2\}, \qquad A_2 = \{1,3\}, \ A_3 = \{1,4\}, \ A_4 = \{2,3\}, \ A_5 = \{2,4\}, \qquad A_6 = \{3,4\}$

3.6.1 LDPC Decoding on Binary Erasure Channel Using Message Passing Algorithm

The message passing algorithms are iterative decoding algorithms which passes the messages back and forward between the bit and check nodes iteratively until the process is stopped. The message labeled$M_i$indicates 0 or 1 for known bit values and e for erased bit thestepwise procedure for LDPC decoding on BEC is as follows:

Step 1: Set M= y, find B_j and A_i of HStep

2: $iter = 1$

Step 3: If all the messages into check j other than M_i are known, compute all check sums by using the following expression

$$E_{i,j} \ = \ \sum_{i' \in B_j, i' \neq i} (M_i mod \ 2) \ldots (3.23)$$

Else $E_{i,j}$= e

Step 4: If $M_i = e$ and if $j \in A_i$ subject to $E_{i,j} \neq e$, set $M_i = E_{i,j}$.

Step 5: If all M_i are known or $iter = iter_{max}$, stop, else

Step 6: $iter = iter + 1$, go to step 3.

3.6.2 Bit-Flipping Decoding Algorithm

The received symbols are hard decoded into 1s and 0s to form a binary received vector y. In each iteration, it computes all check sums, as well as the number of unsatisfied parity checks involving each of the n bits of the vector y. Next, the bits of y are flipped if they involve in the largest number of unsatisfied parity checks. The process is to be repeated until all check sums are satisfied or reaches a predetermined number of iterations. The stepwise procedure of the Bit-flipping decoding algorithm is as follows:

Step 1: Set M= y, define B_j to represent the *j*th parity check equation of H

Step 2: $l = 0$

Step 3: Compute all check sums by using the following expression

$$E_{i,j} = \sum_{i' \in B_j, i' \neq i} (M_{i'} \mod 2) \ldots (3.24)$$

Step 4: Compute the number of unsatisfied parity checks involving each of *n* bits of message.

Step 5: Flip the bits of message when they are involved in largest number of unsatisfied parity checks. The flipping on *i*th bit can be performed by using

$$M_i = (y_i + 1 \mod 2) \ldots (3.25)$$

Step 6: Compute s as follows

$$s = (MH^T) \mod 2 \ldots (3.26)$$

Step 7: If $s = 0$ *or* $l = l_{max}$, stop, else

Step 8: $l = l + 1$, go to step 3.

3.7 Sum Product Decoding

The sum product algorithm is similar to the bit-flipping algorithm as described in theprevious section, but the messages representing each decision (whether the bit value is 1 or0) are now probabilities. Bit-flipping decoding accepts an initial hard decision on the received bits as input and the sum-product algorithm is a soft decision message passing algorithm which accepts the probability of each received bit as input. The input channel or received bit probabilities are known in advance before the LDPC decoder was operated and so they are also called as the a priori probabilities of the received bit. In the sum product decoder, the extrinsic information passed between nodes is also probabilities. The extrinsic information between check node *j* and bit node *i* is denoted by $E_{j,i}$. The $E_{j,i}$ gives the probability for the bit $c_i$to be 1 that causes the parity check equation *j* is satisfied. The $E_{j,i}$

cannot be defined if the bit *i* not included in *j* as there will be no extrinsic information between check node *j* and bit node *i*.
The probability that an odd number of the bits in that parity check equation are 1s is given by

$$P_{j,i}^{ext} = \frac{1}{2} - \frac{1}{2}\prod i' \in B_{j,i' \neq i}\left(1 - 2P_{j,i'}\right) \ldots (3.27)$$

Which is the probability that a parity check equation is satisfied for the bit $c_i$to be 1. The probability that the parity check equation is satisfied for the bit $c_i$to be 0 becomes $1 - P_{j,i}^{ext}$.

The metrics for a binary variable is represented by the following log likelihood ratio (LLR)

$$L(x) = \log \frac{p(x=0)}{p(x=1)} \ldots (3.28)$$

where by log we mean loge. The sign of *L(x)* provides a hard decision on *x* and magnitude $|L(x)|$ is the reliability of this decision. Translating from log likelihood ratios back to probabilities,

$$p(x = 1) = \frac{e^{-L(x)}}{1+e^{-L(x)}} \ldots (3.29)$$

$$p(x = 0) = \frac{e^{L(x)}}{1+e^{-L(x)}} \ldots (3.30)$$

when probabilities need to be multiplied, log likelihood ratios need only be added and by this the complexity of the sum product decoder is reduced. This makes the benefits of thelog arithmetic representation of probabilities. The extrinsic information from check node *j* tobit node *i* is expressed as a log likelihood ratio,

$$E_{j,i} = L(P_{j,i}^{ext} \log \frac{1 - 1 - p_{j,i}^{ext}}{p_{j,i}^{ext}} \ldots (3.31)$$

Now

$$E_{j,i} = \log \frac{\frac{1}{2} + \frac{1}{2}\prod i' \in B_{j,} i' \neq i(1 - 2P_{j,i'})}{\frac{1}{2} - \frac{1}{2}\prod i' \in B_{j,} i' \neq i(1 - 2P_{j,i'})}$$

$$= \log \frac{1 + \prod i' \in B_{j,} i' \neq i(1 - 2\frac{e^{-M_{j,i'}}}{1 + e^{-M_{j,i'}}})}{1 - \prod i' \in B_{j,} i' \neq i(1 - 2\frac{e^{-M_{j,i^i}}}{1 + e^{-M_{j,i'}}})}$$

$$= \log \frac{1 + \prod i' \in B_{j,} i' \neq i \left(\frac{e^{-M_{j,i'}}}{1 + e^{-M_{j,i'}}} \right)}{1 - \prod i' \in B_{j,} i' \neq i \left(\frac{e^{-M_{j,ii}}}{1 + e^{-M_{j,i'}}} \right)} \ldots (3.32)$$

Where $Mj,i' \triangleq L(Pj,i') = \log 1-P\ j,i'\ Pj,i$

Using the relationship

$$\tanh \frac{1}{2} \log \left(\frac{1-p}{p} \right) = 1 - 2p \ldots (3.33)$$

Gives

$$E_{j,i} = \log \frac{1 + \prod i' \in B_{j,} i' \neq i^{\tanh(M_{j,i'}/2)}}{1 - \prod i' \in B_{j,} i' \neq i^{\tanh(M_{j,i'}/2)}} \ldots (3.34)$$

Alternatively, using the relationship

$$2 \tanh^{-1} p = \log \frac{1+p}{1-p} \ldots (3.35)$$

$$E_{j,i} = 2\tanh^{-1} \prod \prod i' \in B_{j,i' \neq i} \tanh \left(\frac{M_{j,i'}}{2} \right) \ldots (3.36)$$

The above equation is numerically challenging due to the presence of the product of the $tanh$ and$tanh$ −1 functions. Following Gallager, we can improve the situation as follows. First, factor Mj,i into its sign and magnitude (or bit value and bit reliability);

$$M_{ji} = \alpha_{ji} \beta_{ji} \ \ldots (3.37)$$

$$\alpha_{ji} = \text{sign}(M_{ji}) \ldots (3.38)$$

$$\beta_{ji} = |M_{ji}| \ldots (3.39)$$

So that eq.(3.39) may be rewritten as

$$\tanh\left(\frac{1}{2}M_{ji}\right) = \prod_{i'} \alpha_{ji} \prod_{i' \in B_{j,} i' \neq i} \tanh\left(\frac{1}{2}\beta_{ji}\right) \ldots (3.40)$$

We then have

$$E_{j,i} = \prod_{i} \alpha_{ji}. 2\tanh^{-1}\left(\prod_{i'} \tanh\left(\frac{1}{2}\beta_{ji}\right)\right)$$

$$= \prod_{i} \alpha_{ji}. 2\tanh^{-1} log^{-1} \log\left(\prod_{i'} \tanh\left(\frac{1}{2}\beta_{ji}\right)\right)$$

$$\prod_{i'} \alpha_{ji}. 2\tanh^{-1} log^{-1} \sum_{i'} \log\left(\frac{1}{2}\beta_{ji}\right) \ldots (3.41)$$

This yields a new from for Eq. 3.41 as

$$E_{ji} = \prod_{i'} \alpha_{ji'} \,. \emptyset\left(\sum_{i'} \emptyset\left(\beta_{ji'}\right)\right) \ldots (3.42)$$

Where $\emptyset(x)$ is defined as

$$\emptyset(x) = -\log\left[tanh\frac{x}{y}\right] = log\left(\frac{e^x+1}{e^x-1}\right) \ldots (3.43)$$

Using the fact that $\emptyset -1\ (x) = \emptyset(x)$ when x > 0.

Each bit node has access to the input LLR, L_i, and to the LLRs from every connected check node. The total LLR of the ith bit is the sum of these LLRs:

$$L_i^{total} = L_i + \sum_{j \in A_i} E_{ji} \ldots (3.44)$$

The hard decision on the received bits is simply given by the signs of the L_i^{total}. Check whether the parity check equations are satisfied (thus $\hat{c}H^T = 0$ is also a stopping criterion for sum product decoding), if not satisfied update M_{ji}

$$M_{ji} = \sum_{j\prime \in A_{i,} j\prime \neq i} E_{j\prime i} + L_i \ldots (3.45)$$

The algorithm outputs the estimated a posteriori bit probabilities of the received bits as log likelihood ratios.

The sum-product decoder immediately stops whenever a valid codeword has been found by a checking of whether the parity check equations are satisfied (i.e., $\hat{c}H^T = 0$) or allowed maximum number of iterations achieved. The decoder is initialized by setting all VN messages M_{ji} equal to

For all *j,I* for which $h_{ij} = 1$. Here, $y_j$represents the channel value that was actually received, that is, it is not a variable here. The $L_i$for different channels can be computed as

BEC

In this case, $y_j \in \{0,1,e\}$

$$L_i = L(c\backslash y_i) = log\left(\frac{Pr(c_i=0|y_i)}{Pr(c_i=1|y_i)}\right) \ldots (3.46)$$

$$L_i = L(c\backslash y_i) = \begin{cases} +\infty y_j = 0, \\ -\infty y_j = 1, \ldots (3.47) \\ 0 \quad y_j = e. \end{cases}$$

BSC

In this case, $y_j \in \{0,1\}$, we have

$$L_i = L(c_i\backslash y_i) = (-1)^{y_i} log\frac{1-p}{p} \ldots (3.48)$$

The knowledge of crossover probability P is necessary

BI-AWGNC

The model for Rayleigh fading channel is similar to that of the AWGNC: $y_i = \alpha_i\beta_i + n_i$where $\{\alpha_i\}$ are independent Rayleigh random variables with unity variance. The channel transition probability can be expressed by

$$P(x_i = x|y_i) = \frac{1}{1 + \exp(-4\alpha_i y_i/N_0} \ldots (3.49)$$

Then

$$L(c_i|y_i) = \frac{4\alpha_i y_i}{N_0} \ldots (3.49)$$

The estimates of $\alpha_i$and $\sigma^2$are necessary in practice.

Now, the stepwise procedure for the log domain sum product algorithm is given in the following Sect.8.8

3.7.1 Log Domain Sum-Product Algorithm (SPA)

Step 1: Initialization: for all i, initialize $L_i$according to Eq. (3.44) for the appropriate channel model. Then, for all i, j for which $h_{i,j} = 1$ set $M_{ji} = L_i$and $l = 0$. Define B_i to represent the set of bits in the j th parity check equation of H and $A_i$to represent the parity check equationsfor the i th bit of the code.

Step 2: CN update: compute outgoing CN message E_{ji} for each CN using Eqs. (3.37), (3.38) and (3.39).

$$M_{ji} = \alpha_{ji}\beta_{ji}$$

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

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

$$E_{ji} = \prod_{i'} \alpha_{ji'} \cdot \emptyset\left(\sum_{i'} \emptyset\left(\beta_{ji'}\right)\right)$$

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

Step 3: LLR total: For $i = 0, 1, \ldots, N-1$ compute total LLR using Eq. (3.41)

$$L_i^{total} = L_i + \sum_{j \in A_i} E_{ji}$$

Step 4: Stopping criteria: For $i = 0, 1, \ldots, N-1$, set

$$\hat{C}_i = \begin{cases} 1 \; if \; L_i^{total} < 0, \\ 0 \; else, \end{cases}$$

To obtain $\hat{c}$. If $\hat{c}H_t = 0$ or the number of iterations equals the maximum limit $(l = l_{max})$ stop;

else

Step 5: VN update: compute outgoing VN message $M_{ji}$for each VN using Eq. (3.42)

$M_{ji} = \sum_{j' \in A_{i,j' \neq j}} E_{j'i} + L_i$. $l = l + 1$go to step 2.

3.7.2 The Min-Sum Algorithm

Consider Eq. (3.39) for E_{ji}. It can be noted from the shape of $\emptyset(x)$that the largest term in the sum corresponds to the smallest β_{ji}. Hence, assuming that this term dominates the sum, the following relation is obtained

$$\emptyset\left(\sum_{i'} \emptyset(\beta_{ji'})\right) \cong \emptyset\left(\emptyset\left(\min_{i'} \beta_{ji'}\right)\right) = \min_{i'} \beta_{ji'} \ldots \quad (3.50)$$

$$M_{ji} = \alpha_{ji}\beta_{ji}$$

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

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

$$E_{ji} = \prod_{i'} \alpha_{ji'} \; . \emptyset\left(\sum_{i'} \emptyset\left(\beta_{ji'}\right)\right)$$

It can also be shown that, in the AWGNC case, the initialization $M_{ji} = 4\ y_i/N_0$ may be replaced by $M_{ji} = y_i$when the simplified log domain sum product algorithm is employed. The advantage, of course, is that an estimate of the noise power N_0 is unnecessary in this case.

3.8 Performance Analysis of LDPC Codes

3.8.1 Performance Comparison of Sum-Product and Min-Sum Algorithms for Decoding of Regular LDPC Codes in AWGN Channel

The BER performance of the Sum-Product and Min-Sum LDPC decoding algorithms is evaluated through a computer simulation assuming that the channel adds white Gaussian noise to the code generated by a (256, 3, 6) regular parity check matrix. In this simulation,

four hundred frames of each of length 256 and three iterations are used. The BER performance of the Sum-Product and Min-Sum Algorithms is shown in Fig. 3.4.

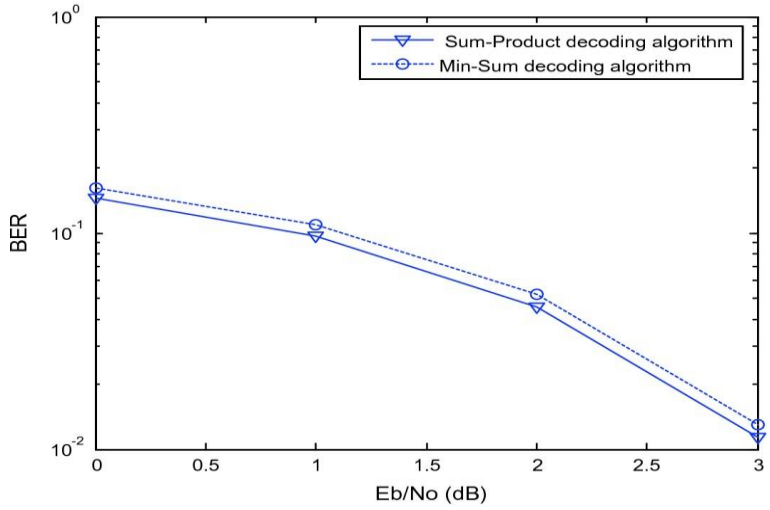


Fig. 3.4 BER performance of Sum-Product and Min-Sum decoding algorithms

3.8.2 BER Performance Comparison of Regular and Irregular LDPC Codes in AWGN Channel

The performance of rate 1/2 regular and irregular codes having the same length is evaluated through a computer simulation. The BER performance of the two codes is shown in Fig. 3.5. From Fig. 3.5, it is observed that there is no significant difference between the BER performance of the Sum-Product and the Min-Sum algorithms. The irregular codes can have improved thresholds for long codes but with an error floor at higher BER than for regular codes of the same rate and length.

3.8.3 Effect of Block Length on the BER Performance of LDPC Codes in AWGN Channel

The effect of block length on the performance of LDPC codes is illustrated through a computer simulation. In this experiment, two 1/2 rate irregular codes of block lengths 256, and 512 are considered and added white Gaussian noise to them, and the noisy codes are decoded using Min-Sum decoding algorithm with 10 iterations. The BER performance of the two codes is shown in Fig. 3.6

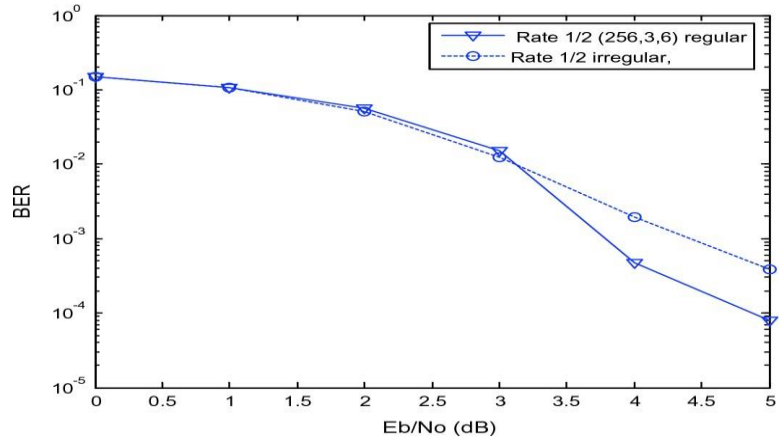


Fig. 3.5BER performance of rate 1/2 regular and irregular LDPC codes using Min-Sum decoding algorithms

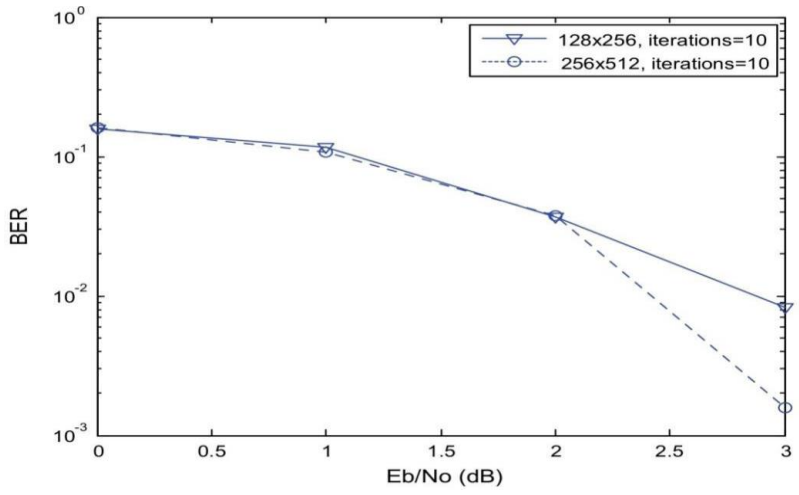


Fig. 3.6BER performance of rate 1/2 regular and irregular LDPC codes using Min-Sum Logarithm decoding algorithms

3.8.4 Error Floor Comparison of Irregular LDPC Codes of Different Degree Distribution in AWGN Channel

The error-floor of an LDPC code is characterized by the phenomenon that as the SNR continues to increase, the error probability suddenly drops at a rate much slower than that in the region of low to moderate SNR can be approximated by Johnson and Weller

$$BER_{ef} \approx \frac{2}{N}\frac{\left(\lambda_2\rho(1)\prime\right)^2}{4}\mathrm{Q}\left(\sqrt{\frac{4RE_b}{N_0}}\right) \ldots . (3.51)$$

with the constraint $\lambda_2\rho(1)' \leq Eexp(\frac{1}{2\sigma_2})$, where E varies from 0 to 1, E = 1 for the traditional optimized degree distributions, E is greater than zero but less than 1 for constrained degree distributions, N is the length of the code and R is the code rate. A trade- off between the threshold and error floor can be achieved with the constrained distributions.

3.9 Quasi Cyclic (QC)-LDPC CODES

The principal property of QC-LDPC codes is that their parity check matrix consists of circulant sub matrices, which could be either based on the identity matrix or a smaller random matrix. The main advantage of this construction principle compared to randomly constructed codes is that QC-LDPC encoding procedure is easier to implement. The encoderof QC-LDPC codes can be implemented by using a series of shift registers, which allows its complexity to be proportional to code length QC-LDPC codes have been used widely in high-throughput systems, such as IEEE 802.16e, IEEE 802.11n, and IEEE 802.11ac, and IEEE 802.11ad. QC-LDPC codes are very suitable for high-throughput and low-latencysystem.

3.9.1 Brief Description of QC-LDPC Codes

Let $H_{M\times N}$be the parity check matrix of a QC LDPC code. The $H_{M\times N}$matrix of consists of m_b rows and n_b columns of sub matrices of size $z \times z$, where z is a block size, $M = (m_b \times z)$ and $N = (n_b \times z)$. A sub matrix is either a null matrix, or a circulate matrix obtained by shifting cyclically each row of an identity matrix to the right for p steps. Define

$$P_z \triangleq \begin{bmatrix} 0 & 1 & 0 & \cdots & 0 \\ 0 & 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & 1 \\ 1 & 0 & 0 & \cdots & 0 \end{bmatrix} \ldots (3.52)$$

then, a sub matrix of size $z \times z$with shift step p, $0 \leq p \leq z - 1$, is $(P_z)^P$ the Pth power of P_z. Conventionally, $(P_z)^P$ is defined as identity matrix.

A trivial example for the parity check matrix H with $M = 6, N = 12, z = 3, m_b = 2, n_b = 4$, which is portioned into 8 sub matrices is as shown

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

3.9.2 Base Matrix and Expansion

Parity check matrix of a QC-LDPC code is often described by a $m_b \times n_b$ base matrix$B_{m_b \times n_b}$. The (i, j)-th entry of $B_{m_b \times n_b}$, denoted by b(i, j) is

$$b(i,j) = -1 \; if \; H_{i,j} = 0_{z \times z} \ldots \text{(3.52)}$$

$$b(i,j) = p \; if \; H_{i,j} = (P_z)^P \ldots \text{(3.53)}$$

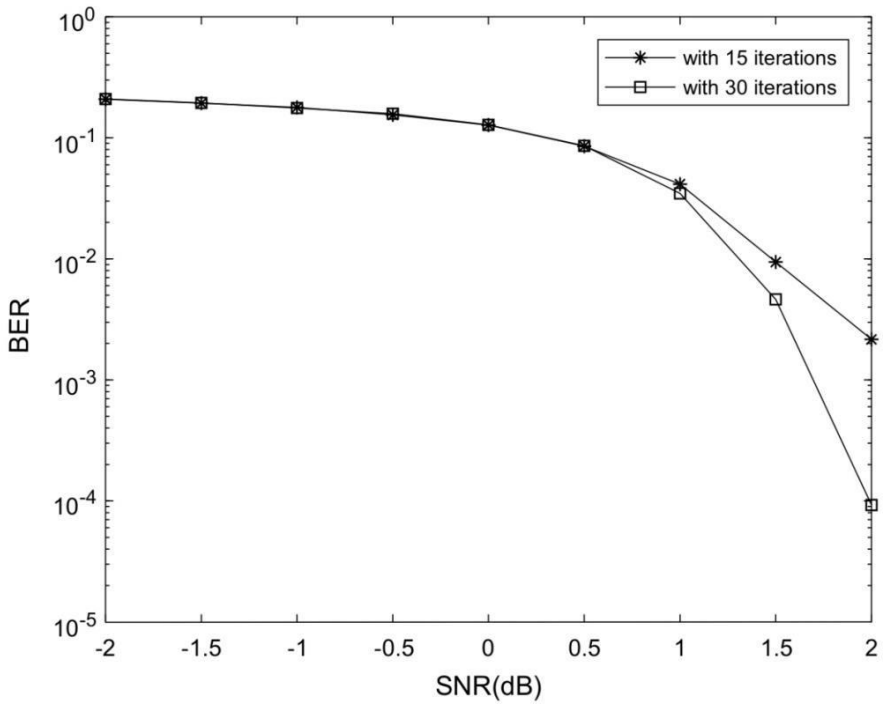
3.9.3 Performance Analysis of QC-LDPC Codes over AWGN Channel

The performance of QC-LDPC codes is illustrated through LDPC-IEEE 802.11n. The following base matrix for $z = 27$ ($rate = 1/2$ $m_b = 12, n_b = 24$) specified in IEEE 802.11n standard is used in this illustration

Base matrix = …

[0 -1 -1 -1 0 0 -1 -1 0 -1 -1 0 1 0 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1
22 0 -1 -1 17 -1 0 0 12 -1 -1 -1 -1 0 0 -1 -1 -1 -1 -1 -1 -1 -1 -1
6 -1 0 -1 10 -1 -1 -1 24 -1 0 -1 -1 -1 0 0 -1 -1 -1 -1 -1 -1 -1 -1
2 -1 -1 0 20 -1 -1 -1 25 0 -1 -1 -1 -1 -1 0 0 -1 -1 -1 -1 -1 -1 -1
23 -1 -1 -1 3 -1 -1 -1 0 -1 9 11 -1 -1 -1 -1 0 0 -1 -1 -1 -1 -1 -1
24 -1 23 1 17 -1 3 -1 10 -1 -1 -1 -1 -1 -1 -1 -1 0 0 -1 -1 -1 -1 -1
25 -1 -1 -1 8 -1 -1 -1 7 18 -1 -1 0 -1 -1 -1 -1 -1 0 0 -1 -1 -1 -1
13 24 -1 -1 0 -1 8 -1 6 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 0 0 -1 -1 -1
7 20 -1 16 22 10 -1 -1 23 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 0 0 -1 -1
11 -1 -1 -1 19 -1 -1 -1 13 -1 3 17 -1 -1 -1 -1 -1 -1 -1 -1 -1 0 0 -1
25 -1 8 -1 23 18 -1 14 9 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 0 0
3 -1 -1 -1 16 -1 -1 2 25 5 -1 -1 1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 0]

The QC-LDPC code is generated using efficient encoding method for LDPC codes. The decoding of QC-LDPC codes is performed based on High-Throughput LDPC Decoders. The following MATLAB Program 8.8 and a function program are written and executed for BER



performance evaluation of QC-LDPC code with the above mentioned base matrix. The SNR versus BER performance with 15 and 30 iterations is shown in Fig. 3.7.

Fig: 3.7 BER performance of QC-LDPC IEEE 802.11n

CHAPTER-4

SYSTEM MODEL

4.1 LDPC ENCODING

LDPC Codes are linear block codes, consists a parity check matrix "H", which is a sparse

i.e. consists more number of zeros (0's) and less number of one's (1's). For an (n, k) LDPC code the size of H matrix was defined as (n-k) ×n where (n-k) is the number of rows; n is the number of columns and k is the number of messages bits. In (n, k) LDPC code w c and w r are considered a weight of the columns (number of 1's in columns) and weight of the rows (number of 1's in rows).

Based on the w c and w r LDPC codes are classified into two types. The number of 1's in each columns and rows are equal then it is called regular LDPC codes otherwise it is called irregular LDPC codes. For a regular code, we have $(n-k)*w_r = n\ w_c$, thus $w_c<w_r$. If all rows are linearly independent, the code rate (R) is; otherwise, it is k/n. Typically, $w_c \geq 3$ at parity check matrix with minimum column weight w_c will have a minimum distance $d_{min} \geq w_c+1$. A parity-check matrix H for an (10, 5) LDPC code with $w_c = 3$ and $w_r = 6$ is shown in Eq 3.1

$$H = \begin{bmatrix} 1\ 1\ 0\ 0\ 1\ 1\ 1\ 1\ 0\ 0 \\ 1\ 0\ 1\ 1\ 0\ 1\ 0\ 1\ 0\ 1 \\ 0\ 1\ 0\ 1\ 1\ 0\ 0\ 1\ 1\ 1 \\ 1\ 0\ 1\ 0\ 1\ 0\ 1\ 0\ 1\ 1 \\ 0\ 1\ 1\ 1\ 0\ 1\ 1\ 0\ 1\ 0 \end{bmatrix} \ldots (4.1)$$

LDPC codes are also represented by Tanner graphs, which are proposed by Tanner in 1981. For an (n, k)parity check matrix H, the number of check nodes C 1 , C 2 , C 3 ,…..C n-k are corresponded to the rows of the parity-check matrix H and bit nodes B 1 , B 2 , B 3 ,…..B n corresponded to the columns of the parity-check matrix H. A Tanner graph for Eq (3.1) is shown in figure (4.1)

The stepwise procedure of LDPC encoding along with an example

Step: 1 Consider a parity check matrix H for an (10, 5) LDPC code.

$$H = \begin{matrix} 1\ 1\ 0\ 0\ 1\ 1\ 1\ 1\ 0\ 0 \\ 1\ 0\ 1\ 1\ 0\ 1\ 0\ 1\ 0\ 1^{1} \\ 0\ 1\ 0\ 1\ 1\ 0\ 0\ 1\ 1\ 1 \\ 1\ 0\ 1\ 0\ 1\ 0\ 1\ 0\ 1\ 1 \\ 0\ 1\ 1\ 1\ 0\ 1\ 1\ 0\ 1\ 0 \end{matrix} \quad \ldots (4.2)$$

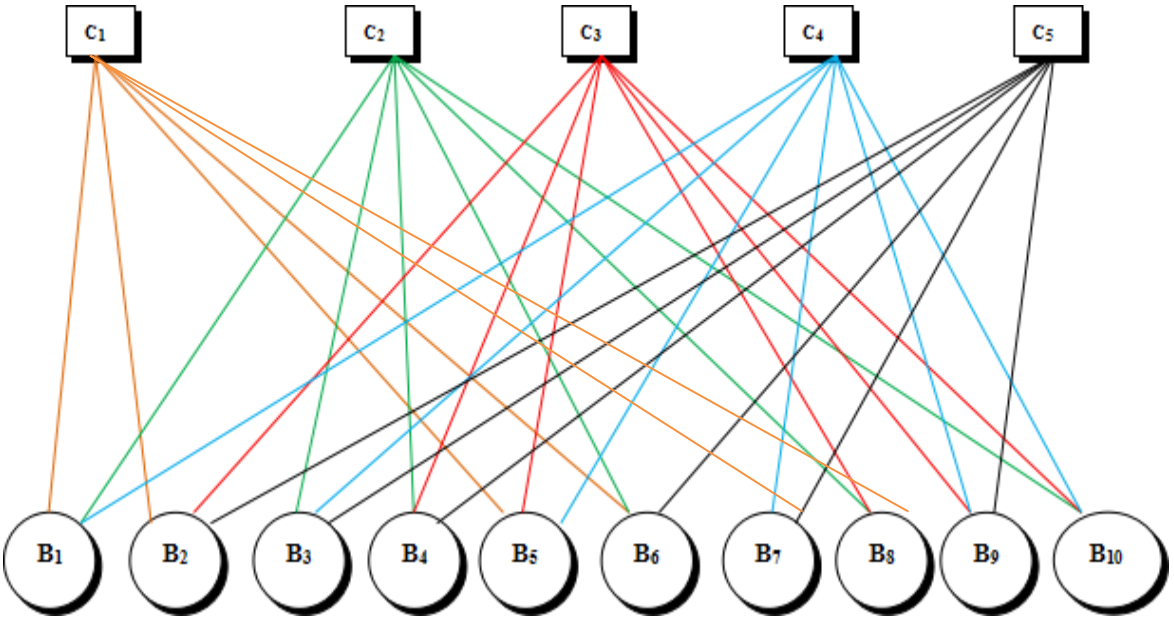


Figure: 4.1 Graphical representation of (10, 5) LDPC code.

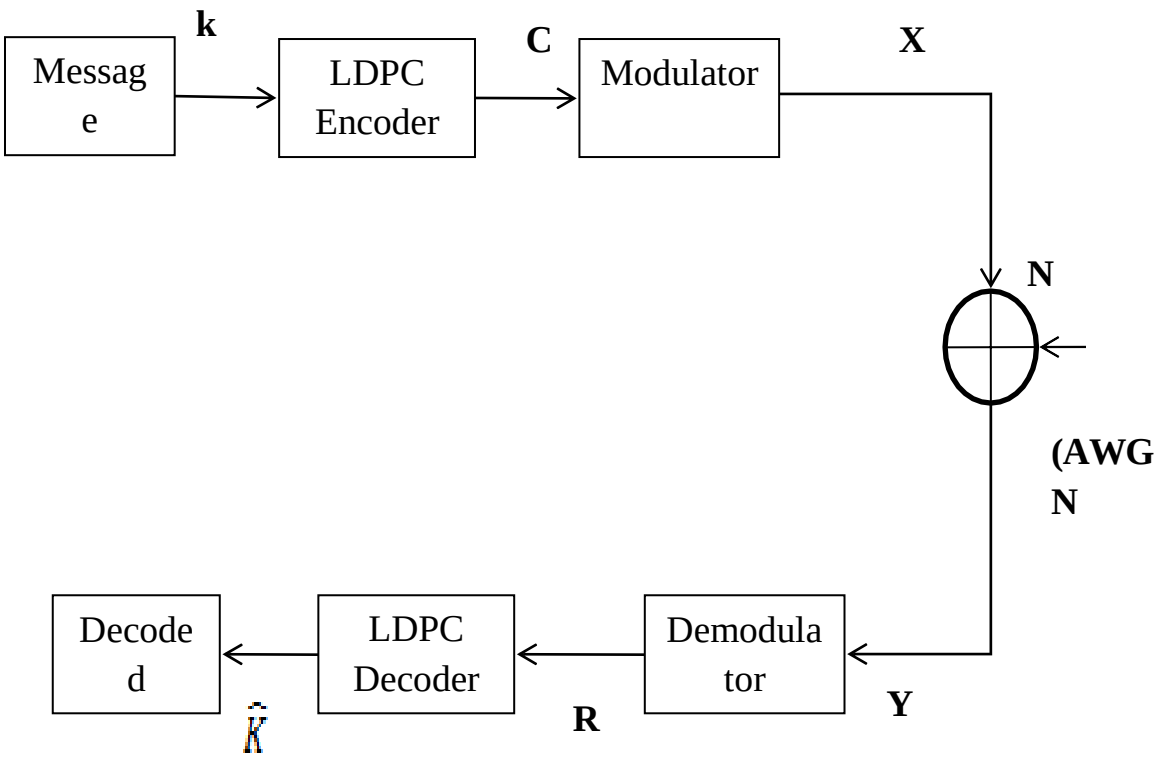


Figure 4.2: Encoding and Decoding System Model for NR-LDPC

Step: 2 Let 'K' be the message signal of size 1×k, K = [1 0 0 0 1] and the code word 'CW' of size 1×n, along with n-k parity bits CW = [1 0 0 0 1 P_1 $P_2P_3P_4P_5$].

For obtaining double diagonal encoding consider (n-k) × (n-k) at the extreme end of the H matrix.

Step: 3 Now by performing row and column permutations on (n-k) × (n-k) matrix, the non-singular parity check matrix H is brought into double diagonal structure. Where (n-k) × (n-k) matrix contains 1's along any two diagonals. Let the resultant matrix be H_d.

$$H_d = \begin{bmatrix} 1\,1\,0\,0\,1\,1\,1\,1\,0\,0 \\ 1\,0\,1\,1\,0\,1\,0\,1\,0\,1 \\ 0\,1\,0\,1\,1\,0\,1\,1\,1\,0 \\ 1\,0\,1\,0\,1\,0\,1\,0\,1\,1 \\ 0\,1\,1\,1\,0\,1\,0\,0\,1\,1 \end{bmatrix} \ldots (4.3)$$

Step: 4 Multiply $H_d * \quad CW^T = 0$ …..(4.4)

On solving the (Eq 4.4) the following expressions are obtained.

$P_1 + P_2 + P_3 = 0$ … (4.5)

$1 + P_1 + P_3 + P_5 = 0$ … (4.6)

$1 + P_2 + P_3 + P_4 = 0$ … (4.7)

$P_2 + P_4 + P_5 = 0$ … (4.8)

$P_1 + P_4 + P_5 = 0$ … (4.9)

By simplifying the above expressions

$P_1 = 0, P_2 = 0, P_3 = 0, P_4 = 1$ and $P_5 = 1$.

Therefore for the message K = [1 0 0 0 1] the LDPC encoded output C= [1 0 0 0 1 0 0 0 11]

4.2 LDPC DECODING:

Bit-Flip decoding algorithm:

The received signals are decoded into 1's and 0's in order to form a vector Y. It computes all check sums and the number of unfulfilled parity checks involving each of the n bits of the vector Y in each and every iteration. The bits of Y are then flipped .The procedure must be performed until all check sums have been satisfied or a predefined number of iterations has been reached. The Bit-flipping decoding algorithm's step-by-step approach is as follows:

Step 1. Let N = Y, defining Cj in order to represent the jth parity check equation of S

Step 2. K= 0

Step 3. Compute all check sums by the expression

$$E_{j,i} = \sum_{i' \in c_{j,i' \neq i}} (N_{i'} \; mod \; 2) \quad \ldots.(4.1)$$

Step 4. Check the number of unsatisfied parity check bits.

Step 5. Flip the bits of message when they are large number of unsatisfied parity checks. The flipping on i th bit can be performed by

$$N_i = (y_i + 1 \bmod 2) \quad \ldots(4.2)$$

Step 6. Compute s as follows

$$s = (NS^T) mod 2 \qquad \ldots(4.3)$$

Step 7. If s = 0 or K = K max, stop, else

Step 8.K= K+ 1, go to Step 3

Log Domain Sum-Product algorithm:

Step1. Initialize i and j for which $M_{ji} = L_i$, and l =0. Define Bj and Ai in order to set the bits in jth and ith parity check matrix S respectively.

Step2.

$$M\_ji = \alpha\_ji\, \beta\_ji \qquad \ldots.(4.4)$$

$$\alpha_{ji} = \text{sign}(M_{ji}) \ldots..(4.5)$$

$$\beta_{ji} = |M_{ji}| \qquad \ldots.(4.6)$$

$$E_{ji} = \prod_{i'} \alpha_{ji'} \; . \emptyset \left(\sum_{i'} \emptyset \left(\beta_{ji'} \right) \right) \qquad \ldots(4.7)$$

$$\emptyset(x) = -\log\left[\tanh\left(\frac{x}{2}\right)\right] = \log\left(\frac{e^x+1}{e^x-1}\right) \qquad \ldots(4.8)$$

Step 3. Calculate LLR total For i = 0, 1,..., N − 1

$$L_i^{total} = L_i + \sum_{j \in A_i} E_{ji} \qquad \ldots(4.9)$$

Step 4. In order to Stop the criteria For i = 0, 1,..., N − 1, set

$$\hat{C}_i = \begin{cases} 1 \text{ if } L_i^{total} < 0, \\ 0 \text{ else,} \end{cases} \qquad \ldots(4.10)$$

To obtain $\hat{c}$. If $\hat{c}H^T = 0$ or the number of iterations equals the maximum limit (l = l max,) stop;

else

Step 5. calculate

$$M_{ji} = L_i + \sum_{j' \in A_{j,i' \neq ji}} E_{j'i} . \; l = l + 1 \qquad \ldots(4.11)$$

Min-Sum decoding algorithm:

Step1: The shape of φ(x) that the largest term in the sum corresponds to the smallest βji and it is assumed that it dominates the sum. Thus the following relation defined.

$$\emptyset\left(\sum_{i'}\emptyset\left(\beta_{ji'}\right)\right) \simeq \emptyset\left(\emptyset\left(\min_{i'}\beta_{ji'}\right)\right) = \min_{i'}\beta_{ji'} \quad \text{...(4.12)}$$

Thus, the Min-Sum algorithm is same as the log domain sum product algorithm with Step 2 replaced by

$$M_{ji} = \alpha_{ji}\beta_{ji} \quad \text{...(4.13)}$$

$$\alpha_{ji} = \text{sign}\left(M_{ji}\right) \quad \text{…(4.14)}$$

$$\beta_{ji} = \left|M_{ji}\right| \quad \text{…(4.15)}$$

$$E_{ji} = \prod_{i'}\alpha_{ji'}.\min_{i'}\beta_{ji'} \quad \text{…(4.16)}$$

CHAPTER-5

MATLAB

5.1 MATLAB Introduction

MATLAB is a high performance language for technical computing. It integratescomputation visualization and programming in an easy to use environment. MATLABstands for matrix laboratory. It was written originally to provide easy access to matrix software developed by LINPACK (linear system package) and EISPACK (Eigen system package) projects. MATLAB is therefore built on a foundation of sophisticated matrix software in which the basic element is matrix that does not require pre dimensioning.

Typical uses of MATLAB

1. Math and computation
2. Algorithm development
3. Data acquisition
4. Data analysis, exploration and visualization
5. Scientific and engineering graphics

The main features of MATLAB

1. Advanced algorithm for high performance numerical computation, especially in the Field matrix algebra
2. A large collection of predefined mathematical functions and the ability to define one's own functions.
3. Two-and three dimensional graphics for plotting and displaying data
4. A complete online help system
5. Powerful matrix or vector oriented high level programming language for individual applications.
6. Toolboxes available for solving advanced problems in several application areas.

5.2 The MATLAB System

The MATLAB System consists of five main parts

- Development Environment:

This is the set of tools and facilities that help you use MATLAB functions and files. Many of these tools are graphical user interfaces. It includes the MATLAB desktop and Command Window, command history an editor and debugger, and browsers for viewing help the workspace, files, and the search path.

- The MATLAB Mathematical Function Library:

This is a vast collection of computational algorithms ranging from elementary functions, like sum sine, cosine, and complex arithmetic, to more sophisticated functions like matrix inverse, matrix Eigen values, Bessel functions, and fast Fourier transforms.

- The MATLAB Language:

This is a high-level matrix/array language with control flow statements, functions, data structures, input/output, and object-oriented programming features. It allows both programming in the small to rapidly create quickly programs, and "programming in the large" to create large and complex application programs.

- Graphics:

MATLAB has extensive facilities for displaying vectors and matrices as graphs, as well as annotating and printing these graphs. It includes high-level functions for two-dimensional and three dimensional data visualization, video processing, animation, and presentationgraphics. It also includes low-level functions that allow you to fully customize the appearance of graphics as well as to build complete graphical user interfaces on your**MATLAB applications**

- The MATLAB Application Program Interface (API):

This is a library that allows you to write C and Fortran programs that interact with MATLAB. It includes facilities for calling routines from MATLAB (dynamic linking), calling MATLAB as a computational engine, and for reading and writing MAT-files

5.3 Starting MATLAB:

On Windows platforms, start MATLAB by double-clicking the MATLAB shortcut icon on your Windows desktop. On UNIX platforms, start MATLAB by typing mat lab at the operating system prompt. You can customize MATLAB start-up.

For example, you can change the directory in which MATLAB starts or automatically execute MATLAB statements in a script file named start-ups.

5.4 MATLAB Desktop:

When you start MATLAB, the MATLAB desktop appears, containing tools (graphical user interfaces) for managing files, variables, and applications associated with MATLAB. The following illustration shows the default desktop. You can customize the arrangement of tools and documents to suit your needs.

5.5 MATLAB Working Environment

- MATLAB Desktop:

MATLAB Desktop is the main MATLAB application window. The desktop contains five sub windows the command window, the workspace browser the current directory window, the command history window, and one or more figure windows, which are shown only whenthe user displays a graphic.

The command window is where the user types MATLAB commands and expressions at the prompt (>>) and where the output of those commands is displayed.MATLAB defines as the workspace as the set of variables that the user creates in a work session. The workspace browser shows these variables and some information about them. Double clicking on a variable in the workspace browser launches the Array Editor, whichcan be used to obtain information and income instances edit certain properties of thevariable.

The current Directory tab above the workspace tab shows the contents of the current directory, whose path is shown in the current directory window. For example, in the windows operating system the path might be as follows: C-MATLAB Work, indicating that directory "work" is a subdirectory of the main directory MATLAB WHICH ISINSTALLED IN DRIVE C. clicking on the arrow in the current directory window shows a list of recently used paths. Clicking on the button to the right of the window allows the user to change the current directory.

MATLAB uses a search path to find M-files and other MATLAB related files, which are organize in directories in the computer file system. Any file run in MATLAB must reside in the current directory or in a directory that is on search path. By default, the files supplied with MATLAB and math works toolboxes are included in the search path. The easiest way to see which directories is on the search path. The easiest way to see which directories are soon the search paths, or to add or modify a search path, is to select set path from the File menu the desktop, and then use the set path dialog box. It is good practice to add any commonly used directories to the search path to avoid repeatedly having the change the current directory.

The Command History Window contains a record of the command window, including both current and previous MATLAB sessions. Previously entered MATLAB commands can be selected and re-executed from the command History window by right clicking on a command or sequence of commands. This action launches a menu from which to select various options in addition to executing the commands. This is a use to select

various options in addition to executing the commands. This is useful feature when experimenting with various commands in a work session.

- Using the MATLAB Editor to create M-Files:

The MATLAB editor is both a text editor specialized for creating M-files and graphical MATLAB debugger. The editor can appear in a window by itself, or it can be a sub window in the desktop, M-files are denoted by the extension m The MATLAB editor window has numerous pull-down menus for tasks such as saving viewing, and debugging files. Because it performs some simple checks and also uses color to differentiate between various elements of code, this text editor is recommended as the tool of choice for writing and editing M functions. To open the editor, type edit at the prompt opens the M-file filenames in an editor window ready for editing. As noted earlier the file must be in the current directory, or in a directory in the search path

- Getting Help:

The principle way to get help online is lo use the MATLAB help browser, opened as a separate window either by clicking on the question mark symbol (?) on the desktop toolbar, or by typing help browser at the prompt in the command window. The help Browser is aweb browser integrated into the MATLAB desktop that displays a Hypertext Markup Language (HTML) documents. The Help Browser consists of two panes, the help navigator pane, used to find information, and the display pane, used to view the information. Self- explanatory tabs other navigator pane are used to perform a scratch.

For example, help on a specific function is obtained by selecting the search tab, selecting Function Name as the Search Type, and then typing in the function name in the Search for field. It is good practice to open the Help Browser at the beginning of a MATLAB session to have help readily available during code development or other MATLAB task.

Another way to obtain for a specific function is by typing doc followed by the function name at the command prompt. For example, typing doc format displays documentation for the function called format in the display pane of the Help Browser. This command opens the browser if it is not already open.

M-functions have two types of information that can be displayed by the user The first is called the HI line, which contains the function name and alone line description the second is a block of explanation called the Help text block. Typing help at the prompt followed by a function name displays both the HI line and the Help text for that function in there command window.

Typically look for followed by a keyword displays all the HI lines that contain that keyword. This function is useful when looking for a particular topic without knowing the names of applicable functions. For example, typing look for edge at the prompt displays the HI lines containing that keyword. Because the HI line contains the function name, it then becomes possible to look at specific functions using the other help methods. Typing look for edge-all at the prompt displays the HI line of all functions that contain the word edge in cither the HI line or the Help text block. Words that contain the characters edge also are detected. For example, the HI line of a function containing the word poly edge in the H1 lineor Help text would also be displayed.

5.6 Saving and Retrieving a Work Session

There are several ways to save and load an entire work session or selected workspace variables in MATLAB. The simplest is as follows.

To save the entire workspace, simply right-click on any blank space in the workspace Browser window and select Save Workspace as from the menu that appears. Thisopens a directory window that allows naming the file and selecting any folder in the system in which to save it. Then simply click Save To save a selected variable from the workspace, select the variable with a left click and then right-click on the highlighted area. Then select Save Selection As from the menu that appears. This again opens a window from which a folder can be selected to save the variable.

To select multiple variables, use shift click or control click in the familiar manner, and then use the procedure just described for a single variable. All files are saved in the double-precision, binary format with the extension mat. These saved files commonly are referred to as MAT-files. For example, a session named, says mywork_2012-02-10, and would appear as the MAT-file mywork_2012_02_10.mat when saved. Similarly, a saved video called final video will appear when saved as final_ video.mat.

To load saved workspaces and/or variables, left-click on the folder icon on the toolbar of the workspace Browser window. This causes a window to open from which a folder containing MAT-file or selecting open causes the contents of the file to be restored in the workspace Browser window. It is possible to achieve the same results described in the preceding paragraphs by typing save and load at the prompt with the appropriate file names and path information. This approach is not as convenient, but it is used when formats other than those available in the menu method are required.

- Graph Components:

MATLAB displays graphs in a special window known as a figure. To create a graph, you need to define a coordinate system. Therefore every graph is placed within Axes, which are contained by the figure. The actual visual representation of the data is achieved with graphics objects like lines and surfaces. These objects are drawn within the coordinate system defined by the axes, which MATLAB automatically creates specifically to accommodate the range of the data. The actual data is stored as properties of the graphics objects.

- Plotting Tools

Plotting tools are attached to figures and create an environment for creating Graphs.

These tools enable you to do the following:

• Select from a wide variety of graph types

• Change the type of graph that represents a variable

• See and set the properties of graphics objects

• Annotate graphs with text, arrows, etc.

• Drag and drop data into graphs

Display the plotting tools from the View menu or by clicking the plotting tools

icon in the figure toolbar, as shown in the following picture.

- Editor/Debugger

Use the Editor/Debugger to create and debug M-files, which are programs you write to run MATLAB functions. The Editor/Debugger provides a graphical user interface for text editing, as well as for M-file debugging.

To create or edit an M-file use File > New or File > Open, or use the edit function.

CHAPTER-6

SIMULATION RESULTS

This project focuses on reducing the Bit error rate and Frame error rate using LDPC decoding techniques and comparing the bit error rate and frame characteristics between the proposed algorithms (i.e. Bit flip, Min-sum, Normal min-sum, Mod min-sum and Log-sum product). MATLAB 2021a is used to analyze decoding algorithms, with the following parameters. Simulation results shows bit error rate and frame error rate performances and how the BER, FER characteristics of LDPC decoding algorithms are varied based on different parameters (i.e. Number of iterations, Expansion Factor, Number of Blocks)

MATLAB 2021a is used to analyze decoding algorithms, with the following parameters.

Parameter	**Specifications**
Decoding Methods	Min-sum, Mod Min-sum, Log Sum-Product, Bit Flip, Normal Min-Sum
Number of iterations	10,20
Expansion factor	2,32,64
Number of blocks	10
SNR range	1-3dB

BIT ERROR RATE:

Table-1: Number of iterations =10, Expansion factor = 2;

Eb/No(SNR)	**Bit-Flip**	Normal Min Sum	Min Sum	Mod Min Sum	LogSum-Product
1	0.3677	0.0744141	0.10723	0.0761719	0.0841797
1.3	0.3650	0.0675451	0.09845	0.0675682	0.0758353
1.5	0.356	0.0562305	0.74941	0.0517969	0.0605664
1.7	0.3540	0.0415678	0.05678	0.0384692	0.0456382
2	0.3531	0.0312109	0.0396	0.0247005	0.0295247
2.3	0.3470	0.0278901	0.02345	0.0164839	0.0178362
2.5	0.3465	0.0120182	0.01225	0.00671387	0.00897624
2.7	0.3367	0.01100012	0.011789	0.00549278	0.0063829
3	0.3246	0.00200195	0.00083	0.000390625	0.000683594

Table-2: Number of iterations =10, Expansion factor =32;

Eb/No(SNR)	**Bit-Flip**	Normal Min Sum	Min Sum	Mod Min Sum	LogSum-Product
1	1	0.0720703	0.10313	0.072073	0.0845703
1.3	1	0.0639821	0.09638	0.069263	0.0783992
1.5	1	0.0538086	0.07311	0.0484375	0.0577734
1.7	1	0.0474281	0.05219	0.037289	0.0473646
2	1	0.0284675	0.0362	0.0204622	0.0266146
2.3	1	0.0174531	0.02849	0.018273	0.0189238
2.5	1	0.00951986	0.0096	0.00459635	0.0068180
2.7	1	0.0073856	0.00735	0.002789	0.0058909
3	1	0.00107422	0.00068	0.000195313	0.000292969

Table-3: Number of iterations =10, Expansion factor = 64;

Eb/No(SNR)	**Bit-Flip**	Normal Min Sum	Min Sum	Mod Min Sum	LogSum-Product
1	1	0.0716797	0.10801	0.0716797	0.0873047
1.3	1	0.0637282	0.09378	0.0683652	0.0783654
1.5	1	0.0521094	0.07127	0.0451742	0.0549414
1.7	1	0.0371894	0.05356	0.0323891	0.0367181
2	1	0.0263281	0.03407	0.0176888	0.0240104
2.3	1	0.0156782	0.02671	0.0091518	0.0167361
2.5	1	0.0077262	0.00844	0.0033414	0.005585594
2.7	1	0.0057832	0.00645	0.0021781	0.00467284
3	1	0.000781125	0.00034	4.8805	0.000146484

Table-4: Number of iterations =20, Expansion factor = 2;

Eb/No(SNR)	**Bit-Flip**	Normal Min Sum	Min Sum	Mod Min Sum	LogSum-Product
1	1	0.0677734	0.10879	0.0603516	0.0734375
1.3	1	0.0583741	0.09161	0.0503451	0.0627931
1.5	1	0.0456445	0.06768	0.0353906	0.0496289
1.7	1	0.0215673	0.05382	0.0217389	0.0283672
2	1	0.0192708	0.02045	0.00880859	0.0179427
2.3	1	0.0022623	0.01734	0.0056835	0.0093567
2.5	1	0.00341471	0.00566	0.000727539	0.00287109
2.7	1	0	0	0	0
3	1	0	0	0	0

Table-5: Number of iterations =20, Expansion factor = 32;

Eb/No(SNR)	**Bit-Flip**	Normal Min Sum	Min Sum	Mod Min Sum	LogSum-Product
1	1	0.97	0.91	0.87	0.78
1.3	1	0.91	0.85	0.78	0.65
1.5	1	0.81	0.69	0.57	0.47
1.7	1	0.65	0.59	0.38	0.26
2	1	0.44	0.36	0.27	1.36667
2.3	1	0.33	0.23	0.19	0.06732
2.5	1	0.109167	0.08417	0.0483333	0.0141667
2.7	1	0	0	0	0
3	1	0	0	0	0

Table-6: Number of iterations =20, Expansion factor = 64;

Eb/No (SNR)	**Bit-Flip**	Normal Min Sum	Min Sum	Mod Min Sum	LogSum - Product
1	1	0.98	0.95	0.83	0.75
1.3	1	0.88	0.84	0.75	0.63
1.5	1	0.77	0.65	0.52	0.41
1.7	1	0.64	0.42	0.44	0.33
2	1	0.40	0.33	0.21	0.11
2.3	1	0.35	0.21	0.10	0.00915
2.5	1	0.089	0.0725	0.0375	0.0091
2.7	1	0	0	0	0
3	1	0	0	0	0

FRAME ERROR RATE:

Table-7: Number of iterations =10, Expansion factor = 2;

Eb/No(SNR)	**Bit-Flip**	Normal Min Sum	Min Sum	Mod Min Sum	LogSum-Product
1	1	1	1	1	1
1.3	1	1	1	1	1
1.5	1	1	1	0.99	0.98
1.7	1	0.9734	0.97	0.96	0.95
2	1	0.93333	0.88	0.826667	0.803333
2.3	1	0.86	0.75	0.784	0.7376
2.5	1	0.705	0.57417	0.485833	0.4225
2.7	1	0.567	0.345	0.2831	0.289
3	1	0.3	0.05	0.05	0.05

Table-8: Number of iterations =10, Expansion factor = 32;

Eb/No(SNR)	**Bit-Flip**	Normal Min Sum	Min Sum	Mod Min Sum	LogSum-Product
1	1	1	1	1	1
1.3	1	0.99	0.99	0.98	0.97
1.5	1	0.99	0.97	0.96	0.93
1.7	1	0.90	0.87	0.89	0.78
2	1	0.88	0.78333	0.72	0.65667
2.3	1	0.79	0.58	0.65	0.48
2.5	1	0.575833	0.43	0.3475	0.279167
2.7	1	0.37	0.36	0.19	0.19
3	1	0.15	0.05	0.05	0.05

Table-9: Number of iterations =10, Expansion factor = 64;

Eb/No(SNR)	**Bit-Flip**	Normal Min Sum	Min Sum	Mod Min Sum	LogSum-Product
1	1	1	1	1	0.9
1.3	1	0.99	0.98	0.99	0.89
1.5	1	0.99	0.97	0.9	0.88
1.7	1	0.89	0.86	0.87	0.78
2	1	0.83	0.72	0.61	0.56
2.3	1	0.68	0.56	0.54	0.37
2.5	1	0.48	0.33	0.25	0.19
2.7	1	0.28	0.18	0.10	0.09
3	1	0.12	0.05	0.05	0.025

Table-10: Number of iterations =20, Expansion factor = 32;

Eb/No(SNR)	**Bit-Flip**	Normal Min Sum	Min Sum	Mod Min Sum	LogSum-Product
1	1	0.9	0.9	0.8	0.8
1.3	1	0.89	0.89	0.71	0.71
1.5	1	0.84	0.78	0.63	0.52
1.7	1	0.79	0.67	0.59	0.45
2	1	0.54333	0.41667	0.326667	0.2
2.3	1	0.38	0.27	0.29	0.10
2.5	1	0.156667	0.10417	0.733333	0.0275
2.7	1	0	0	0	0
3	1	0	0	0	0

Table-11: Number of iterations =20, Expansion factor = 32;

Eb/No(SNR)	**Bit-Flip**	Normal Min Sum	Min Sum	Mod Min Sum	LogSum-Product
1	1	0.9	0.9	0.8	0.7
1.3	1	0.87	0.79	0.79	0.69
1.5	1	0.81	0.69	0.57	0.47
1.7	1	0.68	0.46	0.49	0.35
2	1	0.44	0.36	0.27	0.13666
2.3	1	0.22	0.28	0.10	0.09856
2.5	1	0.109167	0.08472	0.0483333	0.0141667
2.7	1	0	0	0	0
3	1	0	0	0	0

Table-12: Number of iterations =10, Expansion factor = 64;

Eb/No(SNR)	**Bit-Flip**	Normal Min Sum	Min Sum	Mod Min Sum	LogSum-Product
1	1	0.9	0.9	0.8	0.7
1.3	1	0,86	0.89	0.79	0.54
1.5	1	0.77	0.65	0.52	0.41
1.7	1	0,58	0.45	0.43	0.28
2	1	0.40	0.33	0.21	0.11
2.3	1	0.35	0.10	0.17	0.00945
2.5	1	0.08	0.0725	0.03	0.00916
2.7	1	0	0	0	0
3	0	0	0	0	0

Table:1 demonstrates BER performance of different LDPC decoding algorithms like Mod Min-Sum, Normal Min-Sum, Min-Sum, Bit Flip and Log Sum-Product algorithms for 10 number of iterations and with expansion factor 2. As the SNR increases from 1 to 3,the BER gradually decreasing. For example if we consider Bit flip decoding algorithm it is decreasing from 0.3677 to 0.3246.

Table:2 demonstrates BER performance of different LDPC decoding algorithms like Mod Min-Sum, Normal Min-Sum, Min-Sum, Bit Flip and Log Sum-Product algorithms for 10 number of iterations and with expansion factor 32. As the SNR increases from 1 to 3, the BER gradually decreasing for all other decoding algorithms except Bit flip decoding algorithm which maintains a constant BER. For example if we consider Min-Sum decoding algorithm it is decreasing from 0.10313to0.00068.

Table:3 demonstrates BER performance of different LDPC decoding algorithms like Mod Min-Sum, Normal Min-Sum, Min-Sum, Bit Flip and Log Sum-Product algorithms for 10 number of iterations and with expansion factor 64. As the SNR increases from 1 to 3, the BER gradually decreasing for all other decoding algorithms except Bit flip decoding algorithm which maintains a constant BER. For example if we consider Min-Sum decoding algorithm it is decreasing from 0.10801to 0.00034.

Table:4 demonstrates BER performance of different LDPC decoding algorithms like Mod Min-Sum, Normal Min-Sum, Min-Sum, Bit Flip and Log Sum-Product algorithms for 20 number of iterations and with expansion factor 2. As the SNR increases from 1 to 3, the BER gradually decreasing for all other decoding algorithms except Bit flip decoding algorithm which maintains a constant BER. For example if we consider Min-Sum decoding algorithm it is decreasing from 0.10879to 0.

Table:5 demonstrates BER performance of different LDPC decoding algorithms like Mod Min-Sum, Normal Min-Sum, Min-Sum, Bit Flip and Log Sum-Product algorithms for 20 number of iterations and with expansion factor 32. As the SNR increases from 1 to 3, the BER gradually decreasing for all other decoding algorithms except Bit flip decoding algorithm which maintains a constant BER. For example if we consider Min-Sum decoding algorithm it is decreasing from 0.91to 0.

Table:6 demonstrates BER performance of different LDPC decoding algorithms like Mod Min-Sum, Normal Min-Sum, Min-Sum, Bit Flip and Log Sum-Product algorithms for 20 number of iterations and with expansion factor 64. As the SNR increases from1 to 3, the BER gradually decreasing for all other decoding algorithms except Bit flip decoding algorithm which maintains a constant BER. For example if we consider Min-Sum decoding algorithm it is decreasing from 0.95to 0.

Table:7 demonstrates FER performance of different LDPC decoding algorithms like Mod Min-Sum, Normal Min-Sum, Min-Sum, Bit Flip and Log Sum-Product algorithms for 10 number of iterations and with expansion factor 2. As the SNR increases from 1 to 3, the FER gradually decreasing for all other decoding algorithms except Bit flip decoding algorithm which maintains a constant FER. For example if we consider Min-Sum decoding algorithm it is decreasing from 1 to 0.05.

Table:8 demonstrates FER performance of different LDPC decoding algorithms like Mod Min-Sum, Normal Min-Sum, Min-Sum, Bit Flip and Log Sum-Product algorithms for 10 number of iterations and with expansion factor 32. As the SNR increases from 1 to 3, the FER gradually decreasing for all other decoding algorithms except Bit flip decoding algorithm which maintains a constant FER. For example if we consider Min-Sum decoding algorithm it is decreasing from 1to 0.05.

Table:9 demonstrates FER performance of different LDPC decoding algorithms like Mod Min-Sum, Normal Min-Sum, Min-Sum, Bit Flip and Log Sum-Product algorithms for 10 number of iterations and with expansion factor 64. As the SNR increases from 1 to 3, the FER gradually decreasing for all other decoding algorithms except Bit flip decoding algorithm which maintains a constant FER. For example if we consider Min-Sum decoding algorithm it is decreasing from 1to 0.05.

Table:10 demonstrates FER performance of different LDPC decoding algorithms like Mod Min-Sum, Normal Min-Sum, Min-Sum, Bit Flip and Log Sum-Product algorithms for 20 number of iterations and with expansion factor 2. As the SNR increases from 1 to 3, the FER gradually decreasing for all other decoding algorithms except Bit flip decoding algorithm which maintains a constant FER. For example if we consider Min-Sum decoding algorithm it is decreasing from 0.9to 0.

Table:11 demonstrates FER performance of different LDPC decoding algorithms like Mod Min-Sum, Normal Min-Sum, Min-Sum, Bit Flip and Log Sum-Product algorithms for 20 number of iterations and with expansion factor 32. As the SNR increases from 1 to 3, the FER gradually decreasing for all other decoding algorithms except Bit flip decoding algorithm which maintains a constant FER. For example if we consider Min-Sum decoding algorithm it is decreasing from 0.9to 0.

Table:12 demonstrates FER performance of different LDPC decoding algorithms like Mod Min-Sum, Normal Min-Sum, Min-Sum, Bit Flip and Log Sum-Product algorithms for 20 number of iterations and with expansion factor 64. As the SNR increases from 1 to 3, the FER gradually decreasing for all other decoding algorithms except Bit flip decoding algorithm which maintains a constant FER. For example if we consider Min-Sum decoding algorithm it is decreasing from 0.9to 0.

Bit Error Rate:

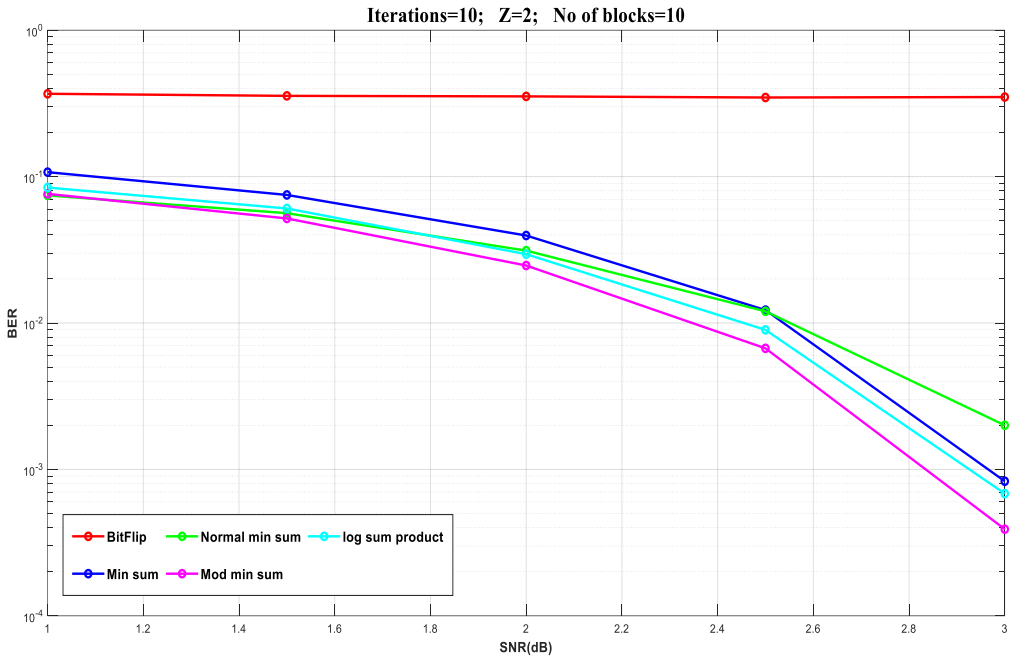


Figure 1: BER of different LDPC decoding algorithms with number of iterations=10, number of blocks=10 and Expansion factor=2.

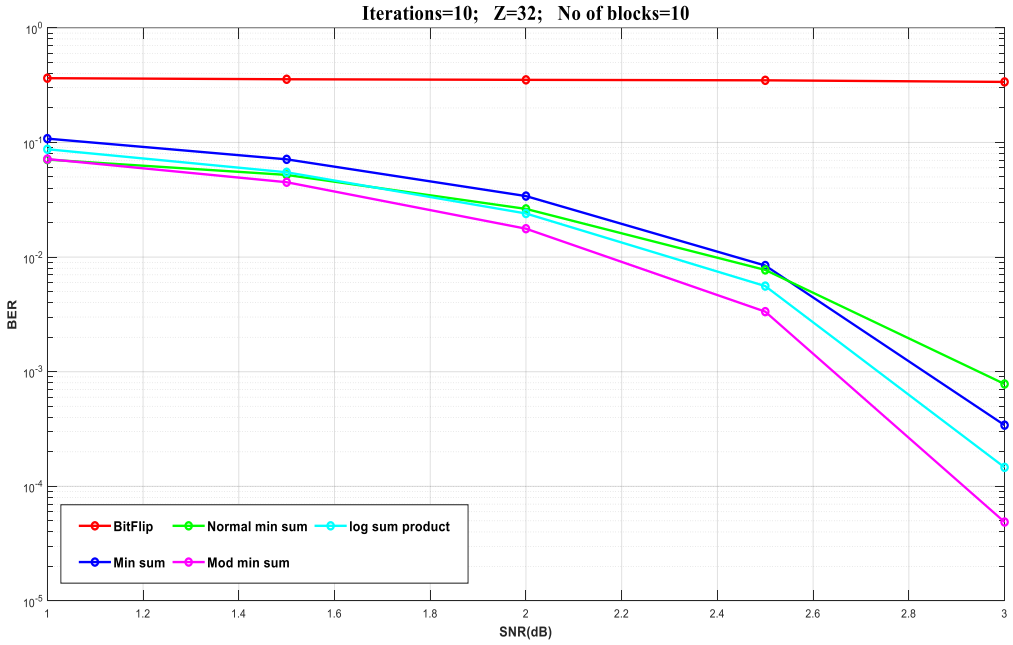


Figure 2: BER of different LDPC decoding algorithms with number of iterations=10, number of blocks=10 and Expansion factor=32.

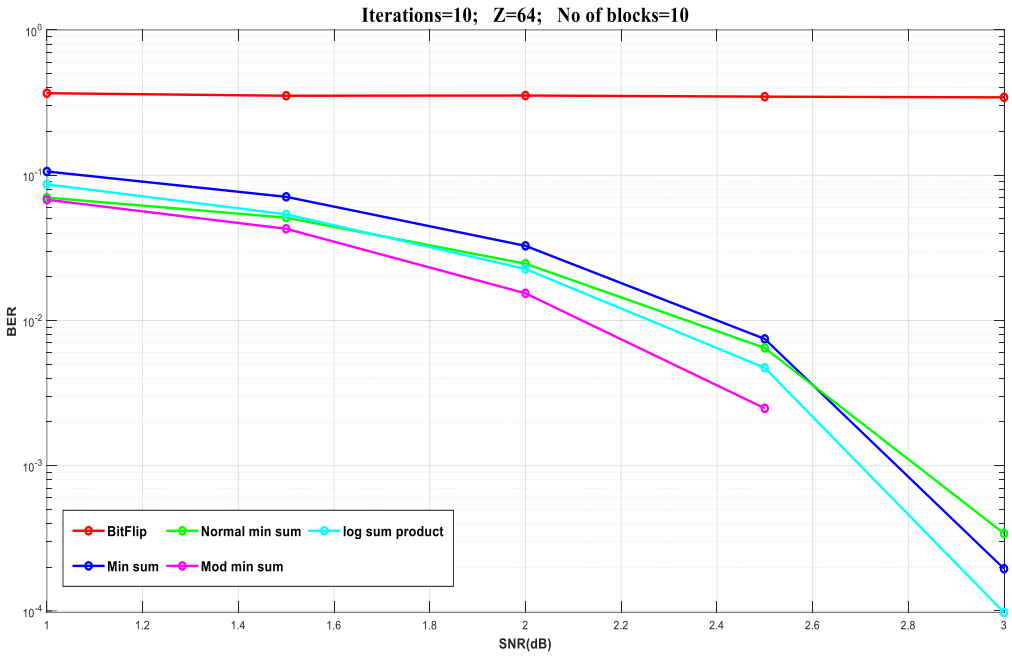


Figure 3: BER of different LDPC decoding algorithms with number of iterations=10, number of blocks=10 and Expansion factor=64.

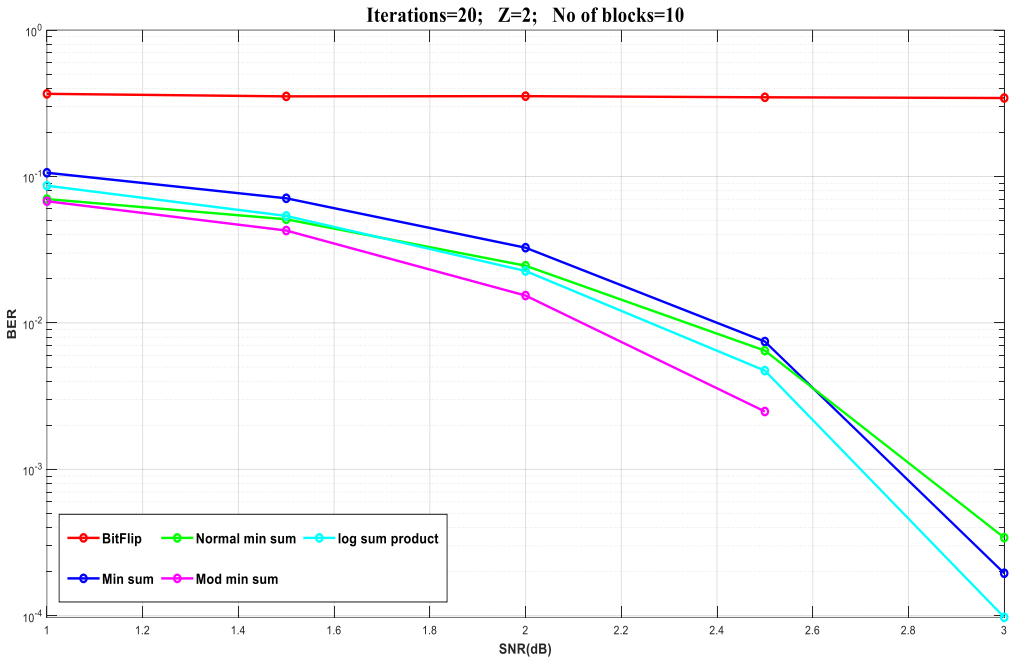


Figure 4: BER of different LDPC decoding algorithms with number of iterations=20, number of blocks=10 and Expansion factor=2.

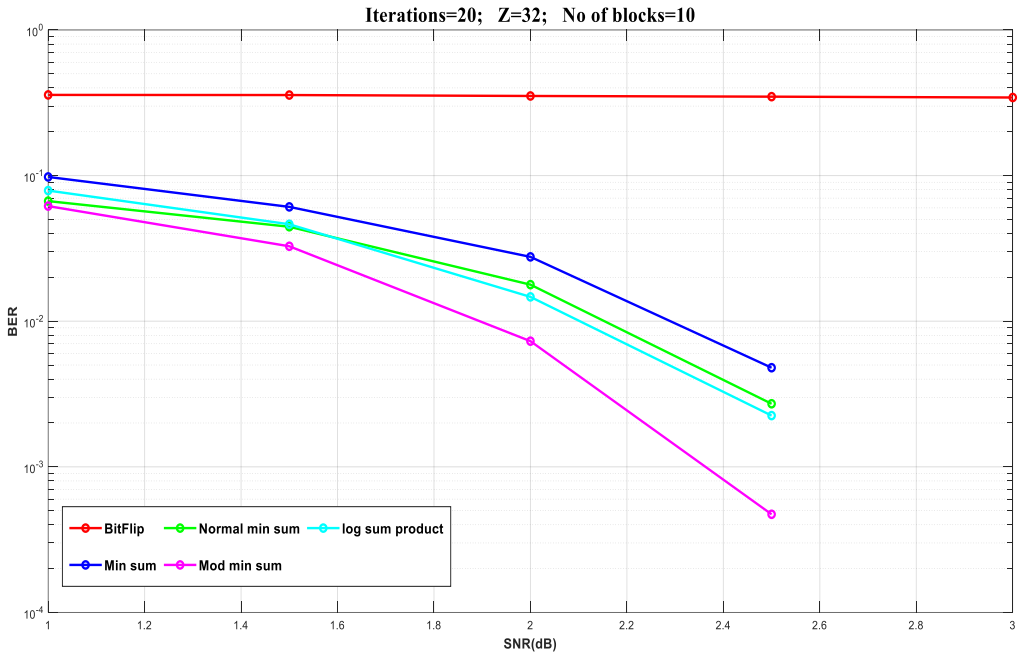


Figure 5: BER of different LDPC decoding algorithms with number of iterations=20, number of blocks=10 and Expansion factor=32.

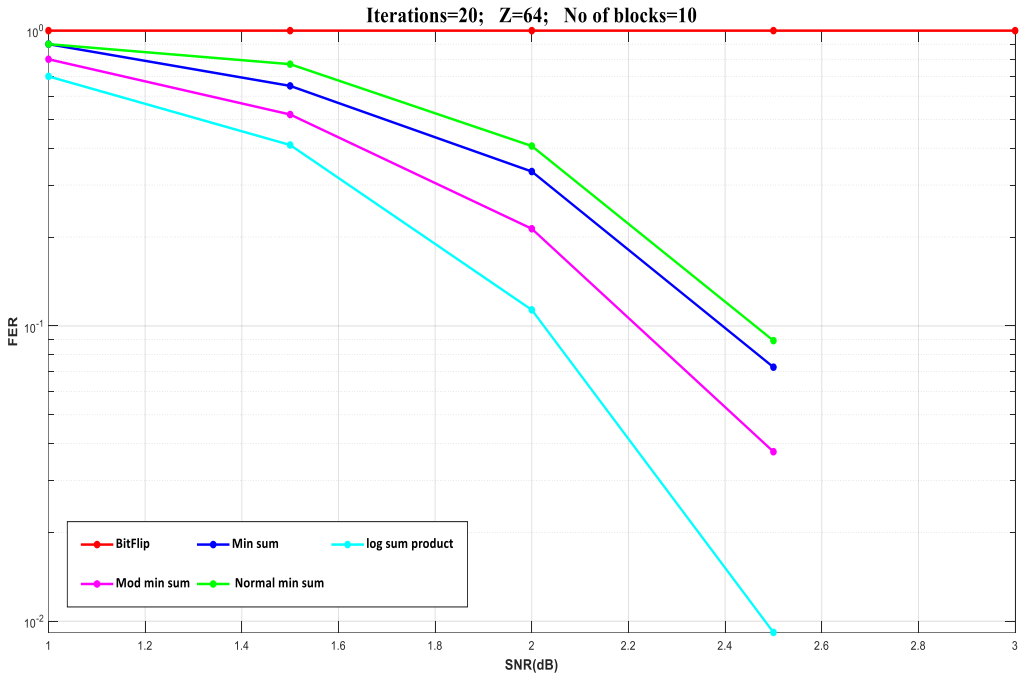


Figure 6: BER of different LDPC decoding algorithms with number of iterations=20, number of blocks=10 and Expansion factor=64.

Frame Error Rate:

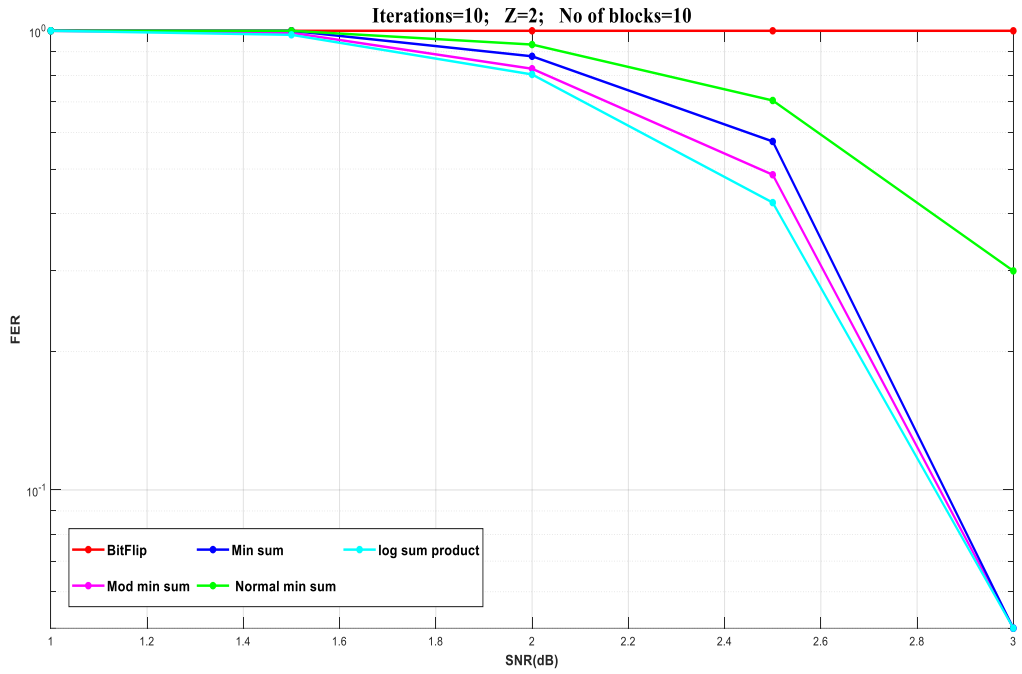


Figure 7: FER of different LDPC decoding algorithms with number of iterations=10, number of blocks=10 and Expansion factor=2.

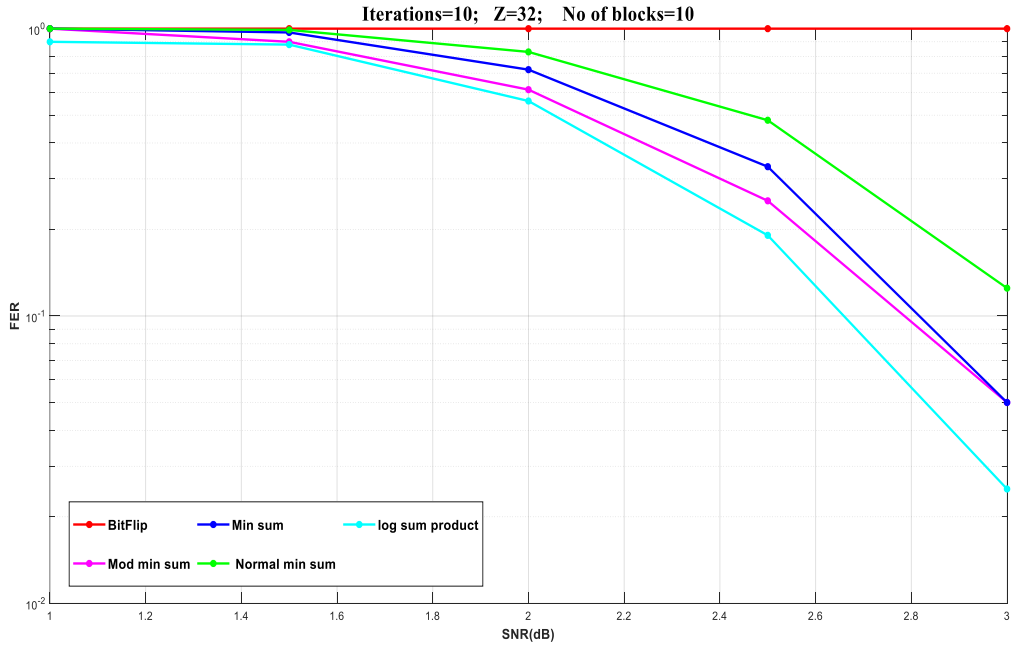


Figure 8: FER of different LDPC decoding algorithms with number of iterations=10, number of blocks=10 and Expansion factor=32.

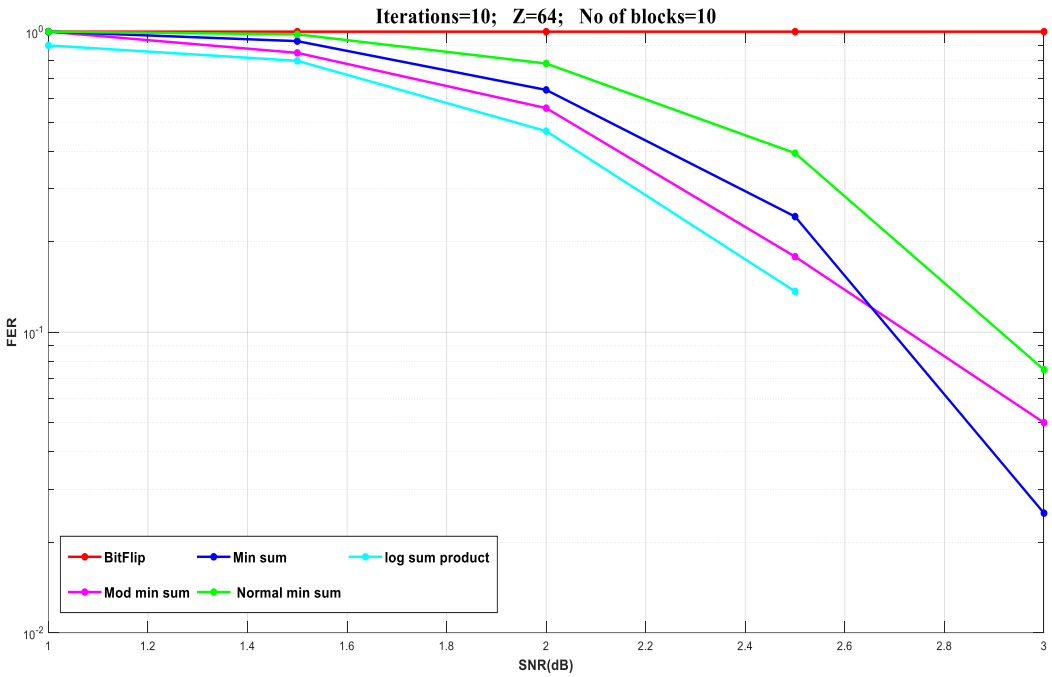
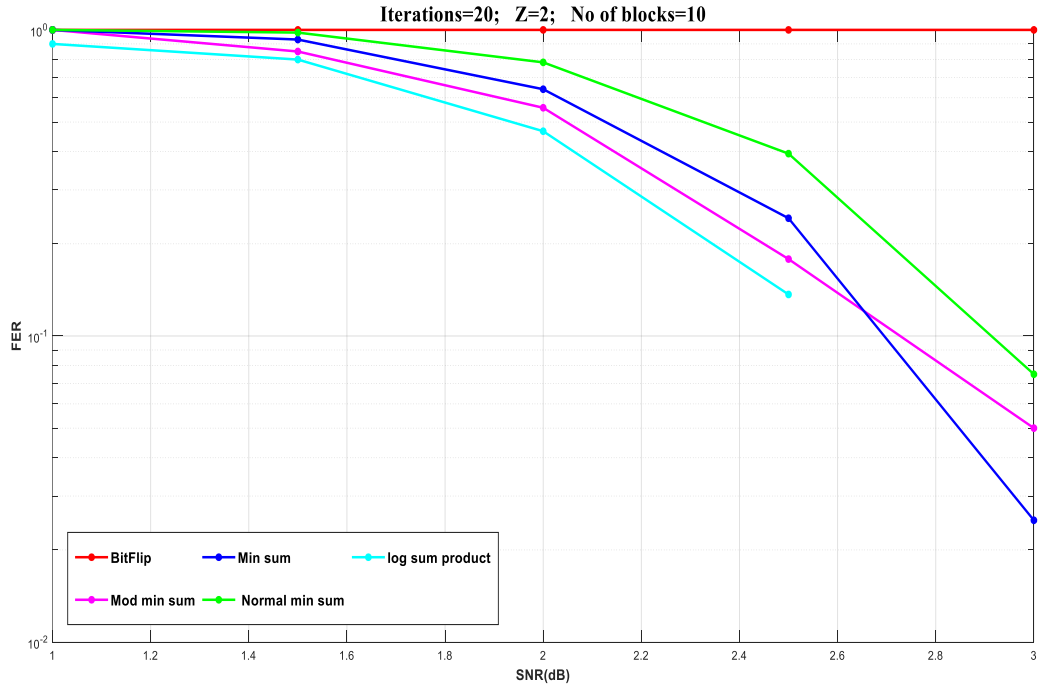


Figure 9: FER of different LDPC decoding algorithms with number of iterations=10, number of blocks=10 and Expansion factor=64.



Figure

Figure 10: FER of different LDPC decoding algorithms with number of iterations=20, number of blocks=10 and Expansion factor=2.

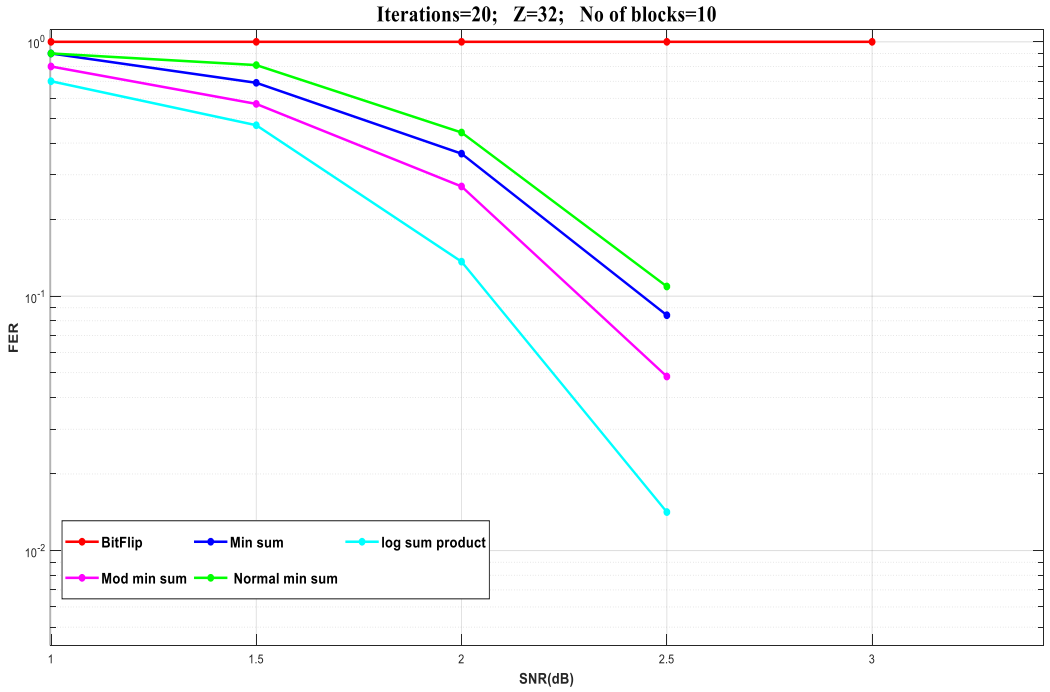


Figure 11: FER of different LDPC decoding algorithms with number of iterations=20, number of blocks=10 and Expansion factor=32.

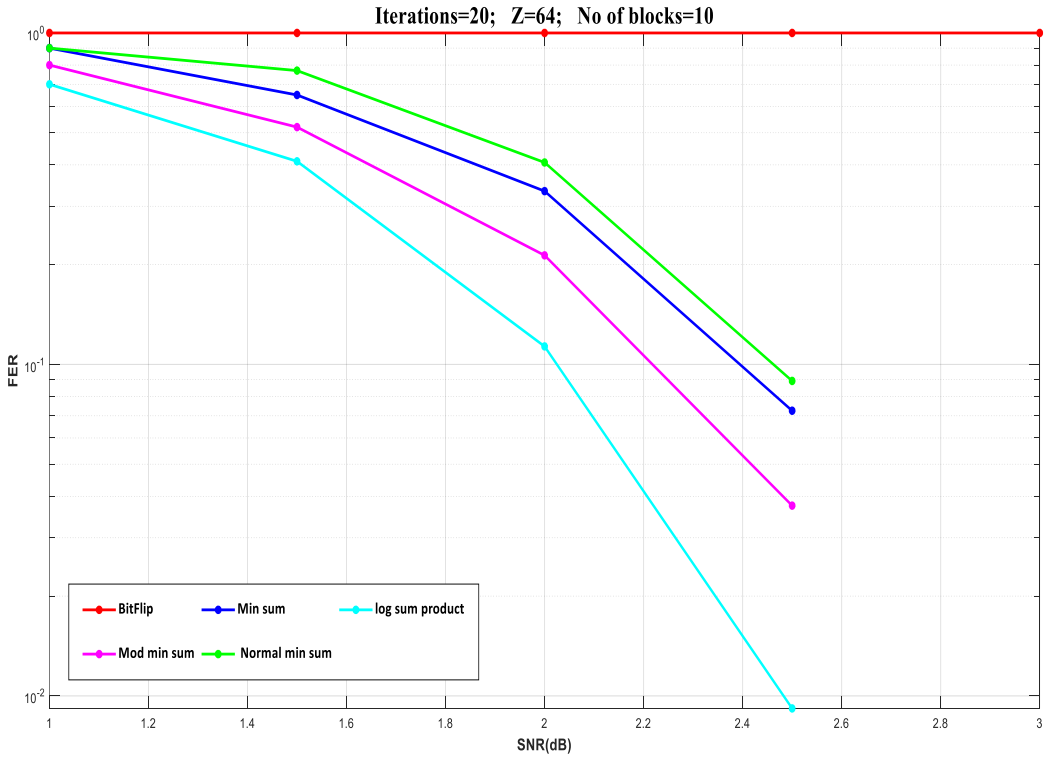


Figure 12: FER of different LDPC decoding algorithms with number of iterations=20, number of blocks=10 and Expansion factor=64.

CONCLUSION:

As the number of iterations and expansion factor increases, the BER and FER are gradually decreasing with an increase in SNR. On comparing between these decoding algorithms Mod Min-Sum decoding algorithm gives better BER performance and Log Sum-Product algorithm gives better FER performance.

FUTURE SCOPE:

Latency is the one of the main services offered by 5G communications. As per ITU the standard latency offered by 5G communication is ≤ 1msec. This project mainly focuses on BER and FER performances of different LDPC decoding algorithms for next generation wireless communications. In future this work can be extended by calculating better latency decoding algorithms. Different channels can be used in order to improve performance.

PAPER PUBLICATION DETAILS:

The proposed paper is communicated to International Journal Of Communication Systems, Wiley.

REFERENCES:

[1] R.K. Deergha Rao “Performance of digital communication over fading channels” Channel coding techniques for wireless communications, Springer, New Delhi (2019).

[2] R.G. Gallager, “Low-Density Parity-Check Codes”, in Research Mono-graph series. Cambridge, MIT Press, 1963.

[3] Honary, B., et al.: On construction of low density parity check codes. In: 2nd International Workshop on Signal Processing for Wireless Communication (SPWC 2004), London, UK, 2–4 June 2004.

[4] Ahmed A.Emran ; Maha Elsabrouty “Simplified variable-scaled min sum LDPC decoder for irregular LDPC codes”, 2014 IEEE 11th Consumer Communications and Networking Conference (CCNC) 10-13 Jan. 2014.

[5] Hashemi, S.A.; Condo, C.; Gross, W.J.: Fast Simplified Successive Cancellation List Decoding of polar Codes, in Proc. IEEE Wireless Communications and Networking Conf. Workshops, San Francisco, California, March 2017.

[6] Jung hyun bae, Ahmed abotabl, Hsien-ping lin, Kkee-bong song and Jungwon lee “An overview of channel coding for 5G NR cellular communications” SIP (2019), vol. 8, e17, page 1 of 14 © The Authors, 2019. doi:10.1017/ATSIP.2019.10

[7] 3GPP: RP-161214 Study on New Radio Access Technology, 2016

[8] 3. 3GPP: TR 38.802 Study on New Radio Access Technology Physical Layer Aspects, 2017