CHAPTER 7
ENCODER AND DECODER IMPLEMENTATION

7.1 INTRODUCTION

The encoder and decoder of the PG-LDPC codes is described. The encoder design is a standard method available in the literature. This is briefly covered for completeness, but as such the decoder design is the main focus of the thesis which is a novel approach.

7.2 ENCODER

The encoder implements the code vector \( C_X = I_X \times G_{\text{MATRIX}} \) where \( I_X \) is the length \( k \) information vector over \( \mathbb{Z}_q \). The encoder can be designed using \( k \) or \( n-k \) shift registers over \( \mathbb{Z}_q \) with feedback determined by the generator polynomial \( g(X) \) of PG\((2, 2^t, 2^s)\) over \( \mathbb{Z}_q \).

Since the code is cyclic any serial scheme based on shift registers that are applicable for encoding binary/symbol based cyclic codes can be employed for the encoder design. Instead of Galois field additions and multiplications, \( p \)-adic integer addition and multiplication are used. Also shift registers store \( p \)-adic integers. Figure 7.1 depicts the generic structure of the encoder. Let the feedback coefficients be \( a_0 \) to \( a_{n-k} \). The additions and multiplications are over modulo \( 2^t \). In the figure, \( I_X \) and \( C_X \) are the information and the encoded vectors/polynomials.
The encoder architecture is discussed in many standard text books. Hence, further discussion is not necessary.

7.3 Strategic Decoding Procedure

For the application, an appropriate simple majority logic decoding is used to simplify the complexity. Since majority logic is applied in binary domain, one can implement a sequential method to decode the received vector $R(X)$ over $\mathbb{Z}_q$. The implementation is bit serial in nature which is well suited for VLSI architecture.

7.3.1 Strategy to Formulate The Sequential Decoding

Let the information polynomial of degree $k-1$ be given by

$I(X) = \sum_{j=0}^{k-1} i_j X^j$ for $0 \leq j \leq k-1$.

Then $I(X)$ can also be written as

$$I(X) = i_{00} + i_{01} X + i_{02} X^2 + \ldots + i_{0,k-1} X^{k-1}$$

$$+ 2^{\left[ i_{10} + i_{11} X + i_{12} X^2 + \ldots + i_{1,k-1} X^{k-1}\right]}$$

$$+ 2^2 \left[i_{20} + i_{21} X + i_{22} X^2 + \ldots + i_{2,k-1} X^{k-1}\right]$$
\[ + 2^{k-1}[i_{t-10} + i_{t-11}X + i_{t-12}X^2 + \ldots + i_{t-1k-1}X^{k-1}] \]  

...(7.1)

where \( i_{jl} \in Z_2 \) for \( 0 \leq j \leq k-1 \) and \( 0 \leq l \leq k-1 \).

Then, \( I(X) \times G_{\text{MATRIX}} = \)

\[ \{[i_{00} + i_{01}X + i_{02}X^2 + \ldots + i_{0k-1}X^{k-1}] \times G_{\text{MATRIX}} \]  

\[ + 2[i_{10} + i_{11}X + i_{12}X^2 + \ldots + i_{1k-1}X^{k-1}] \times G_{\text{MATRIX}} \]  

\[ + 2^2[i_{20} + i_{21}X + i_{22}X^2 + \ldots + i_{2k-1}X^{k-1}] \times G_{\text{MATRIX}} \]  

\[ \ldots \]  

\[ + 2^{k-1}[i_{t-10} + i_{t-11}X + i_{t-12}X^2 + \ldots + i_{t-1k-1}X^{k-1}] \times G_{\text{MATRIX}} \} \]  

...(7.2)

mod \( Z_q = C(X) \) the code polynomial of degree \( n-1 \).

Assume the code word

\( C(X) = c_0 + c_1X + \ldots + c_{n-1}X^{n-1} \) where \( c_j \) belongs to \( Z_q \) for \( j \) from 0 to \( n-1 \).

Then \( C(X) \) can also be written as

\( C(X) = c_{00} + c_{01}X + c_{02}X^2 + \ldots + c_{0n-1}X^{n-1} \)

\[ + 2[c_{10} + c_{11}X + c_{12}X^2 + \ldots + c_{1k-1}X^{k-1}] \]  

\[ + 2^2[c_{20} + c_{21}X + c_{22}X^2 + \ldots + c_{2n-1}X^{k-1}] \]  

\[ \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots

where \( c_{jl} \in Z_2 \) for \( 0 \leq j \leq n-1 \) and \( 0 \leq l \leq n-1 \).
\[ C(X) \mod(2) = \{ [i(X)] * G_{\text{MATRIX}} \} \mod(2) \]

\[ = c_{00} + c_{01}X + \ldots + c_{0, n-1}X^{n-1}. \quad \text{Thus} \]

knowing \( c_{00} + c_{01}X + c_{02}X^2 + \ldots + c_{0, n-1}X^{n-1}, \quad i_{00} + i_{01}X + i_{02}X^2 + \ldots + i_{0, k-1}X^{k-1} \) can be obtained by the relation

\[ (i_{00} + i_{01}X + i_{02}X^2 + \ldots + i_{0, k-1}X^{k-1}) \cdot g(X) \mod X^{n-1}. \quad \ldots \{(7.4)\} \]

This is implemented by an \( n-k \) shift register binary divider circuit with \( c_{00} + c_{01}X + c_{02}X^2 + \ldots + c_{0, n-1}X^{n-1} \) as the dividend polynomial and \( g(X) \) as the divider polynomial. \( g(X) \) is the generator polynomial of \( \text{PG}(2, 2^t, 2^s) \mod(2) \). Feedback taps of the divider circuit is determined by \( g(X) \). The \( (n-k) \) is degree of the generator polynomial

Residual \( C(X) = C(X) - (i_{00} + i_{01}X + i_{02}X^2 + \ldots + i_{0, k-1}X^{k-1}) \cdot G_{\text{MATRIX}} \)

\[ = 2\{c_{10} + c_{11}X + c_{12}X^2 + \ldots + c_{1, k-1}X^{k-1}\} \]

\[ + 2^2\{c_{20} + c_{21}X + c_{22}X^2 + \ldots + c_{2, n-1}X^{n-1}\} \]

\[ \ldots \ldots \ldots \]

\[ + 2^{t-1}\{c_{t-10} + c_{t-11}X + c_{t-12}X^2 + \ldots + c_{t-1, n-1}X^{n-1}\}. \quad \ldots \{(7.5)\} \]

Thus Residual \( C(X)/2 = \{c_{10} + c_{11}X + c_{12}X^2 + \ldots + c_{1, k-1}X^{k-1}\} \)

\[ + 2^1\{c_{20} + c_{21}X + c_{22}X^2 + \ldots + c_{2, n-1}X^{k-1}\} \]

\[ \ldots \]

\[ + 2^{t-2}\{c_{t-10} + c_{t-11}X + c_{t-12}X^2 + \ldots + c_{t-1, n-1}X^{n-1}\}. \quad \ldots \{(7.6)\} \]
This implies \((\text{Residual } C(X)/2) \mod (2) = [c_{10} + c_{11}X + c_{12}X^2 + \ldots + c_{1-k-1}X^{n-1}]\). Knowing \([c_{10} + c_{11}X + c_{12}X^2 + \ldots + c_{1-k-1}X^{n-1}]\), \([i_{10} + i_{11}X + i_{12}X^2 + \ldots + i_{1-k-1}X^{n-1}]\) is obtained by the divider circuit with \([c_{10} + c_{11}X + c_{12}X^2 + \ldots + c_{1-k-1}X^{n-1}]\) as the dividend polynomial and \(g(X)\) as the divider polynomial.

The up-dation Residual is

\[
C(X) = \text{Residual } C(X) - 2[i_{10} + i_{11}X + i_{12}X^2 + \ldots + i_{1-k-1}X^{n-1}] \cdot G_{\text{MATRIX}}
\]

\[
= 2^2[c_{20} + c_{21}X + c_{22}X^2 + \ldots + c_{2-k-1}X^{n-1}]
\]

\[\ldots\]

\[+ 2^{k-1}[c_{t-10} + c_{t-11}X + c_{t-12}X^2 + \ldots + c_{t-1-k-1}X^{n-1}].\] \hspace{1cm} \ldots(7.7)

By similar logic, decode other components of \(I(X)\). After \(t\) steps Residual \(C(X)\) = all zero polynomial of degree \(n-1\). The components of \(I(X)\) namely \([i_{0} + i_{1}X + i_{2}X^2 + \ldots + i_{k-1}X^{k-1}]\), \([i_{10} + i_{11}X + i_{12}X^2 + \ldots + i_{1-k-1}X^{n-1}]\), \([i_{20} + i_{21}X + i_{22}X^2 + \ldots + i_{2-k-1}X^{n-1}]\), \(\ldots\) \([i_{t-10} + i_{t-11}X + i_{t-12}X^2 + \ldots + i_{t-1-k-1}X^{n-1}]\) when combined as

\[
[i_{0} + i_{1}X + i_{2}X^2 + \ldots + i_{k-1}X^{k-1}]
\]

\[+ 2[i_{10} + i_{11}X + i_{12}X^2 + i_{1-k-1}X^{n-1}]\]

\[+ 2^2[i_{20} + i_{21}X + i_{22}X^2 + \ldots + i_{2-k-1}X^{n-1}]\]

\[+ \ldots\]

\[+ 2^{t-1}[i_{t-10} + i_{t-11}X + \ldots + i_{t-1-k-1}X^{n-1}].\] \hspace{1cm} \ldots(7.8)

results in the information vector \(I(X)\).
The above observation is the key for sequential decoding of \(R(X)\) polynomial using majority logic decoding technique. The two steps in decoding are

1. Correcting the received polynomial \(R(X)\) of degree \(n-1\) to obtain the \(C(X)\) polynomial.

2. Obtain the Information Vector

Let \(R(X) = C(X) + E(X)\) where \(E(X)\) is a polynomial of degree \(n-1\) representing channel induced errors. Assume the received vector as \(R(X) = r_0 + r_1X + r_2X^2 + \ldots r_{n-1}X^{n-1}\), \(r_j\) belongs to \(\mathbb{Z}_q\) for \(j\) between \(0\) to \(n-1\). Then \(R(X)\) can also be written as

\[
R(X) = r_{00} + r_{01}X + r_{02}X^2 + \ldots r_{0\ n-1}X^{n-1} \\
+ 2[r_{10} + r_{11}X + r_{12}X^2 + \ldots r_{1\ k-1}X^{k-1}] \\
+ 2^2[r_{20} + r_{21}X + r_{22}X^2 + \ldots r_{2\ n-1}X^{n-1}] \\
\ldots.. \\
+ 2^{t-1}[r_{t-10} + r_{t-11}X + \ldots + r_{t-1\ n-1}X^{n-1}].
\]

...(7.9)

where \(r_{ji} \in \mathbb{Z}_2\) for \(0 < j < n-1\) and \(0 < i < n-1\). Similarly let \(E(X) = e_0 + e_1X + e_2X^2 + \ldots e_{n-1}X^{n-1}\) where \(e_j \in \mathbb{Z}_q\) for \(0 < j < n-1\).

\(E(X)\) can also be written as

\[
E(X) = e_{00} + e_{01}X + e_{02}X^2 + \ldots e_{0\ n-1}X^{n-1} \\
+ 2[e_{10} + e_{11}X + e_{12}X^2 + \ldots e_{1\ k-1}X^{k-1}] \\
+ 2^2[e_{20} + e_{21}X + e_{22}X^2 + \ldots e_{2\ n-1}X^{n-1}] \\
\]
$\ldots\ldots$

$$+2^{t-1}[e_{t-10} + e_{t-11}X+e_{t-12}X^2+\ldots e_{t-1}X^{n-1}]$$

...(7.10)

where $e_{ji} \in Z_2$ for $0 \leq j \leq n-1$ and $0 \leq l \leq n-1$.

Let $e_j \in Z_q$ be one of the non zero elements of $E(X)$. This indicates an error in position $j$ of $R(X)$. Note: position/vector elements are designated from 0 to $n-1$ to be in one-to-one correspondence with polynomial representation. Then

$$e_j = e_{j0} + 2e_{j1} + 2^2e_{j1} + \ldots 2^{(t-1)}e_{jt-1}$$

where $e_{jl} \in Z_2$ ....(7.11)

$$R(X) \mod(2) = (r_0 + r_1X+ r_2X^2+\ldots r_{n-1}X^{n-1}) \mod(2)$$

$$= r_{00} + r_{01}X+ r_{02}X^2+\ldots r_{0\cdot n-1}X^{n-1}$$

$$=[(c_{00} + c_{01}X+ c_{02}X^2+\ldots c_{0\cdot n-1}X^{n-1})$$

$$+\{ e_{00} + e_{01}X+ e_{02}X^2+\ldots e_{0\cdot n-1}X^{n-1}\}] \mod(2).$$

...(7.12)

When $r_{00} + r_{01}X+ r_{02}X^2+\ldots r_{0\cdot n-1}X^{n-1}$ is subjected to majority logic decoding it detects an error $e_{0j}$ in position $j$. Then $e_{0j}$ is XOR-ED with $r_{0j}$ to apply error correction in position $j$. $r_{00} + r_{01}X+ r_{02}X^2+\ldots r_{0\cdot n-1}X^{n-1}$ is reduced to $c_{00} + c_{01}X+ c_{02}X^2+\ldots c_{0\cdot n-1}X^{n-1}$. Obtain Partially Corrected $R(X)$ as Partially Corrected $R(X) = [R(X) - e_0] \mod Z_q$. Then extract $i_{00}+i_{01}X+i_{02}X^2+\ldots i_{0\cdot k-1}X^{k-1}$ from $c_{00} + c_{01}X+ c_{02}X^2+\ldots c_{0\cdot n-1}X^{n-1}$ and get Residual $R(X)$ as

Residual $R(X) = \{\text{Partially Corrected } R(X) - [i_{00}+i_{01}X+i_{02}X^2+\ldots i_{0\cdot k-1}X^{k-1}]\}

*G\_MATRIX* \mod Z_q
\[ \text{where } r_{jl} \in \mathbb{Z}_2 \text{ for } 0 \leq j \leq n-1. \]

\[(\text{Residual } R(X)/2) \mod (2) = r_{10} + r_{11}X + r_{12}X^2 + \ldots + r_{1\, k-1}X^{n-1}\]
is subjected to majority logic decoding. It detects an error \( e_{1j} \) in position \( j \). Then \( e_{1j} \) is XOR-ed with \( r_{1j} \) to apply error correction in position \( j \).

Now

\[ r_{10} + r_{11}X + r_{12}X^2 + \ldots + r_{1\, n-1}X^{n-1} \text{ reduced } c_{10} + c_{11}X + c_{12}X^2 + \ldots + c_{1\, n-1}X^{n-1}. \]

Obtain updated Partially Corrected \( R(X) = [\text{Residual } R(X) - 2e_{1j}] \mod Z_q \).

Extract

\[ i_{10} + i_{11}X + i_{12}X^2 + \ldots + i_{1\, k-1}X^{k-1} \text{ from } c_{10} + c_{11}X + c_{12}X^2 + \ldots + c_{1\, n-1}X^{n-1}. \]

and get updated Residue \( R(X) \) as

\[ \text{Residue } R(X) = \{ \text{Partially Corrected } R(X) - 2[i_{10} + i_{11}X + i_{12}X^2 + \ldots + i_{1\, k-1}X^{k-1}] \} \cdot G\_MATRIX \mod Z_q \]

\[ = 2^2[r_{20} + r_{21}X + r_{22}X^2 + \ldots + r_{2\, n-1}X^{k-1}] \]

...........

\[ +2^{t-1}[r_{t-10} + r_{t-11}X + \ldots + r_{t-1\, n-1}X^{n-1}]. \]........(7.14)
where \( r_{ji} \in \mathbb{Z}_2 \) for \( 0 \leq j \leq n-1 \).

Do it for all steps. In the next section, the bit serial implementation using the above strategy is described and also provide pseudo code in the form of Matlab function.

### 7.3.2 Bit Serial Implementation of the Decoder

**Register Details:** The decoder has 5 registers as shown in the fig. 7.2.

---

**Fig. 7.2 Serial Bit Architecture**
1. **RX_BUF**: This is a symbol register of length n to store the RX vector over $\mathbb{Z}_q$. Subsequently this is also used to store the modified RX vector during each cycle. The RX_BUF is a circular register with feedback from RX_BUF(n) to RX_BUF(1).

2. **SR0**: This is an n-bit shift register used for implementing majority logic decoder. The outputs of SR0(1) to SR0(n) are connected to $J = 2^s + 1$ majority gates. The connections are determined by the lines of the projective space $\text{PG}(2, 2^s)$ which intersect at the point $(\lambda^{2^s+2^s})$. SR0 is purposefully chosen to be a circular register. During a circular shift (SR0(n) + MLD output) modulo 2, which is essentially the corrected RX bit is fed back to SR0(1). Use following nomenclature for Matlab implementation.
   - MLD output = MLD-bit
   - Corrected RX bit = CorrectedBit

3. **SR1**: This is an n-k bit shift register having feedback connections determined by the binary generator polynomial $g(X)$. Let $g(X) = g_0 + g_1x + g_2x^2 + \ldots + g_{n-k}x^{n-k}$. It implements standard binary dividing circuit to produce quotient polynomial $Q(X)$ at the output. At each cycle $Q(X)$ output is given by $g_{n-k} \times SR1(n-k)$. The output produced is from MSB to LSB It means coefficient $q_{n-1}$ of $Q(X)$ appears first. Input to any SR1(i) is given by
   Input to $SR1(i) = (g_{n-k} \times SR1(n-k) \times g_{i-1}) + SR1(i-1)$ for $2 \leq i \leq n-k$
   Input to $SR1(1) = ((SR0(n) + MLD\ _bit) \ mod \ 2) * (g_{n-k} \times SR1(n-k) \times g_0)$  
   ... (7.15)
g(X) in this case is the modulo 2 component of the generator polynomial for the \( \text{PG}(2, 2^s, 2^t) \) code under consideration. Q(X) output is essentially the decoded IX vector. If correctly decoded the first n-k Q(X) output will be zeros because IX vector will be of length k.

Use following nomenclature for Matlab coding.

- \( g_{n-k} \cdot \text{SR1(n-k)} \) = multiplier.
- \((\text{SR0(n)}+\text{MLD output}) \mod 2) = \text{CorrectedBit}
- \( g_i \) is represented as \( g_i(i+1) \) for \( 0 \leq i \leq n-k \)

4. **SR2**: This is a memory register to store the \( k \)th row of generator matrix of \( \text{PG}(2, 2^s, 2^t) \) code over \( \mathbb{Z}_q \). Its purpose is to implement \( \text{IX} \cdot 2^{(t-1)} \cdot \text{G_MATRIX} \) where IX is the decoded binary IX vector at step \( t \). The operation is sequential and gets completed in \( n \) cycles. However operation starts only after \( n-k \) cycles because IX vector is of length \( k \). The SR2 register is augmented with necessary circuitry to implement the above encoding process. The output of the circuitry is used to erase the effect \( \text{IX} \cdot 2^{(t-1)} \cdot \text{G_MATRIX} \) from the contents of RX_BUF.

5. **IX_BUF**: This is an accumulator of length \( k \) over \( \mathbb{Z}_q \) which after \( t \) steps gives the decoded IX vector over \( \mathbb{Z}_q \). The following operation need to be executed at the end of every step:

\[
\text{IX register} = \text{IX register} + 2^{(t-1)} \text{binary IX vector at } t^{th} \text{ step}
\]

IX register is augmented with circuitry to do the above operation.
7.3.3 Implementation Details: A general approach is shown through the flow chart:

```
start

Initialize the values of s, p, n, k, t

i = 1

Initialise the value of SR0

Shift the register associated with decode to clear SR1

Load SR0 with binary vector and SR2 with g(x) vector

Test the received bits using Majority decoding for errors

Correct the errors by encoding bitwise

Decode the bits serially and then reencode using shifting operation

If i <= t

YES

y = IX_BUF

NO

stop
```

Fig.7.3 Flow chart for Decoding Scheme
The implementation involves t-steps and each step involves n cycles.
Initially R(X) vector is loaded to RX_BUF. IX_BUF is set to 0.

At the beginning of every step, for step = 1 to t the following initializations take place.

1. SR0 register is loaded with RX_BUF/2^(step-1) modulo 2. Circuit switch1 1/(2^(step-1)) is used for accomplishing division by 2^(step-1).
2. SR2 register is loaded with the last row of G_MATRIX.

At each step, during every cycle (unit time step) the following operations take place.

If MLD_bit of majority logic decoder is “1”

1. CorrectedBit = (MLD_bit+SR(n)) mod(2). This operation corrects the error at SR0(n). This is a combinational logic operation executed by the XOR-GATE1 CorrectedBit is fed as an input to the SR1 register which implements divider circuit. The sequence of CorrectedBits forms the dividend polynomial (corrected R(X)) for the divider circuit.
2. RX_BUF(n) = (RX_BUF(n)-2^{step-1}) mod(2^t). Let RX(n) have the 2-adic representation \( a_0+a_12+a_22^2+...+a_{t-1}2^{t-1} \). Then the operation nullifies the error at 2-adic position \( a_{step} \). This operation is known as store operation and the Circuit gives \( 2^{(step-1)} \) value.
3. Combinational logic outputs available at inputs to registers SR1(2) to SR1(n-k) of divider circuit at the beginning of each cycle is described by the Matlab program

*Register Outputs*

```matlab
multiplier = SR1(deg_gx)*gx(deg_gx+1);
for j = 1:deg_gx-1
    SR1OUT(j) = rem(SR1(j)+gx(j+1)*multiplier, 2);
end
b = rem(CorrectedBit + gx(1)*multiplier, 2);
```

4. For cycle counts > n-k, say n-k+j, j >=1, if the value of the multiplier is equal to “1” it implies that (k-j+1)st bit of Q(X) at the current step is “1” and the IX_REG is updated as

   \[ \text{IX}_\text{REG}(k-j+1) = \text{IX}_\text{REG}(k-j+1) + 2^{(\text{step}-1)} \]

5. The SR0 is rotated right to shift the CorrectedBit to the 1st bit-register of SR0. Outputs of SR1 bit-registers 1 to n-k-1 are shifted to SR1 bit-registers 2 to n-k. SR1 bit-register 1 will be loaded with the binary value b given by

   \[ b = (\text{CorrectedBit} + \text{gx}(1) \times \text{multiplier}) \mod 2 \]
RX_BUF is rotated by one. For cycles > n-k, say cycle n-k+j SR2 register is shifted left by 1 to bring the k-j+1 row of G_MATRIX to be used during next cycle. It should be noted that row shifting is from row k to row 1.

PSEUDOCODE: function [y] = bitserial(n, k, t, G_MATRIX, GateTaps, rx)

• deg_gx = n-k;
• gx = rem(G_MATRIX(1, 1:n-k+1), 2);

Stores I(X)

• IX_BUF = zeros(1, k);

Load Rx Into Rx_Buf

• RX_BUF = zeros(1, n)
• RX_BUF = rx;
• for ite = 1:t

Initial Clear

Shift Register Associated With Majority Logic Decoding

• SR0 = zeros(1, n);

Shift Register Associated With Ix-Decoder

• SR1 = zeros(1, n-k+1);

Initial Loading Of SR0 With Binary Vector
SR0 = \text{rem}(\text{RX\_BUF}/2^{(\text{ite}-1)}, 2);

\textit{Initial loading of SR2 with gx vector}

\[ \text{SR2} = \text{zeros}(1, \, n); \]

\[ \text{SR2}(1, \, k:n) = \text{G\_MATRIX}(1, \, 1:n-k+1); \]

\text{cnt} = 1;

\text{for} \ i = 1:n

\textit{Majority Voting}

\[ \text{majcnt} = 0; \]

\text{for} \ j = 1:\text{length}(\text{GateTaps(:, 1)})

\[ \text{majcnt} = \text{majcnt} + \text{rem}(\sum(\text{SR0}(\text{GateTaps}(j, 2:length (\text{GateTaps}(1, :))))), 2); \]

\text{end}

\text{if}(\text{majcnt} \geq \text{ceil}(\text{length}(\text{GateTaps(:, 1)})/2))

\[ \text{MLD\_bit} = 1; \]

\text{else}

\[ \text{MLD\_bit} = 0; \]

\text{end}

\[ \text{CorrectedBit} = \text{rem}(\text{SR0}(n)+\text{MLD\_bit}, 2); \]

\text{if}(\text{MLD\_bit} == 1)

\[ \text{RX\_BUF}(n-i+1) = \text{RX\_BUF}(n-i+1)-2^{(\text{ite}-1)}; \]

\text{if}(\text{RX\_BUF}(n-i+1) < 0)

\[ \text{RX\_BUF}(n-i+1) = \text{RX\_BUF}(n-i+1) + 2^{t}; \]
Update Operation For Decoding

\[
\text{multiplier} = \text{SR1(deg}\_\text{gx})*\text{gx(deg}\_\text{gx}+1);
\]

for \( j = 1:\text{deg}\_\text{gx}-1 \)

\[
\text{SR1OUT}(j) = \text{rem} (\text{SR1}(j)+\text{gx}(j+1)*\text{multiplier}, 2);
\]

end

\[
\text{b} = \text{rem} (\text{CorrectedBit} + \text{gx}(1)*\text{multiplier}, 2);
\]

Updating I(X) And Getting Residual R(X)

if\( (i > n-k) \)

if\( (\text{multiplier} == 1) \)

\[
\text{IX\_BUF}(k-\text{cnt}+1) = \text{IX\_BUF}(k-\text{cnt}+1) + 2^{\text{ite}-1};
\]

\[
\text{cnt} = \text{cnt} + 1;
\]

\[
\text{RX\_BUF} = \text{rem} (\text{RX\_BUF} - \text{SR2}(1, 1:n)*2^{\text{ite}-1}, 2^t);
\]

for \( l = 1:n \)

if\( (\text{RX\_BUF}(l) < 0) \)

\[
\text{RX\_BUF}(l) = \text{RX\_BUF}(l) + 2^t;
\]

end

end

else

\[
\text{cnt} = \text{cnt} + 1;
\]

end

end
Shifting

Majority Decoder

\[ SR0 = \text{[CorrectedBit SR0}(1, 1:n-1)]; \]

Ix Decoder

\[
\text{for } j = 1:deg_{gx}-1 \\
SR1(j+1) = SR1OUT(j); \\
\text{end} \\
SR1(1) = b;
\]

Shifting Operation For \([I(X)\text{Mod}(2)]^*G_{Matrix}\)

\[
\text{if}(i > n-k) \\
SR2 = [SR2(2:n) 0]; \\
\text{end} \\
\text{end} \\
\text{end}
\]

\[ y = IX\_BUF; \]

7.4 ANALYSIS OF THE ALGORITHM

The algorithm is bit serial and therefore is well suited for VLSI implementation. The algorithm involves \(t\) steps for \(PG(2, 2^t, 2^n)\) code and each step involves \(n\) cycles (i.e., the code length). In each cycle there is a constant \(k\) number of operations which include logical and arithmetic operations over \(GF(2)\) and \(p\)-adic integers \(Z_q\). Modulo operations over \(Z_q\) are just truncations for \(q = 2^t\) and therefore does
not involve extra cost. The value of k may be less than 10 and is independent of n. Thus the complexity of the algorithm is linear in n for a given t. This structure is well suited for error correction in nanoscale memories especially for byte error correction for which Reed-Solomon code is highly prohibitive because of its decoder complexity. Extension of the concept for byte error correction is done with a view to use them in applications to storage involving nanoscale memories.

7.5 APPLICABILITY TO NANOSCALE MEMORIES

The photolithographic fabrication scaling below 32 nanometers as mentioned in [6],[100] is challenged by technology and financial barriers. Silicon nanowires and carbon nanotubes which can be self assembled addresses these issues. Additionally, power and gating issues present in system architecture are also taken care. But at the same time, the assembling of these circuits is susceptible to very high defect rates and fault rates. Today’s microscale devices such as memories can be constructed with lithographic techniques exhibit error rates less than 1%, whereas the storage components constructed based on nanoscale elements such as carbon nanotubes and the single-crystal based semiconductor nanowires exhibit fault rates as high as 10%. This is the major concern. The research work concentrates on suggesting a replacement of high complex RS decoder with appropriate error detection and correction scheme at the nanoscale memory level given in [101],[102],[103] to simplify the complexity of the decoder and to provide dynamic error correcting
capability (for burst type of errors that are found while retrieving a word from a memory location) and also to provide desired accuracy of error correction by enhancing/switching the layered architecture appropriately.

Among the two types of errors such as static defects and soft errors, the soft error correction is a challenging job and is critical for nanoscale devices, storage (e.g., nanomemory), the computation devices (e.g., nano-ALU) or communication (e.g., signal transmitters and receivers at the nano scale level) devices. The research work concentrates on an architecture using nano-PLA blocks and simple nanogates (e.g., gates such as majoritygate, universal gates - nand/norgates) as design components. One needs to employ online error correction. However, for type of correction required, the old conventional correction techniques are not feasible because of their decoder complexity and non-scalability. For example conventional codes like RS or BCH codes achieve higher error-correcting capability using complex Galois field arithmetic. This is difficult to implement and not scalable. The complexity of Reed Solomon codes grows exponentially with the code length. Another prerequisite for the ECC applicable for nanoscale memories is reconfigurability.

As per the reliability behaviour of the circuitry, when the lifetime of a circuitry increases, the faults reduce in the normal period of operation of the circuit while at the infant(initial) mortality rate, and
the aging circuitry part, the faults increase. Hence, naturally, for error correction at the nanoscale memory, the scheme should do higher error-correction in the beginning and final stages, but disable the unused circuit parts of the decoder, thereby the power saving can be achieved. The observations are that the PG\((2, 2^s, 2^t)\) codes are amenable for reconfigurabilty. The analysis of the algorithm for PG\((2, 2^s, 2^t)\) codes suggests that they efficiently sort out the above issues and therefore become good candidates for being used in burst and random error corrections in nanoscale memories, in place of Reed Solomon Codes.

### 7.6 CONCLUSIONS

The sequential decoder design is proposed based on the bit serial implementation to handle the random error detection and correction particularly in nanoscale memory, since, the non linear complex decoding technique does not suit in this application. More importantly, the decoder design is very much suitable for VLSI implementation and has a progressive decoding approach based on the requirement.

Before presenting the results of the p-adic extended PG-LDPC decoder, It is possible to construct generic PG-LDPC decoder which can work for LDPC codes irrespective of their construction scheme is being regular or irregular. At the same time, this decoding can be used for fault tolerance using Perfect Difference Networks (PDN) as explained in the next chapter.