International Journal of Science and Research (IJSR)
ISSN (Online): 2319-7064
Index Copernicus Value (2015): 78.96 | Impact Factor (2015): 6.391

FPGA Implementation of LDPC Encoder and Decoder using Bit Flipping Algorithm

B. Sai Reddy[1], V. Seetha Rama Rao[2]

[1]Assistant Professor, Department of ECE, SNIST, Hyderabad

[2]Assistant Professor, Department of ECE, SNIST, Hyderabad

Abstract: ***Communication system transmits data from source to transmitter through a channel or medium such as wired or wireless. The efficiency of received data depends on medium and noise . LDPC codes are used to correct the errors by adding redundant symbols to the original data called as Error Correction Codes (ECCs).Without ECCs data need to retransmit if it could detect there is an error in the received data. ECC are also called as forward error correction (FEC) as we can correct bits without retransmission. ECCs are really helpful for high speed and long distance Communication LDPC consist of mainly Encoder and Decoder blocks, AWGN channel, Detector. These blocks are designed by using Verilog HDL with Xilinx ISE Design suite 12.4 version tool. The designs implemented in Xilinx SPARTAN 3E XC3S500EFG320 FPGA board.***

Keywords: FPGA, LDPC, HDL, Error Correction Codes, ISE, AWGN

1. Introduction

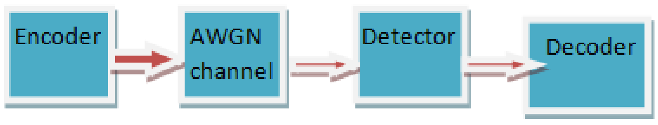
New generalizations of Gallager's LDPC codes by a number of researchers produced new regular LDPC codes which easily outperform the best turbo codes, as well as offering certain practical advantages and an arguably cleaner setup for theoretical results. Today, design techniques for LDPC codes exist which enable the construction of codes which approach the Shannon's capacity to within hundredths of a decibel. So rapid progress has been in this area that coding theory today is in many ways unrecognizable from its state just a decade ago.

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

A proper choice of an LDPC code and its decoding algorithm depends on the application and is mainly driven by the performance/complexity considerations. Maximum likelihood decoding algorithm requires exhaustive search and is suitable only for shorter codeword's. Belief propagation decoding algorithm is computation intensive and requires unnecessary iterations.
So to achieve the good performance with the lower decoding complexity we are using Bit Flipping decoding which is a iterative binary message passing algorithms. Message passed in this technique are binary 0's and 1's hence computational complexity is small compared to computation of probabilities in Belief Propagation and also number computations are less compared to maximum likelihood decoding technique. This technique uses Tanner graphical representation to detect and corrects the errors The chapter I presents the motivation, objective of low density parity check codes.(LDPC) Chapter II Presents system design of LDPC Chapter III presents simulation results of individual blocks Chapter IV presents conclusion and future scope

2. System Design

LDPC architecture mainly consists of four main blocks, random bits are transmitted and a then transmitted through AWGN channel and then through LDPC Decoder the output is obtained.



2.1 LDPC Algorithm

A code word c is generated as

$$C= UG$$

Where U is the block of information bits and G is the generator matrix. A valid codeword can be verified using

$$HC^T= 0$$

Where H is the parity check matrix. If the result in (4.2) is nonzero, the codeword C is invalid and an error correction procedure should be used in this case.

2.2 Encoder Design

At the encoder, information message bits are coded by multiplying message blocks with generator matrix to obtain the codeword's, i.e. C= [U][G] as shown in figure 4.2.

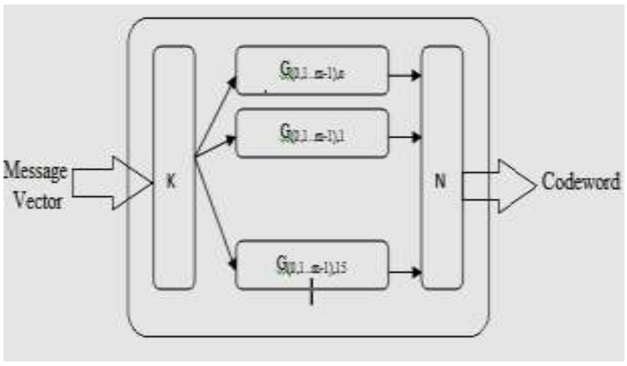


Figure 4.2: Encoder Block Diagram

Each structure labeled $G_{\{0,1,,,m-1\}i}$ are XOR structures performs modulo-2 operations on the incoming message bits and the resultant code words will be of N bits.

2.3Decoder Design

LDPC architecture designed using Bit Flipping decoding technique. At the decoder received codeword is verified and if there is any error, it will be corrected using Bit Flipping decoding algorithm. For the bit-flipping algorithm the messages passed along the Tanner graph edges are binary. Initially a bit node sends a message declaring if it is a one or a zero and each check node sends a message to each connected bit node, declaring what value the bit is based on the information available to the check node.

The check node determines that its parity-check equation is satisfied if the modulo 2 sum of the incoming bit values is zero. If the majority of the messages received by a bit node are different from its received value the bit node flips its current value. Block diagram of decoder is shown in

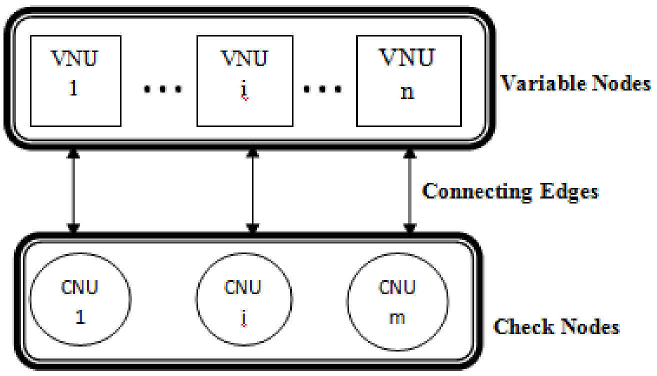


Figure: Decoder Block Diagram

3. System Implementation of LDPC

In this section, LDPC encoder, AWGN channel and LDPC decoder blocks designed using Xilinx synthesis tool are described.

3.1 Encoder Implementation

Encoder uses generator matrix to encode the information bits in to the codeword. Both generator and parity check matrix are in inter-related. In standard form parity check matrix is given by

$$H = [A \mid I_{n-k}]$$

and generator matrix is given by

$$G = [I_k \mid A^T]$$

Initially regular parity check matrix is constructed by considering the uniform row weights and uniform column weights. Using that regular parity matrix, generator matrix is constructed by using Gaussian elimination method. There are two types of parity matrices in LDPC coding one is Regular matrix and another one is irregular matrix.

Regular matrix is one in which column weight W_c is same for all columns and row weight is given by $W_r = W_c(n/m)$. In this project, regular parity matrix size of 8x16 for coding 8-bit message blocks has been used. Irregular matrices are those which doesn't have uniform row and column weight. LDPC coding technique with respect to two different regular parity matrices is described below.

3.1.1 Regular Parity Matrix1

By considering column weight $W_c = 2$ and row weight Wr = 4 for all rows and column regular parity check matrix of size 8x16 is constructed as shown in figure 5.1.

$$\begin{pmatrix}
1&0&1&0&0&0&0&0&1&0&0&1&0&0&0&0\\
0&1&0&0&0&1&0&0&1&0&1&0&0&0&0&0\\
0&1&0&0&0&0&1&0&0&1&0&0&0&0&0&1\\
0&0&0&1&1&1&0&0&0&0&0&0&0&0&0&1\\
1&0&0&0&0&0&0&1&0&0&1&0&0&1&0&0\\
0&0&0&1&0&0&1&0&0&0&0&0&1&1&0&0\\
0&0&0&0&0&0&0&1&0&1&0&1&0&0&1&0\\
0&0&1&0&1&0&0&0&0&0&0&0&1&0&1&0
\end{pmatrix}$$

Figure 5.1: Regular Parity Matrix1

To transfer the above regular parity check matrix to standard form i.e $H=[A \mid I_{n-k}]$ Gaussian elimination method is applied . Gaussian elimination involves elementary row operations which includes interchanging of two rows and modulo 2 addition of one row to another row. Elementary row operations which are necessary to transfer the parity matrix to standard form are listed below and resulting parity matrix in its standard form.

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

Figure 5.2: Standard Parity Matrix1

Obtained parity check matrix is translated to standard form of generator matrix i.e $G=[I_k \mid A^T]$ is shown in figure 5.2 .

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

Figure 5.3: Generator Matrix1

Let us consider an 8 bit information message U =[1 0 0 1 0 1 0 0], this message vector encoded by multiplying it with generator matrix i.e,

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

$$=[1\,0\,0\,1\,0\,1\,0\,0\,1\,0\,0\,0\,0\,1\,0\,0]$$

Figure 5.4: Encoding of Regular Parity Matrix

From resultant 16-bit codeword, we can observe that the 8-bit information bits are positioned at the LSB bits and these types of codes are called 'Systematic Codes'.

3.1.2 Regular Parity Matrix2:

Consider another 8x16 regular parity check matrix of $W_c = 2$ and Wr = 4 as shown in figure 5.5.

$$\begin{matrix}1&0&0&1&1&0&0&0&0&0&0&0&0&0&1&0\\1&0&0&0&0&0&0&0&0&1&0&0&1&1&0&0\\0&0&1&0&1&0&0&0&1&0&0&1&0&0&0&0\\0&0&0&1&0&0&0&1&0&0&1&1&0&0&0&0\\0&0&1&0&0&1&0&0&0&0&0&0&0&1&0&1\\0&0&0&0&0&0&1&0&1&1&1&0&0&0&0&0\\0&1&0&0&0&0&1&0&0&0&0&0&0&0&1&1\\0&1&0&0&0&1&0&1&0&0&0&0&1&0&0&0\end{matrix}$$

Figure 5.5: Regular Parity Matrix2

To transfer the above regular parity check matrix to standard form i.e $H=[A \mid I_{n-k}]$ Gaussian elimination method is applied to the above matrix. After applying Gaussian elimination obtained parity matrix as shown in figure 5.6.

$$H=\begin{pmatrix}0&0&0&0&0&0&0&0&1&0&0&0&0&0&0&0\\0&0&1&1&1&0&1&1&0&1&0&0&0&0&0&0\\0&0&1&1&1&0&0&1&0&0&1&0&0&0&0&0\\0&0&1&0&1&0&0&0&0&0&0&1&0&0&0&0\\0&1&0&0&0&1&0&1&0&0&0&0&1&0&0&0\\1&1&1&1&1&1&1&0&0&0&0&0&0&1&0&0\\1&0&0&1&1&0&0&0&0&0&0&0&0&0&1&0\\1&1&0&1&1&0&1&0&0&0&0&0&0&0&0&1\end{pmatrix}$$

Figure 5.6: Standard Parity Matrix2 Obtained parity matrix is translated to standard form of generator matrix i.e $G= [I_k | A^T]$ as

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

Figure 5.7: Generator Matrix2

Let us consider an 8 bit information message U = [0 0 1 1 0 0 1 1], this message vector encoded by multiplyingit with generator matrix i.e,

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

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

Figure 5.8: Encoding of Regular Parity Matrix2

Coding for this encoder part is done on verilog and encoding is tested for various information message blocks satisfactorily. RTL schematic for encoder block obtained from Xilinx as shown in figure 5.9(a) and figure 5.9(b).

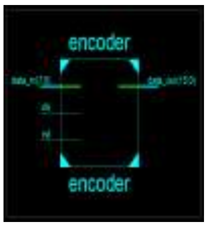


Figure 5.9(a): Encoder RTL schematic

3.2 AWGN Channel Design

AWGN is often used as a channel model in which the only impairment to communication is a linear addition of wideband or white noise with a constant spectral density and a Gaussian distribution of amplitude. The model does not account for fading, frequency selectivity, interference, nonlinearity ordispersion. However, it produces simple and tractable mathematical models which are useful for gaining insight into the underlying behavior of a system before these other phenomena are considered.

The AWGN channel is a good model for many satellite and deep space communication links. It is not a good model for most terrestrial links because of multipath, terrain blocking, interference, etc.

When encoded of signals are transmitted through channel, coded signal may get corrupted by noise in the channel and other interferers. To model that effect here we are adding AWGN noise to the coded signal.

Additive White Gaussian Noise (AWGN) is a basic noise model used in Information theory to mimic the effect of many random processes that occur in nature, which is the statistically random radio noise characterized by a wide frequency range with regards to a signal in the communications channel. The modifiers denote specific characteristics:

- 'Additive' because it is added to any noise that might be intrinsic to the information system.
- 'White' refers to idea that it has uniform power across the frequency band for the information system. It is an analogy to the colour white which has uniform emissions at all frequencies in the visible spectrum.
- 'Gaussian' because it has a normal distribution in the time domain with an average time domain value of zero.

For encoded codeword of regular matrix 1, 8-bit noise is added which corrupts the codeword. Encoded codeword is
C= [1 0 0 1 0 1 0 0 1 0 0 0 0 1 0 0] and
8-bit noise is [1 0 0 0 0 0 0 0]
[1 0 0 1 0 1 0 0 0 1 1 0 1 1 0 0] + [1 0 0 0 0 0 0 0] =
[0 0 0 1 0 1 0 0 1 0 0 0 0 1 0 0]

Here plus symbol indicates modulo-2 addition operation and we can observe that noise has corrupted the codeword by flipping first bit '1' to '0'. Similarly for encoded codeword of example 2, 8-bit noise is added which corrupts the codeword. Encoded codeword is
C= [0 0 1 1 0 0 1 1 0 0 1 1 1 1 1 0] and 8-bit noise is [1 0 0 0 0 0 0 0]
[0 0 1 1 0 0 1 1 0 0 1 1 1 1 1 0] + [0 0 0 0 0 0 0 1] =
[0 0 1 1 0 0 1 0 0 0 1 1 1 1 1 0]

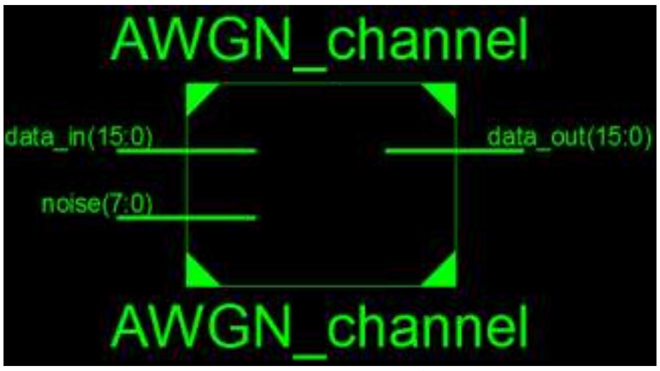


Figure 5.11(b): RTL schematic of AWGN channel

3.3 Decoder Implementation

At the decoder initially, received codeword is tested whether it is valid or not. Codeword is tested by multiplying it with standard parity matrix. If the resultant of multiplication is zero than the codeword is valid and LSB 8-bits are the transmitted message vector.
i.e $HY^T = 0$

If the resultant of multiplication is not zero than errors in the received codeword is corrected by bit flipping decoding technique.

3.3.1 Decoding of Regular Matrix1:

For regular matrix 1, received codeword is verified as shown in figure

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

Figure 5.12: Detection

Since the resultant of detection is not zero, Bit Flipping Decoding technique is used to correct the flipped bits in the received codeword.

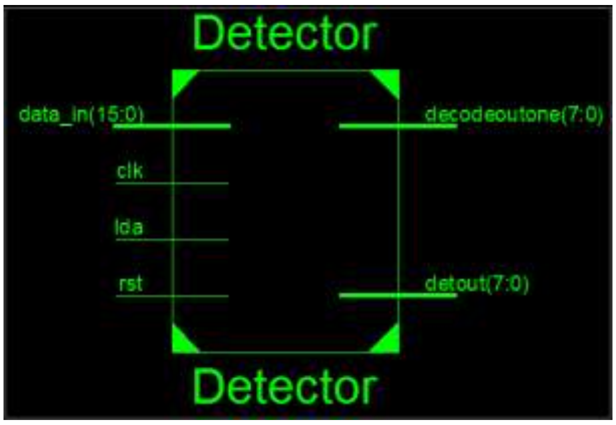


Figure 5.13 (a): RTL schematic of detector

At the decoder Bit Flipping technique is used to correct the error. The bit flipping algorithm is a hard-decision message-passing algorithm for LDPC codes. Here Graphical representation of parity check matrix known as Tanner Graph is used in order to correct errors. This graph consists of variable nodes, check nodes and their connecting edges as shown in figure 5.14. For eight rows there will be eight check nodes and for sixteen columns there will be sixteen variable nodes.

$$\begin{matrix} 1&0&1&0&0&0&0&0&1&0&0&1&0&0&0&0 \\ 0&1&0&0&0&1&0&0&1&0&1&0&0&0&0&0 \\ 0&1&0&0&0&0&1&0&0&1&0&0&0&0&0&1 \\ 0&0&0&1&1&1&0&0&0&0&0&0&0&0&0&1 \\ 1&0&0&0&0&0&0&1&0&0&1&0&0&1&0&0 \\ 0&0&0&1&0&0&1&0&0&0&0&0&1&1&0&0 \\ 0&0&0&0&0&0&0&1&0&1&0&1&0&0&1&0 \\ 0&0&1&0&1&0&0&0&0&0&0&0&1&0&1&0 \end{matrix}$$

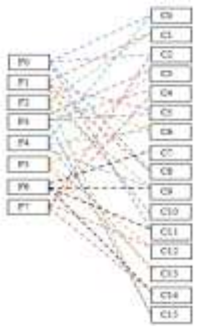


Figure 5.14: Parity Matrix and its Tanner Graph

The LDPC codeword obtained from regular matrix1 for the 8-bit input [1 0 0 1 0 1 0 0]
C = [1 0 0 1 0 1 0 0 1 0 0 0 0 1 0 0]
C is sent though a AWGN channel and the received signal is
Y = [0 0 0 1 0 1 0 0 1 0 0 0 0 1 0 0]
Various operations associated with bit flipping technique to correct the errors are described below in three steps.

Step 1:
Initially received 16-bit of codeword bits are assigned to 16 variable nodes of tanner graph. Than these variable nodes sends the assigned bit values to the connected check nodes.
Variable Nodes: v0 v1 v2 v3 v4 v5 v6 v7 v8 v9 v10 v11 v12 v13 v14 v15
Assigned Bits: 0 0 0 1 0 1 0 0 1 0 0 0 0 1 0 0

Step 2:
Check nodes computes the responses for all connected variable nodes.

The 1-st check node is connected to the variable nodes [0, 2, 8, 11] and so the message from the 1-st check node are,

$$C1,0 = 0+1+0=1;$$
$$C1,2 = 0+1+0=1;$$
$$C1,8 = 0+0+0=0;$$
$$C1,11 = 0+0+1=1;$$

The 2-nd check node is joined to the variable nodes [1, 5, 8,10] and so the message from the 2-nd check node are,

C2,1 = 1+1+0=0;
C2,5 = 0+1+0=1;
C2,8 = 0+1+0=1;
C2,10 = 0+1+1=0;

The 3-rd check node is joined to the variable nodes [1, 6, 9,15] and so the message from the 3-rd check node are,

C3,1 = 0+0+0=0;
C3,6 = 0+0+0=0;
C3,9 = 0+0+0=0;
C3,15 = 0+0+0=0;

The 4-th check node is joined to the variable nodes [3, 4, 5, 15] and so the message from the 4-th check node are,

C4,3 = 0+1+0=1;
C4,4 = 1+1+0=0;
C4,5 = 1+0+0=1;
C4,15 = 1+0+1=0;

The 5-th check node is joined to the variable nodes [0, 7, 10,13] and so the message from the 5-th check node a

C5,0 = 0+0+1=1;
C5,7 = 0+0+1=1;
C5,10 = 0+0+1=1;
C5,13 = 0+0+0=0;

The 6-th check node is joined to the variable nodes [3, 6, 12, 13] and so the message from the 6-th check node are,

C6,3 = 0+0+1=1;
C6,6 = 1+0+1=0;
C6,12 = 1+0+1=0;
C6,13 = 1+0+0=1;

The 7-th check node is joined to the variable nodes [7, 9, 11, 14] and so the message from the 7-th check node are,

C7,7 = 0+0+0=0;
C7,9 = 0+0+0=0;
C7,11 = 0+0+0=0;
C7,14 = 0+0+0=0;

The 8-th check node is joined to the variable nodes [2, 4, 12, 14] and so the message from the 8-th check node are,

C8,2 = 0+0+0=0;
C8,4 = 0+0+0=0;
C8,12 = 0+0+0=0;
C8,14 = 0+0+0=0;

All responses computed by check nodes are forwarded to the variable nodes.

Step 3:

In step 3 Majority check operation is performed at the variable nodes. Each variable node receives two responses from connected check nodes. Now each variable nodes have three bits, one is initially assigned and other two are received from check nodes. Majority check operation is computed at each variable nodes and initially assigned bits which are different from results of majority check operation are flipped.

First eight LSB bits in the corrected codeword will be the transmitted information. From the above table we can observe that 0-th bit is corrected by flipping it from ‘0’ to ‘1’. Corrected codeword is [1 0 0 1 0 1 0 0 1 0 0 0 0 1 0 0] and starting 8-bits of LSB will be the transmitted information bits i.e U = [1 0 0 1 0 1 0 0]. Majority check operations at variable nodes are tabulated in Table 5.1.

Table 5.1: Majority Check of Regular Matrix 1

Variable Nodes	Initially Assigned	From Check nodes		Corrected codeword
V0	0	1	1	1
V1	0	0	0	0
V2	0	1	0	0
V3	1	1	1	1
V4	0	0	0	0
V5	1	1	1	1
V6	0	0	0	0
V7	0	1	0	0
V8	1	0	1	1
V9	0	0	0	0
V10	0	0	1	0
V11	0	1	0	0
V12	0	0	0	0
V13	1	0	1	1
V14	0	0	0	0
V15	0	0	0	0

3.3.2 Decoding of Regular Matrix2

Now consider the error correction for the received codeword of regular matrix2. The 16-bit LDPC codeword for the 8-bit input [00110011] is,

C = [0 0 1 1 0 0 1 1 0 0 1 1 1 1 1 0]

Codeword received at the decoder is,

Y = [0 0 1 1 0 0 1 0 0 0 1 1 1 1 1 0]

Initially at the decoder, codeword is verified weather that is valid or not by multiplying it with standard parity matrix. Figure 5.15 shows the detection of error in the received codeword.

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

Figure 5.15: Detection

Step1

Initially received 16-bit of codeword are assigned to 16 variable nodes of tanner graph. Than these variable nodes sends the assigned bit values to the connected check nodes.

Variable Nodes: v0 v1 v2 v3 v4 v5 v6 v7 v8 v9 v10 v11 v12 v13 v14 v15

Assigned Bits: 0 0 1 1 0 0 1 0 0 0 1 1 1 1 1 0

Step 2

In Step 2 the check node messages are calculated. The 1-st check node is joined to the variable nodes [0, 3, 4, 14] and so the message from the 1-st check node are,

C1,0 = 1+0+1=0;
C1,3 = 0+0+1=1;
C1,4 = 0+1+1=0;
C1,14 = 0+1+0=1;

The 2-nd check node is joined to the variable nodes [0, 9, 12, 13] and so the message from the 2-nd check node are,

C2, 0 = 0+1+1=0;
C2, 9 = 0+1+1=0;
C2,12 = 0+0+1=0;
C2,13 = 0+0+1=1;

The 3-rd check node is joined to the variable nodes [2, 4, 8, 11] and so the message from the 3-rd check node are,

C3,2 = 0+0+1=1;
C3,4 = 1+0+1=0;
C3,8 = 1+0+1=0;
C3,11 = 1+0+0=1;

The 4-th check node is joined to the variable nodes [3, 7, 10, 11] and so the message from the 4-th check node are,

C4,3 = 0+1+1=0;
C4,7 = 1+1+1=1;
C4,10 = 1+0+1=0;
C4,11 = 1+0+1=0;

The 5-th check node is joined to the variable nodes [2, 5, 13, 15] and so the message from the 5-th check node are,

C5,2 = 0+1+0=1;
C5,5 = 1+1+0=0;
C5,13 = 1+0+0=1;
C5,15 = 1+0+1=0;

The 6-th check node is joined to the variable nodes [6, 8, 9, 10] and so the message from the 6-th check node are,

C6,6 = 0+0+1=1;
C6,8 = 1+0+1=0;
C6,9 = 1+0+1=0;
C6,10 = 1+0+0=1;

The 7-th check node is joined to the variable nodes [1, 6, 14, 15] and so the message from the 7-th check node are,

C7,1 = 1+1+0=0;
C7,6 = 0+1+0=1;
C7,14 = 0+1+0=1;
C7,15 = 0+1+1=0;

The 8-th check node is joined to the variable nodes [1, 5, 7, 12] and so the message from the 8-th check node are,

C8,1 = 0+0+1=1;
C8,5 = 0+0+1=1;
C8,7 = 0+0+1=1;
C8,12 = 0+0+0=0;

All responses computed by check nodes are forwarded to the variable nodes.

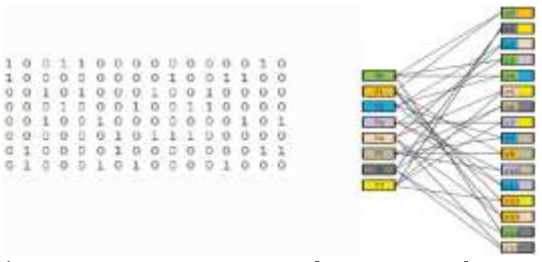


Figure 5.16: Parity Matrix and Tanner Graph

Step 3

Majority check operations are performed at the each variable node and the Majority check operations at the variable nodes are tabulated in table 5.2.

Table 5.2: Majority check of Regular matrix 2

Variable Nodes	Initially Assigned	From Check nodes		Corrected codeword
V0	0	0	0	0
V1	0	0	0	0
V2	1	1	1	1
V3	1	0	1	1
V4	0	1	0	0
V5	0	0	0	0
V6	1	1	1	1
V7	0	1	1	1
V8	0	0	0	0
V9	0	1	0	0
V10	1	1	1	1
V11	1	1	1	1
V12	1	1	1	1
V13	1	0	1	1
V14	1	0	1	1
V15	0	0	0	0

The 7-th bit is corrected by flipping it from '0' to '1' .Corrected codeword is [0 0 1 1 0 0 1 1 0 0 1 1 1 1 1 0] and LSB 8-bits will be the transmitted information bits i.e U=[0 0 1 1 0 0 1 1].

Initially reset (rst) signals is made high, for that output(decode out) equals to zero. When reset signal is made low and load (lad) signal is made high, decoder activates and variable node sends the received codeword bits to connected check nodes. Check nodes in response computes parity operations on received bits and send back the resulting bits to variable nodes. Now each variable node has three bits, one is initially assigned received codeword bit and other two response bits from check nodes. On these three bits, variable nodes computes majority check operation and those resulting bits are the original bits of the codeword.

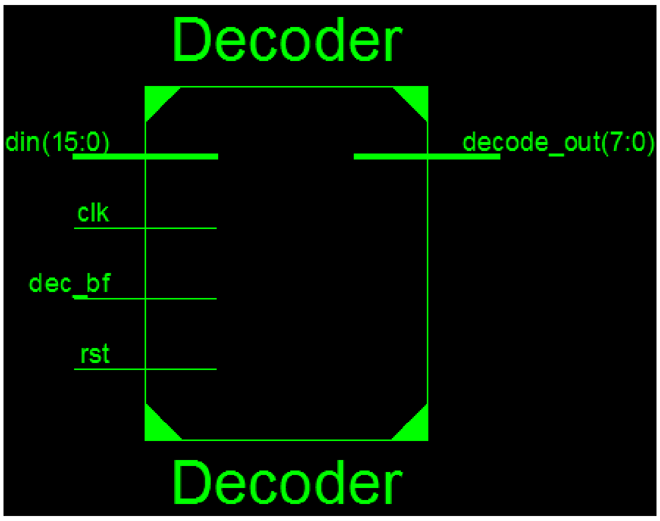


Figure 5.18(a): RTL schematic of decoder

Initially reset (rst) signals is made high, for that output(decode out) equals to zero. When reset signal is made low and load (lad) signal is made high, decoder activates and variable node sends the received codeword bits to connected check nodes. Check nodes in response computes parity operations on received bits and send back the resulting bits to variable nodes. Now each variable node has three bits, one is initially assigned received codeword bit and other two response bits from check nodes. On these three bits, variable nodes computes majority check operation and those resulting bits are the original bits of the codeword.

Initially reset signal is made high, for that condition both encoder and decoder datives. Next reset is made high, for that condition encoder activates and generates 16-bit codeword (enc_data) for given 8-bit input (data in). Then this codeword is corrupted by the 8-bit noise (noise). Corrupted codeword (err_data_out) is received at the receiver and decoder is activated by making load signal low. Bit flipping decoding technique verifies the received codeword and decodes the correct 8-bit codeword (decode_out).

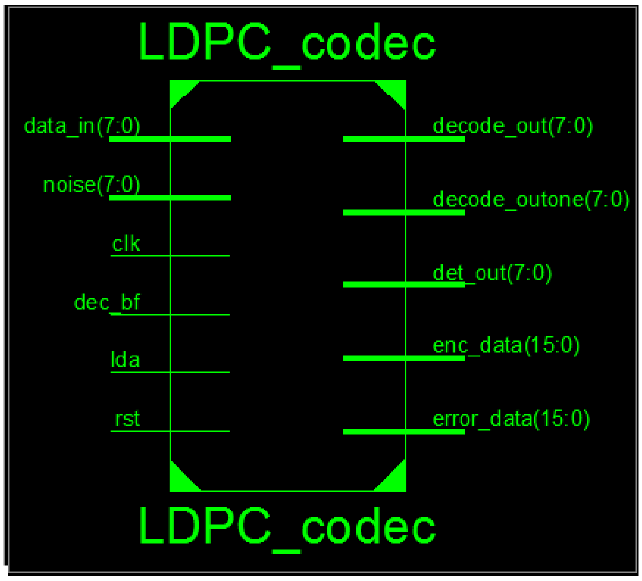


Figure 5.20 (a): RTL schematic of LDPC system

4. Simulation Results

In this section simulation results are obtained for two regular matrices are analyzed.

4.1. Regular Matrix1

Encoder simulation result is obtained for regular matrix1 is shown in figure 6.1. Initially reset (rst) is made high and output is zero and when reset goes low encoder gives 16-bit encoded codeword.

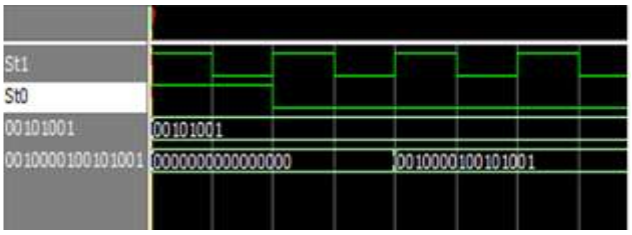


Figure 6.1: Encoder Simulation of Regular Matrix1

For the 8-bit input [1 0 0 1 0 1 0 0], it's respective 16-bit encoded codeword is C=[1 0 0 1 0 1 0 0 1 0 0 0 0 1 0 0].

When 16-bit codeword is transmitted through channel by adding 8-bit noise, noise corrupts the codeword and transmits the corrupted codeword to decoder. Channel simulation results obtained from modelsim is shown in figure 6.2.

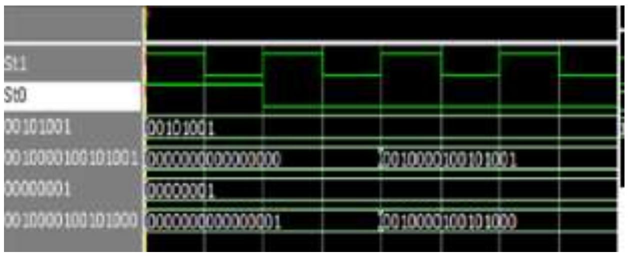


Figure 6.2: Channel Simulatio Regular Matrix1

Received 16-bit codeword is Y=[0 0 0 1 0 1 0 0 1 0 0 0 0 1 0 0]. Decoder detects and corrects the error by the use of Bit Flipping decoding technique and it's simulation result is shown in figure 6.3. Corrected codeword is C=[1 0 0 1 0 1 0 0 1 0 0 0 0 1 0 0] and the desired 8-bit message block is U=[1 0 0 1 0 1 0 0].

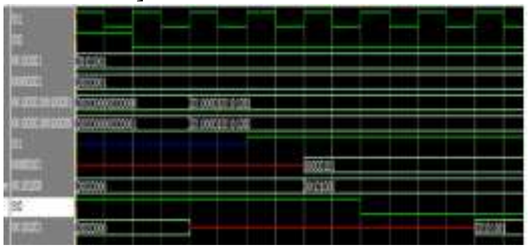


Figure 6.3: Decoder Simulation of Regular Matrix1

4.2. Regular Matrix2

For regular matrix 2, 8-bit input message block is U=[0 0 1 1 0 0 1 1] and it's coded 16-bit codeword is C= [0 0 1 1 0 0 1 1 0 0 1 1 1 1 1 0]. Simulation result is shown in figure 6.4.

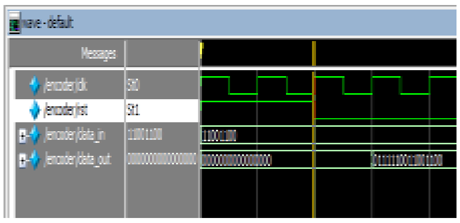


Figure 6.4: Encoder simulation of Regular Matrix2

Channel simulation results obtained from modelsim is shown in figure. These corrupted codewords are then becomes input to decoder.

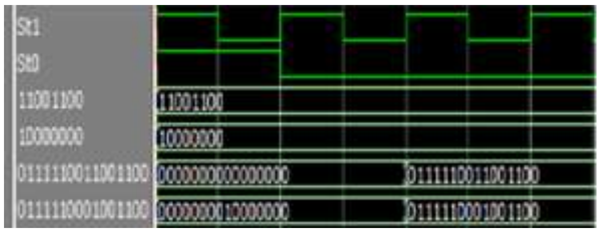


Figure 6.5: Channel Simulation of Regular Matrix2

At the Receiver received 16-bit corrupted codeword is verified and corrected by decoder. Received 16-bit codeword is Y=[0 0 1 1 0 0 1 0 0 0 1 1 1 1 1 0]. Decoder detects and corrects the error by the use of Bit Flipping decoding technique. Corrected codeword is C=[0 0 1 1 0 0 1

1 0 0 1 1 1 1 1 0] and transmitted 8-bit message block is U=[0 0 1 1 0 0 1 1].

Simulation results for regular matrix2 are shown in figure 6.6.

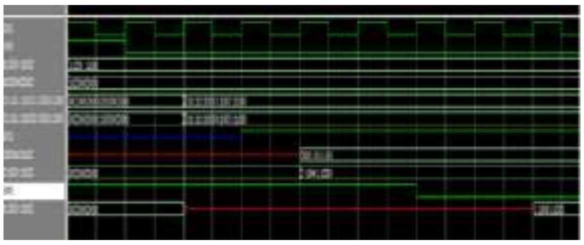


Figure 6.6: Decoder simulation of Regular Matrix2

5. Conclusion and Future Scope

LDPC is a linear error correcting code for transmitting a message over a noisy transmission channel. LDPC codes are finding increasing use in applications requiring reliable and highly efficient information transfer over bandwidth or return channel constrained links in the presence of data corrupting noise. LDPC architecture for 8-bit message blocks using regular parity check matrix of a size of 8 rows and 16 columns is designed and the simulation results for encoder, decoder, AWGN channel and detector are obtained.

Bit flipping decoding is message passing decoding technique which is having less computation complexity that can be used to improve the performance of LDPC codes. Coding technique is verified for various 8-bit message blocks successfully.

Future work involves improvement of the performance of decoding technique by making it capable of correct more than one bit errors. To correct more than one bit errors we may need to increase the number of message passing iterations .LDPC code is found to be a good error correcting code and its performance near to Shannon’s limit makes it a good candidate for various kind of modulation schemes such as OFDM,DVB-S2 and 802.3 an(Wi-Max) . In future it can be extended for 4G and 5G error correcting codes.

References

[1] Daisuke Miyashita, Ryo Yamaki, Kazunori Hashiyoshi,”An LDPC Decoder with Time-Domain Analog and Digital Mixed-Signal Processing” published in IEEE journal of solid-state circuits, vol. 49, no. 1, january 2014.
[2] R. G. Gallager, “Low density parity check codes,” IRE Trans. Inform. Theory, vol. IT-8, no.1, pp. 21–28, Jan. 1962.
[3] G.Boopathi Raja1, Dr.M.Madheswaran “design of improved majority logic fault detector/corrector based on efficient ldpc codes” published in ijareeie in Vol. 2, Issue 7, July 2013.
[4] Manjunatha P N , Prof .T.S Bharath kumar, 3Dr. M Z Kurian **“**design of ldpc architecture using verilog coding” published in ICICE-2014. gggouhiljj;okoo
[5] D. Miyashita, R. Yamaki, K. Hashiyoshi, H. Kobayashi, S. Kousai, Y. Oowaki, and Y. Unekawa, “A 10.4 pJ/b (32, 8) LDPC decoder with time-domain analog and digital mixed-signal processing,” in IEEE Int. Solid-State Circuits Conf. Dig. Tech. Papers, 2013, pp. 420–421.
[6] S. Hemati, A. H. Banihashemi, and C. Plett, “A 0.18-□mCMOSanalog min-sum iterative decoder for a (32,8) low-density parity-check (LDPC) code,” IEEE J. Solid-State Circuits, vol. 41, no. 11, pp. 2531–2540, Nov. 2006.
[7] T. Tian, C. Jones, J. Villasenor, and R. D.Wesel, “Construction of irregular ldpc codes with low error floors,” in Proceedings IEEE International Conference on Communications, 2003
[8] H. Zhong and T. Zhang, “Block-LDPC: A practical LDPC coding system design approach,”IEEE Trans. Circuits Syst. I, vol. 52, no. 4, pp. 766 – 775, Apr. 2005.
[9] Verilog HDL- Samir Palnitkar (2nd Edition). [13] S. J. Johnson, “Introducing Low-Density Parity-Check Codes,”.