CHAPTER 5

MAJORITY LOGIC DECODING OF 2D PG-LDPC CODES

5.1 INTRODUCTION

The essence of the MLD scheme as given in [2],[80],[82],[84] is that the parity checks on a particular received bit cast “votes” to decide the correctness of that bit. A simple majority of these votes can be used to determine the error pattern. For each bit position, this scheme requires, the parity check equation should form an orthogonal set. The PG LDPC codes satisfy this important orthogonality requirement by virtue of their construction.

Consider an (n, k) PG-LDPC code C over PG(2, 2^s). C is in the null space of incidence matrix of PG(2, 2^s). The incidence matrix is also called parity check matrix of C and is denoted as H. If v is a vector in C, then its inner product over GF(2) with any row vector w of H will be equal to 0 i.e.,

\[ v \cdot w = 0 \]  \hspace{1cm} (5.1)

The relation (5.1) is known as parity check equation.

Assume the received vector \( r = v + e \) where \( e = (e(0), e(1), \ldots, e(n-1)) \). For any vector w in H. The parity check sum can be formed by the linear sum over GF(2) of the received digits.

\[ A = w(0)r(0) + w(1)r(1) + w(2)r(2) + \ldots + w(n-1)r(n-1) \]  \hspace{1cm} (5.2)

A is called the parity check sum. If received vector r is a valid code vector, then the parity check Matrix A will be zero. Otherwise,

\[ A = w(0)e(0) + w(1)e(1) + \ldots + w(n-1)e(n-1) \]  \hspace{1cm} (5.3)
If coefficient $w(i) = 1$, the error digit $e(i)$ is checked by the checksum $A$. Consider the $J$ row vectors of matrix $H$ having the following properties:

1. $(n-1)^{th}$ coefficient of each vector is 1.
   
   That is, $w(1, n-1) = w(2, n-1) = \ldots = w(J, n-1) = 1$

2. For $i$ not equal to $n-1$, there is at most one vector whose $i^{th}$ coefficient is 1. If $w(1, i) = 1$, then rest of them will be zero;

Then, one concludes that $J$ vectors are orthogonal on the digit $(n-1)$. From (5.2) and the above $J$ orthogonal vectors one can get,

$$w_i \cdot r = w(i, 0)*r(0) + w(i, 1)*r(1) + \ldots + w(i, n-1)*r(n-1)$$

for $0 \leq i \leq J$

This implies

$$w_i \cdot r = \text{summation of the products} \ (w(i, 0)*e(0)) + \ldots \ (upto \ (n-1) \text{terms})$$

for $0 \leq i \leq J$.

Hence, All the check sums have checked the error digit $e(n-1)$. Other error digits are checked by at most one check sum. The important point is that these $J$ check sums help in detecting the error on the digit $e(n-1)$ because they are orthogonal on that particular digit. Thus we can write,

$$A_j = e(n-1) + \sum e(i), i \text{ not equal to } n-1 \text{ and for } 0 \leq j \leq J$$

If all the error digits found in the sum $A_j$ are zero for $i$ not equal to $n-1$, the value of $e(n-1) = A_j$. Using this clue, parity check sums which are orthogonal on the error digit $e(n-1)$ are used to compute the estimation of $e(n-1)$ or in other words, one can decode the received digit $r(n-1)$. Assuming the number of errors less than or equal to
floor(J/2) in error vector from e(0) to e(n-1). If e(n-1) = 1, other errors can distribute among the floor(J/2)-1 check sums which are orthogonal on e(n-1). Therefore, at least (J - floor(J/2)+1) check sums or more than half the check sums being orthogonal on e(n-1) are equal to e(n-1) = 1. Suppose e(n-1) = 0, then the other errors can distribute among floor(J/2) number of check sums which are orthogonal on e(n-1). Therefore, at least (J - floor(J/2)) or , one can say at least half check sums orthogonal on e(n-1) are equal to e(n-1) = 0. Based on the above facts an algorithm for decoding e(n-1) is given as:

The error digit e(n-1) is assumed to be 1 if the majority of the check sums which are orthogonal on e(n-1) = 1. Otherwise it is decoded as 0. Hence, the value of error digit e(n-1) is the value obtained by voting of the parity check sums that participated in voting. Next, consider the choice of J rows of H matrix of PG_LDPC codes over GF(2, 2^s) which are orthogonal on digit (n-1).

5.2 PROPERTIES OF H MATRIX OF PG-LDPC CODES OVER PG(2, 2^s)

The H matrix of PG-LDPC codes over PG(2, 2^s) is the incidence matrix of PG(2, 2^s) which establishes incidence relation between points and lines of the projective space. There are J = 2^s+1 lines intersecting at a given point. Between any two lines of PG(2, 2^s) there is only one point in common. Thus for the set J since the point of intersection is the common point, there are no other common points between any two lines. Thus the union of points of the set J = (2^s+1)*number of points on a line - 2^s = (2^s+1)*(2+1) - 2^s = 2^{2s}+2^s+1. Thus
the union covers all the distinct points of the projective space implying it covers all the digits of the received vector. If one chooses the set $J$ as the $2^s+1$ lines intersecting at point $(\lambda^{(2^2s+2^s)})$ of projective space, then the set $J$ will satisfy all the properties for majority logic decoding. Thus to design circuitry for majority logic decoding of PG-LDPC codes over $\text{PG}(2, 2^s)$ we should take the following approach:

1. Get the set $J$ of rows of $H$ matrix which have 1s in position $2^{2s+2^s}$

2. Each gate corresponds to a line of the set $J$. Input to each gate is the set of digits of the received vector corresponding to the labels of the points belonging to the line.

The Matlab function **GateDesign** computes the set $J$ for a given ‘s’ and decides on the gate inputs. Input to the algorithm is the value ‘s’ and the $H$ matrix

PSEUDOCODE :function [J, GateInputs] = GateDesign(H_MATRIX)

- $J = 0$;
- GateInputs = 0;
- disp('Vectors which are orthogonal On $2^{2s+2^s+1}$')
- cnt = 1;
- for i = 1:n

  ref = H_MATRIX(i, :);
  if(H_MATRIX(i, n) == 1)
  J(cnt, 1:n+1) = [cnt-1 H_MATRIX(i, :)];
  cnt1 = 1;
  end

end
for j = 1:n
    if(ref(j) == 1)
        GateInputs(cnt,cnt1) = j-1;
        cnt1 = cnt1 + 1;
    end
end

cnt = cnt + 1;

end
end

disp('Gate Inputs Exponents Of The Points Of The Projective Space')

[a, b] = sort(GateInputs(:, 1));
temp = [];
for i = 1:length(a)
    temp(i, :) = GateInputs(b(i), :);
end

GateInputs = temp

Output for s = 2

Vectors Which Are Orthogonal On $2^{2s+2^s+1}$

J =

0 0 0 1 1 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 1 0 0 1 1 0 0 0 0 0 1 0 1
2 0 1 0 0 0 0 0 0 0 0 1 0 0 1 1 0 0 0 0 0 1
3 0 0 0 0 1 0 1 0 0 0 0 0 0 0 0 0 0 1 0 0 1 1
4 1 0 0 0 0 1 0 1 0 0 0 0 0 0 0 0 0 1 0 0 1
**Gate inputs - exponents of the points of the projective space**

GateInputs (Refer to Fig 5.1) =

<p>| | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>5</td>
<td>7</td>
<td>17</td>
<td>20</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>11</td>
<td>14</td>
<td>15</td>
<td>20</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>3</td>
<td>8</td>
<td>10</td>
<td>20</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>6</td>
<td>16</td>
<td>19</td>
<td>20</td>
</tr>
<tr>
<td>5</td>
<td>9</td>
<td>12</td>
<td>13</td>
<td>18</td>
<td>20</td>
</tr>
</tbody>
</table>

**EXEMPLARY OF THE POINTS OF THE PROJECTIVE SPACE ORTHOGONAL ON $2^{2s} + 2^s + 1 = '20'$$**

![Diagram](image)

Fig.5.1 Lines orthogonal On Point $2^{2s} + 2^s + 1$

### 5.3 MAJORITY LOGIC DECODER

The decoder’s structure is shown in Fig.5.2: The $r(X)$ is the received vector in polynomial form. $CS_1$, $CS_2$,........$CS_J$ are the J check sums.
5.4 THE ERROR CORRECTION PROCEDURE

1) Turn on gate1 while gate2 is turned off. The buffer register is used to store the received vector $r(X)$. This vector is fed into the buffer register having the length equal to the total length of the message vector. The first received digit is $r(n-1)$.

2) Form the J parity check sums which are orthogonal on $e(n-1)$ by adding the relevant received digits over $GF(2)$.

3) The orthogonal check sums ($J$) are applied to the gate of majority logic scheme. The digit $r(n-1)$ emerging out from the buffer is corrected based on the resultant output of the majority logic gate.
At the end of the above step, the contents of the buffer register have been shifted by one place to the right side with gate 2 'on'. The buffer register at this point, contains the second received digit at the appropriate i.e., rightmost stage of the buffer register. This is corrected in exactly the same way as the first received digit was. The steps two and three are repeated by the decoder.

5) Decode digit by digit until the n shifts are over

5.4.1 Interpreting The Final Contents Of The Register:

For the majority logic to work, the received vector should contain \([J/2]\) or fewer errors, then the buffer contains the correct (transmitted) code vector. At this point, the inputs to the majority gate are supposed to be all zero. If not all the inputs to the majority gate are zero, an uncorrectable error pattern is detected. At each step of orthogonalization, the errors sums needed for the next step can be determined only if \([J/2]\) or fewer errors occur in Majority logic decoding. Refer to [2] equation 8.22, page 244.

The Matlab function \(y = \text{MLD}(RX, H\text{\_MATRIX})\) implements the majority logic decoding scheme of figure 5.1. Inside it calls the Matlab function \([J, \text{GateTaps}] = \text{GateInputs}(H\text{\_MATRIX})\) to get gate input details.

PSEUDOCODE: Function \(y = \text{MLD}(RX, H\text{\_MATRIX})\)

- \(y = \text{zeros}(1, n);\)

Get Gate Inputs

- \([J, \text{GateInputs}] = \text{GateDesign}(H\text{\_MATRIX})\)

Initialize Rxbuf SR

- \(\text{RXBUF} = \text{zeros}(1, n);\)

Load Rx
• RXBUF = RX;
• y = zeros(1, n);

Error Correction
• for i = 1:n
  acc = 0;
  for j = 1:length(J(:, 1))
    taps = GateInputs(j, 2:length(GateInputs(1, :))); 
    acc = acc + rem(sum(RXBUF(taps+1)), 2);
  end
  if(acc >= ceil(length(J(:, 1))/2))
    res = 1;
  else
    res = 0;
  end
• a = rem(RXBUF(n)+res, 2);
• y(n-i+1) = a;
• temp = RXBUF(1:n-1);
• RXBUF = [a temp];
• end

5.4.2 An Example Program for Majority Logic Decoding

Example Program For Majority Logic Decoding

Get Information Vector IX
• disp('Information Vector Of Length K')
• IX = randint(1, k, [0 1])

Encode and Get Codevector C
• disp('CODEVECTOR CX')
• CX = rem(IX*G_MATRIX, 2)

*Get Error Vector Ex*

• disp('Error Vector EX')
• EX = randerr(1, n, floor(length(J(:, 1))/2))

*Get Received Vector*

• disp('Received Vector RX')
• RX = rem(CX + EX, 2)
• y = MLD(RX, H_MATRIX)

Output of the Program ExampleMLD

Information Vector Of Length K

IX = 0 1 0 1 0 0 1 0 1 0 0

Code Vector CX

CX = 0 1 0 0 0 0 1 0 1 1 1 1 1 0 0 1 1 1 1 0 1 0 0

Error Vector EX

EX = 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 1 0 0 0

Received Vector RX

RX = 0 1 0 0 0 0 1 0 1 1 1 0 0 0 1 1 1 1 1 1 0 0

y = 0 1 0 0 0 0 1 0 1 1 1 1 1 0 0 1 1 1 0 1 0 0
5.5 CONCLUSIONS

In this chapter, the majority logic decoding scheme is discussed for correcting the received vector $r(X)$ of a PG-LDPC Code over PG$(2, 2^s)$. The simulation is done by developing the following Matlab codes:

1. $[J, \text{GateTaps}] = \text{GateInputs}(H\_MATRIX)$
2. $[y] = \text{MLD}(RX, H\_MATRIX)$

These programs are used in decoding PG codes based on $p$-adic integers detailed in the next chapter.