Computation of Parity Check Matrices for Binary EG-LDPC Codes used in Communication Systems

J. Chinna Babu, C. Chinnapu Reddy, M.N. Giri Prasad
1Research Scholar, JNTUA Ananthapuramu & Assistant Professor, AITS, Rajampet.
2Head, Dept. of ECE, Govt. Polytechnic Adoni, India.
3Director of Admissions, JNTUA, Ananthapuramu, India.
Email: jchinnababu@gmail.com, ccreddyece@gmail.com, mahendragiri1960@gmail.com

Received: 05th January 2019, Accepted: 29th January 2019, Published: 28th February 2019

Abstract
The Low Density Parity Check (LDPC) are direct codes, which are true block and Shannon Limit codes. These codes are attained least error floors of data bits for data transfer applications used in communication systems. However, the proposed LDPC codes are more beneficial than Turbo codes because of reduction in the decoding complexity and detection of the errors in less cycle time. This results the reduction of decoding time, low decoding latency, complexity and as well as least error floorings in communication, when the transmitted data contains multiple error bits. This paper is proposed to represent the majority logic decoding/detecting of LDPC codes. This paper proposes the Generation of Originator and Parity estimated matrices for the Binary LDPC Codes. Here, the proposed techniques are hard decision decrypting and soft decision decrypting schemes. These schemes uses majority logic decoding based on the data transmission and reception in communication channel. This paper also elaborates the effective calculation of Euclidean distance and algorithm for constructing the LDPC codes.

Keywords
LDPC Codes; BF Algorithms; EG-LDPC; Turbo Codes; Coding Theory.

Introduction
Correcting Error Codes
The terminated data or parity bits is auxiliary to the novel coded data, therefore that the exact coded data can be recovered at the destination end without the requirement for the data re-transmission and similarly the faults can be discovered and rectified, if any faults are existing. For this, we are using sophisticated codes, which are called error-correcting codes.

Shannon's Theorem:
For the consistent broadcasting of data over a given communication medium, the data transmission rate should not exceed the channel capacity, which is proved by Shannon theorem.

LDPC Codes:
LDPC codes are established by R.G. Gallagher, hence these are also identified as Gallagher codes. Due to unbearable practical comprehension, these codes are abandoned, although these codes were designed in the early 1960's. These LDPC codes are direct error improving codes and methodologies near Shannon capability. These LDPC codes are the better blunder improving codes, used for encoding and decoding at present scenario [1]-[3]. Here are binary categories of LDPC codes are of two types.
LDPC Regular Codes: Regular type of codes take equivalent row masses \( W_r \) and equivalent column masses \( W_c \) (Equivalent number of ones in rows and columns).

\[
\begin{bmatrix}
1 & 0 & 0 & 1 & 0 & 1 \\
0 & 1 & 0 & 1 & 1 & 0 \\
0 & 1 & 1 & 0 & 0 & 1 \\
1 & 0 & 1 & 0 & 1 & 0
\end{bmatrix}
\]

Here \( W_r = 3 \) for all of the rows and \( W_c = 2 \) for all of the columns.

LDPC Irregular Codes: In these, all of the rows might have dissimilar masses \( W_r \neq \text{constant} \) and all of the columns should have dissimilar masses \( W_c \neq \text{constant} \). For example let us assume the following equivalence matrix which is having four number of rows and four number of columns. Generally, LDPC codes are having higher number of zeros than the ones. It can be considered in equivalence check matrix is as shown below.

\[
\begin{bmatrix}
1 & 0 & 0 & 1 \\
0 & 1 & 1 & 1 \\
1 & 0 & 0 & 0 \\
0 & 0 & 1 & 1
\end{bmatrix}
\]

If whole number of rows and columns in the above equivalence matrix comprise uniform amount of ones. Therefore, it can be named as Binary LDPC codes.
Various decoding algorithms have been established for decrypting of LDPC algorithms based on the following two schemes and they are
1) Hard Decision Decoding (HDD)
2) Soft Decision decoding schemes (SDD).
Decision Decoding:

Table I: Code Word Generation

<table>
<thead>
<tr>
<th>Input Bit 1</th>
<th>Input Bit 2</th>
<th>Added Parity from Transmitter</th>
<th>Created Code Word</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>000</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>011</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>101</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>111</td>
</tr>
</tbody>
</table>

Table II: Hamming Distance Estimation

<table>
<thead>
<tr>
<th>All Possible Code Words</th>
<th>Decision Output</th>
<th>Hamming Distance</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>001</td>
<td>1</td>
</tr>
<tr>
<td>011</td>
<td>001</td>
<td>1</td>
</tr>
<tr>
<td>101</td>
<td>001</td>
<td>1</td>
</tr>
<tr>
<td>110</td>
<td>001</td>
<td>3</td>
</tr>
</tbody>
</table>
Depends on the threshold value we may estimate the Euclidean Distance for all the code words, then the code word which may have least hamming distance that is the exact code word which was transmitted earlier [4]-[7].

<table>
<thead>
<tr>
<th>Valid Code Words</th>
<th>Voltage Level at Each Sampling Instance of Traditional Waveform</th>
<th>Calculation of Euclidean Distance</th>
<th>Euclidean Distance</th>
</tr>
</thead>
<tbody>
<tr>
<td>000 (0v, 0v, 0v)</td>
<td>0.2v, 0.4v, 0.7v</td>
<td>(0.2-0)^2+(0.4-0)^2+(0.7-0)^2</td>
<td>0.69</td>
</tr>
<tr>
<td>011 (0v, 1v, 1v)</td>
<td>0.2v, 0.4v, 0.7v</td>
<td>(0.2-0)^2+(0.4-1)^2+(0.7-1)^2</td>
<td>0.49</td>
</tr>
<tr>
<td>101 (1v, 0v, 1v)</td>
<td>0.2v, 0.4v, 0.7v</td>
<td>(0.2-1)^2+(0.4-0)^2+(0.7-1)^2</td>
<td>0.89</td>
</tr>
<tr>
<td>110 (1v, 1v, 0v)</td>
<td>0.2v, 0.4v, 0.7v</td>
<td>(0.2-1)^2+(0.4-1)^2+(0.7-0)^2</td>
<td>1.49</td>
</tr>
</tbody>
</table>

Table III: Euclidean Distance Calculation

LDPC Codes - Basics

 Undertake that the coded data to be encrypted is a k-bit data establishing a standard data, \( m = (m_1, m_2 \ldots m_k) \), and it is one of the vector of \( 2^k \) number of possible code words. The transmitter receipts this code pattern and creates a code word \( c = (c_1, c_2 \ldots c_n) \), where \( n \) is greater than \( k \), that is there is an addition of redundancy. Moreover coding of blocks and coding of convolution types are also the tools for the redundancy addition in error improving coding methods.

Linear Block Codes Definition

A slab of data code word \( c \) is a direct code word, if the code words form a trajectory subset of the trajectory set \( V_n \); it has \( k \) exactly self-governing vectors that are represented as code words, such that each probable code word is an exact grouping of subsets. This clarification represents that the set of \( 2^k \) possible code patterns establishes a trajectory subset of the set of words of \( n \) number of bits. An exact code is categorised by the datum that the summation of any two code patterns is also a code word or code pattern or code vector.

**Originator (Generator) Matrix**

Assume \( c \ (n, k) \) can be an exact direct code word and the vectors \( (g_1, g_2 \ldots g_k) \) are \( k \) exactly self-governing trajectories. Each code word is an exact grouping of them:

\[
C = m_1g_1 + m_2g_2 \ldots m_kg_k
\]

Where, all of the trajectory and matrix procedures are using XOR operation. These originally self-governing trajectories can be organised by using a matrix form said to be originator matrix \( G \):

\[
G = \begin{pmatrix}
g_1 & g_{1,1} & g_{1,2} & g_{1,3} & \cdots & g_{1,n} 
g_2 & g_{2,1} & g_{2,2} & g_{2,3} & \cdots & g_{2,n} 
\vdots & \vdots & \vdots & \vdots & \ddots & \vdots 
g_k & g_{k,1} & g_{k,2} & g_{k,3} & \cdots & g_{k,n}
\end{pmatrix}
\]

Where, the given code vector is \( m = (m_1, m_2 \ldots m_k) \), the equivalent code vector is attained by using multiplication of the matrices is as shown below.

\[
c = m \cdot G = (m_1, m_2, \ldots m_k) \cdot \begin{pmatrix}g_1 
g_2 
g_3 
\vdots 
g_k\end{pmatrix} = m_1g_1 + m_2g_2 + \cdots + m_kg_k
\]

**Equivalence Check Matrix \((H)\)**

Here, the matrix having equivalence check equations can be of size \((n-k)xn\) matrix with \((n-k)\) self-governing rows and it is the code, which is having dual space \( c \), i.e. \( GH^T = 0 \).
It can also established that the equivalence check estimations can be attained from the equivalence check matrix \( H \), i.e. \( CH^T = 0 \). Where, this context also assumes entirely a block cypher.

**Systematic Form of Block Codes**

The construction of a code word in methodical form is as shown in Fig. 2.1. In this method, a code word contains of \( k \) message bits monitored by \( n - k \) parity check bits.

![Fig. 2.1: Code Vector Systematic Form of a Block Code](image)

Thus, a systematic direct block code \( c(n, k) \) might be stated by the succeeding creator matrix:

\[
G = \begin{pmatrix}
1 & 0 & 0 & \ldots & 0 & P_{1,k+1} & P_{1,k+2} & \cdots & P_{1,n} \\
0 & 1 & 0 & \ldots & 0 & P_{2,k+1} & P_{2,k+2} & \cdots & P_{2,n} \\
0 & 0 & 1 & \ldots & 0 & P_{3,k+1} & P_{3,k+2} & \cdots & P_{3,n} \\
\vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\
0 & 0 & 0 & \ldots & 1 & P_{k,k+1} & P_{k,k+2} & \cdots & P_{k,n}
\end{pmatrix}
\]

Identity Matrix \((k \times k)\) Parity Matrix \((k \times n - k)\)

Which, in a compact notation, is \( G = [I_{k \times k} \ P_{k \times (n - k)}] \). The comparable equivalence check matrix is given by

\[
H = [P_{(n - k) \times k} \ I_{(n - k) \times (n - k)}].
\]

**Interpreting the Rectilinear Block Codes**

It can perceive from the communication scheme, that as a significance of its broadcast through a noisy medium, a code word might be acknowledged comprising some faults. The acknowledged vector can consequently be dissimilar from the conforming transferred code word, and it will be represented as \( r = (r_1, r_2, \ldots, r_n) \). An error occasion can be demonstrated as a fault pattern or fault code word \( e = (e_1, e_2, \ldots, e_n) \), where \( e = r + c \).

To distinguish the faults, it can be used by the fact that any legal code word must satisfy the state \( c.H^T = 0 \). A fault finding tool is created from the above notation, which accepts the succeeding expression \( s = r \times H^T \), where \( s = (s_1, s_2, \ldots, s_n) \) is named as the code word pattern. The perceiving process is accomplished over the traditional code pattern. If \( s \) is the entirely nil trajectory, the traditional trajectory is a legal code word or else, there are faults in the traditional code pattern. The pattern collection is tested to catch the conforming error pattern \( Ej \) for \( j = 1, 2, \ldots, N \) and the decrypted code word is attained by using \( m' = r + Ej \).

**Definition of LDPC Codes**

The LDPC codes are direct block codes, which are represented as \((n, k)\) or \((n, Wc, Wr)\), here \( n \) can be span of the code word, \( k \) can be distance of the data bits, \( Wc \) can be mass of the column (i.e. the amount of non-zero elements in a column of the parity check matrix), and \( Wr \) become mass of the row (i.e. the amount of non-zero elements in a row of the equivalence check matrix). Here are two noticeable features of LDPC codes. They are as is follows.

**Parity Check Notation**

These codes are characterized by an equivalence check matrix \( H \), here \( H \) is a dual matrix, contains ones and zeros. It must satisfy \( c.H^T = 0 \), here \( c \) is a code pattern.

**Low-Density**

Where \( H \) is a matrix, and it is a sparse matrix. It indicates that amount of ones are lower compared to zeros. It is the characteristic of \( H \) that assures the lower computational complexity.

**LDPC Construction Algorithm Construction of LDPC Codes: Algorithm**

\[
\text{PROCEDURE : } \text{LDPC CONSTRUCTION} \ (n, r, V, h) \\
n: \text{required length}; \ r: \text{rate}; V: \text{Column}; h: \text{degrees} \\
H \leftarrow \text{all zeros} \ n \ (1 - r) \times n \ \text{matrix} \\
\Rightarrow \text{Matrix initialization}
\]
\[ \alpha \leftarrow [\emptyset] \]
\[ \text{for } i \leftarrow 1: \text{max}(v) \text{ do} \]
\[ \text{for } j \leftarrow 1: V_{i \times n} \text{ do} \]
\[ \alpha \leftarrow \alpha(i,j) \]
\[ \text{end for} \]
\[ \text{end for} \]
\[ \beta \leftarrow [\emptyset] \]
\[ \text{for } i \leftarrow 1: \text{max}(h) \text{ do} \]
\[ \text{for } j \leftarrow 1: h_{x \times m} \text{ do} \]
\[ \beta \leftarrow \beta(i,j) \]
\[ \text{end for} \]
\[ \text{end for} \]
\[ \text{for } i \leftarrow 1: n \text{ do} \quad \triangleright \text{CONSTRUCTION} \]
\[ C \leftarrow \text{random variant} \ (\beta, \alpha_i) \]
\[ \text{for } j \leftarrow 1: \alpha_i \text{ do} \]
\[ H(C_j, i) \leftarrow 1 \]
\[ \text{end for} \]
\[ \alpha \leftarrow (\alpha - C) \]
\[ \text{end for} \]

\textit{end PROCEDURE}

The line 1 to 2 initializes the matrix \( H \) and \( \alpha \) vector with \( n(1-r)X \ n \) and null respectively. The lines 3 through 7 are initialization of \( \alpha \) vector with size of \( H \) matrix. The line 8 is initialized the \( \beta \) vector with Null value. The lines 9 through 13 are computed \( \beta \) vector with size \( H \) matrix. The line 14 is construct code word with size \( n \). In line 15, generate a random variant of \( C \) using vectors of \( \beta \) and \( \alpha \). The lines 16 through 19 to compute all 1’s in a given matrix. This algorithm will execute till end of required length. From the above algorithm we can develop or construct the LDPC Codes of the required size [8]-[9].

**Fig. 3: Encoder Block Diagram**
TABLE IV: Galois Addition

<table>
<thead>
<tr>
<th>+</th>
<th>$\alpha^0$</th>
<th>$\alpha^1$</th>
<th>$\alpha^2$</th>
<th>$\alpha^3$</th>
<th>$\alpha^4$</th>
<th>$\alpha^5$</th>
<th>$\alpha^6$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\alpha^0$</td>
<td>1</td>
<td>0</td>
<td>$\alpha^3$</td>
<td>$\alpha^1$</td>
<td>$\alpha^5$</td>
<td>$\alpha^2$</td>
<td>$\alpha^4$</td>
</tr>
<tr>
<td>$\alpha^1$</td>
<td>$\alpha^2$</td>
<td>$\alpha^3$</td>
<td>0</td>
<td>$\alpha^4$</td>
<td>$\alpha^0$</td>
<td>$\alpha^6$</td>
<td>$\alpha^5$</td>
</tr>
<tr>
<td>$\alpha^2$</td>
<td>4</td>
<td>$\alpha^6$</td>
<td>$\alpha^4$</td>
<td>0</td>
<td>$\alpha^5$</td>
<td>$\alpha^1$</td>
<td>$\alpha^3$</td>
</tr>
<tr>
<td>$\alpha^3$</td>
<td>3</td>
<td>$\alpha^1$</td>
<td>$\alpha^0$</td>
<td>$\alpha^5$</td>
<td>0</td>
<td>$\alpha^6$</td>
<td>$\alpha^2$</td>
</tr>
<tr>
<td>$\alpha^4$</td>
<td>6</td>
<td>$\alpha^2$</td>
<td>$\alpha^1$</td>
<td>$\alpha^6$</td>
<td>0</td>
<td>$\alpha^0$</td>
<td>$\alpha^3$</td>
</tr>
<tr>
<td>$\alpha^5$</td>
<td>7</td>
<td>$\alpha^6$</td>
<td>$\alpha^3$</td>
<td>$\alpha^2$</td>
<td>$\alpha^0$</td>
<td>0</td>
<td>$\alpha^1$</td>
</tr>
<tr>
<td>$\alpha^6$</td>
<td>5</td>
<td>$\alpha^2$</td>
<td>$\alpha^5$</td>
<td>$\alpha^0$</td>
<td>$\alpha^4$</td>
<td>$\alpha^3$</td>
<td>$\alpha^1$</td>
</tr>
</tbody>
</table>

TABLE V: Galois Multiplication

<table>
<thead>
<tr>
<th>$\times$</th>
<th>$\alpha^0$</th>
<th>$\alpha^1$</th>
<th>$\alpha^2$</th>
<th>$\alpha^3$</th>
<th>$\alpha^4$</th>
<th>$\alpha^5$</th>
<th>$\alpha^6$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\alpha^0$</td>
<td>$\alpha^0$</td>
<td>$\alpha^1$</td>
<td>$\alpha^2$</td>
<td>$\alpha^3$</td>
<td>$\alpha^4$</td>
<td>$\alpha^5$</td>
<td>$\alpha^6$</td>
</tr>
<tr>
<td>$\alpha^1$</td>
<td>$\alpha^2$</td>
<td>$\alpha^3$</td>
<td>$\alpha^4$</td>
<td>$\alpha^5$</td>
<td>$\alpha^6$</td>
<td>$\alpha^0$</td>
<td>$\alpha^1$</td>
</tr>
<tr>
<td>$\alpha^2$</td>
<td>4</td>
<td>$\alpha^3$</td>
<td>$\alpha^4$</td>
<td>$\alpha^5$</td>
<td>$\alpha^6$</td>
<td>$\alpha^0$</td>
<td>$\alpha^1$</td>
</tr>
<tr>
<td>$\alpha^3$</td>
<td>3</td>
<td>$\alpha^4$</td>
<td>$\alpha^5$</td>
<td>$\alpha^6$</td>
<td>$\alpha^0$</td>
<td>$\alpha^1$</td>
<td>$\alpha^2$</td>
</tr>
<tr>
<td>$\alpha^4$</td>
<td>6</td>
<td>$\alpha^5$</td>
<td>$\alpha^6$</td>
<td>$\alpha^0$</td>
<td>$\alpha^1$</td>
<td>$\alpha^2$</td>
<td>$\alpha^3$</td>
</tr>
<tr>
<td>$\alpha^5$</td>
<td>7</td>
<td>$\alpha^6$</td>
<td>$\alpha^0$</td>
<td>$\alpha^1$</td>
<td>$\alpha^2$</td>
<td>$\alpha^3$</td>
<td>$\alpha^4$</td>
</tr>
<tr>
<td>$\alpha^6$</td>
<td>5</td>
<td>$\alpha^0$</td>
<td>$\alpha^1$</td>
<td>$\alpha^2$</td>
<td>$\alpha^3$</td>
<td>$\alpha^4$</td>
<td>$\alpha^5$</td>
</tr>
</tbody>
</table>

LDPC Check Matrix Construction

1. Calculation of Primitive polynomial
2. Calculation of Degree terms
3. Galois addition and Galois multiplication
4. Calculation of $G(x)$
5. Calculation of $H$-matrix from $G(x)$.

For the better understanding of the procedure, here let us consider a binary code of length (16,8).

To assume ($n,k$) = (16,8)

**Step 1: The Calculation of Primitive Polynomial**

*Assume the odd numbers from 1 to 16

1, 3, 5, 7, 9, 11, 13, 15

* Write the odd numbers in binary form

1 − 0001, 3 − 0011, 5 − 0101, 7 − 0111, 9 − 1001, 11 − 1011, 13 − 1101, 15 − 1111

* Concatenate 1 as MSB bit

1 − 10001; 3 − 11011; 5 − 11101; 7 − 11111, 9 − 11001, 11 − 11011, 13 − 11101, 15 − 11111

*Now reverse the bits which are obtained after concatenation

1 − 10001; 3 − 11011; 5 − 10111; 7 − 11111, 9 − 10011, 11 − 11011, 13 − 10111, 15 − 11111

* Now relate the appended bits with the reversed bits
If both bit streams are equal ignore them otherwise note the numbers.

3 − 10011 − 11001
7 − 10111 − 11101
9 − 11001 − 10011
13 − 11101 − 10111

* Select the minimum value of remaining expression

For the primitive polynomial

\[ P(x) = x^4 + x^3 + x + 1 \]

**Step 2: Calculation Degree Terms**

While calculating the degree term value, if the value exceeds the 16, then ex-or it with the primitive polynomial \( x^4 + x^3 + x + 1 \), \( P(x) = 11001 \).

\[ \alpha^0 = 0001 = 1 \]

\[ \alpha^1 = 0010 = 2 \]
\[ \alpha^2 = 0100 = 4 \]
\[ \alpha^3 = 1000 = 8 \]
\[ \alpha^4 = 10000 \cdot 11001 = 01001 \]
\[ \alpha^5 = 10010 \cdot 11001 = 01011 \]
\[ \alpha^6 = 10110 \cdot 11001 = 01111 \]
\[ \alpha^7 = 11110 \cdot 11001 = 00111 \]
\[ \alpha^8 = 01010 \]
\[ \alpha^9 = 10100 \cdot 11001 = 00101 \]
\[ \alpha^{10} = 11010 \cdot 11001 = 00011 \]
\[ \alpha^{11} = 11100 \cdot 11001 = 00001 \]

**Step 3: Calculation of Generator Polynomial**

*The normalized form of originator polynomial \( G(x) \) is given by

\[ G(x) = (x - \alpha^0)(x - \alpha^1) \ldots (x - \alpha^n) \]

Where \( (n, k) = (16, 8) \), therefore

\[ G(x) = (x - \alpha^0)(x - \alpha^1)(x - \alpha^2)(x - \alpha^3)x - \alpha^4 \]
\[ (x - \alpha^5)(x - \alpha^6)(x - \alpha^7) \]

*By solving above equation the result is

\( G(x) = x^7 + x^6 + x^5 + x^4 + x^3 + x^2 + x + 1 \)

\[ \Rightarrow (1 1 1 1 1 1 1 1) \]

**Step 4: determination of parity check matrix**

\[
H = \begin{bmatrix}
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 & 1 & 0 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1
\end{bmatrix}
\]

*The first column is obtained by writing the bits from LSB to MSB of \( \alpha^0 = 1 = 0001 \) and in the next step downshift the data in column 1 to obtain column 2 and so on.

*If the ‘1’ appears as last bit in the column then for the next shift we should Ex-or the shifted data with \( G(x) \) to obtain that
column, otherwise data is shifted left/down.

*Here the last bit in column 3 is ‘1’ hence it is shifted down and is then ex-or with \( G(x) = [1 1 1 1 1 1 1] \) to obtain data in column 4. This is the procedure to obtain H-matrix for binary LDPC codes as shown in Fig.3.2.

The code word can be generated by making the product of message ‘m’ with originator matrix ‘G’, \( c = m \cdot G \)

1. Arranging the equivalence check matrix in systematic coded form

\[
H_{sys} = \begin{bmatrix} I_M & P_{M \times K} \end{bmatrix}
\]

2. Reorganising the regular equivalence check matrix

\[
G = \begin{bmatrix} P_{K \times M} & I_K \end{bmatrix}
\]

\[
G = \begin{bmatrix}
1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 \\
0 1 1 1 1 0 0 1 0 0 0 0 0 0 0 0 \\
1 0 1 1 1 0 1 1 0 0 1 0 0 0 0 0 \\
0 1 0 1 1 1 1 0 0 0 1 0 0 0 0 0 \\
0 0 1 0 1 0 0 0 0 0 0 1 0 0 0 0 \\
0 0 1 1 1 1 0 1 0 0 0 0 0 1 0 0 \\
0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 \\
0 0 0 1 1 1 0 0 0 0 0 0 0 0 0 1
\end{bmatrix}
\]

Let us assume an 8-bit data message \( m = [1 0 0 1 0 1 0 0] \). This data vector can be encrypted by reproducing it with originator matrix i.e. Then the resultant code word for Encrypting is

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

The code pattern is supposed to be legal, if the code vector fulfils the conditional design \( Z = C \cdot H^T = 0 \)

Bit Flipping algorithms are of the most efficient LDPC decrypting algorithms which have decent error concert and have minimal hardware decrypting complication [10].

**Steps for Decrypting of LDPC Codes Using Bit Flipping Algorithms:**

Bit Flipping algorithms are of the greatest efficient LDPC decrypting algorithms which have decent error concert, and has minimal hardware computational complexity. The following steps reveals the process for decrypting of LDPC codes using BF algorithms. Brief explanation/ steps are offered below.

**Step 1:** Initially the received 32-bit code pattern and assigned to 32 variable nodes.

**Step 2:** Verify the Check node connections with all variable nodes and compute check node values.

**Step 3:** In step 3, majority check operations are performed for all variable nodes.

In this paper, Fig.4 shows the sequential hybrid LDPC decipherer which is used for Decrypting of LDPC codes. The decipherer consists of Input buffer (which grasps the code data), a buffer which is rollback (It grasps the outcome of complemented bits), an output buffer (It grasps the fault free data bits), check and variable nodes, a flipped unit (Which flips the bits of data, when limits are not met), a variable hub unit (VPU), and a check unit hub (CUH), and shift registers (these are used to alter the data bits as the reiteration starts) are used [11]-[16].

**Conclusion**

The LDPC codes have the near Shannon limit presentation and the creation of the equivalence check matrices is clarified in this paper for the binary LDPC codes. Where the methodology of creating the LDPC Codes and its corresponding general decoding block diagram of LDPC codes is also explained. General Decoding scheme for bit flipping algorithms is also elaborated. Hence, these LDPC codes have higher coding rates compared to other conventional codes available. From these construction, we conclude that these codes are used in Security and Signal processing applications which provides authentication between source and destination.
Fig. 4. Block Diagram of BF Decoder for EG-LDPC Codes

References