MODIFIED MATRIX CODES FOR SHIELDING MEMORIES AGAINST ADJACENT ERRORS

Neelima K*, C. Subhasb

aDepartment of ECE, School of Engineering, Mohan Babu University erstwhile Sree Vidyanikethan Engineering College, Tirupati, India
bChadalawada Ramanamma Engineering College, Tirupati, India

Graphical abstract

Abstract

Soft errors are caused in memories due to radiation effects as the technology scales down. This paper concentrates on correcting adjacent errors in memories using indirect decoding mechanisms. Several matrix codes were also used to correct a maximum of two adjacent and random errors. In this paper, matrix representation of two rows for half the data bits matrix representations is used and each row is encoded with extended hamming code parity bits. Three ways of decoding are proposed. Among them, the method-1 uses only extended hamming bits of rows and vertical parity bits for decoding which is capable of correcting odd number of adjacent bits in half of data bits. The method-2 uses all parity bits and is capable of correcting 1 bit less than half the data bits. The method-3 uses all parity bits and with a change in decoding mechanism allows correction of half erroneous data bits. The method-3 based decoder proves to be more reliable either in lower half or upper half of data, enabling it to be used in image processing applications. But method-3 compromises with decrease in code rate, increase in bit overhead, area and power delay product by at least 26.38%, 10.76%, 9.6% and 5%

Keywords: Bit Overhead, Code Rate, Code Efficiency, Hamming Code, Matrix Codes.

© 2024 Penerbit UTM Press. All rights reserved

1.0 INTRODUCTION

The soft errors gain prominence as the compact structures cause high radiation effects inducing random and adjacent errors. In many modern applications, there is a necessity of high reliability which encourages error detection and correction codes. The applications for which they can be used are military applications, medical applications, etc. The introduction of telemedicine to society needs higher reliability of data stored and communicated. In embedded systems customized for these applications store a lot of information by the way of encoded data [1,2,3].

The memories are the devices in embedded systems that are majorly prone to errors. Also the operating system is also stored in read only memory and the data used momentarily can be stored in random access memory [4,5]. The data that is to be stored is first encoded by parity bits and the information is stored as code word along with parity bits using write operation in encoder.

Depending upon whether ROM or RAM, the number of write operations are used. For ROM, the data is written once and read many times. But in RAM, many write and many read operations are used [6,7]. So in decoder, the code word is read and the parity bits are verified for error detection. The change or bit flips in parity predict the possibility of erroneous data. Then by using the specified algorithm, the error correction mechanism is performed to obtain error free data [8,9].

The model utilized for memory representation for error detection and correction is as shown in figure 1 where H indicates Horizontal and V indicates Vertical parity bits.
The basic EDAC codes [10] utilize the various blocks as shown in figure 2(a). The encoder uses two step process i.e., the HVHC (Horizontal Vector Hamming Code) Encoding and the discussed Encoders to generate code word.

This code word is written into the memory [11,12]. Using read operation, the data is decoded first by HVHC Decoder and later by Parity Decoder which verify the parity bits for correcting detected errors. The entire process of EDAC is based on Hamming codes for all the existing or proposed Parity codes in detail is shown in figure 2 (b).

Majorly the changes are incorporated in parity encoder and decoder blocks for EDAC Codes. The additional bits required depends on type of method adopted in encoder on the basis of Hamming Distance which aid in the development of code word that is written into memory [13,14].

Several error detection and correction codes have been proposed in literature which have gained importance in recent years. Among them the Hamming and extended Hamming codes are used as basis [15,16,17].

Upon these, matrix code and its modified versions show good reliability and have achieved correction capability of quad errors. Also Hsiao code, Golay Code, etc are also used for SEC-DED, SEC-DEC-TED and SEC-DEC-TEC [18,19,20]. This paper proposes modifications for decoding mechanism so as to obtain higher order reliability. Here one basic way of encoding is proposed and three versions of decoding are proposed.

### 2.0 METHODOLOGY

The basic mechanism of encoding that is used is as depicted in figure 1. Consider the 8-bit data being arranged as shown in figure 3. Here V[3...0] are the vertical encoded parity bits, R[5...0] are hamming bits and H[1...0] are the extended hamming code parity bits [21,22].

\[
\begin{align*}
D_7 & \oplus D_6 & D_5 & D_4 & H_1 & R_5 & R_4 & R_3 \\
D_3 & D_2 & D_1 & D_0 & H_0 & R_2 & R_1 & R_0 \\
V_3 & V_2 & V_1 & V_0 & & & & \\
\end{align*}
\]

The vertical parity bits are calculated as

\[ V_i = D_i \oplus D_{i+1} \ldots (1) \]

The hamming parity bits are calculated as

\[ R_0 = D_0 \oplus D_1 \oplus D_3 \ldots (2) \]
\[ R_1 = D_0 \oplus D_2 \oplus D_3 \ldots (3) \]
\[ R_2 = D_1 \oplus D_2 \oplus D_3 \ldots (4) \]
\[ R_3 = D_4 \oplus D_5 \oplus D_7 \ldots (5) \]
\[ R_4 = D_4 \oplus D_6 \oplus D_7 \ldots (6) \]
\[ R_5 = D_5 \oplus D_6 \oplus D_7 \ldots (7) \]

The extended hamming parity bits are calculated as

\[ H_0 = D_0 \oplus D_1 \oplus D_2 \oplus D_3 \ldots (8) \]
\[ H_1 = D_4 \oplus D_5 \oplus D_6 \oplus D_7 \ldots (9) \]

The code word consists of 20 bits and the positions are represented as shown below

\[
\begin{array}{cccccccccccccccc}
V_3 & V_2 & V_1 & V_0 & H_0 & R_0 & R_5 & R_4 & R_3 & R_2 & R_1 & R_0 & V_3 & V_2 & V_1 & V_0 \\
\end{array}
\]

The above code word is stored in memory with the specified location by write operation.

The decoder is designed by using the parity bits for error evaluation in data and correction to ensure reliability. The decoding mechanisms are developed in three ways represented by method-1, method-2 and method-3.

In method - 1, we use only the extended parity bits and vertical bits for decoding and the hamming bits remain as hidden bits and are not used. Consider the data to be "00000000" as the data to be encoded [23]. Then by using the encoding equations the corresponding parity bits are also all zeros as shown below. Then the code word is "00000000000000000000"

\[
\begin{align*}
D_7 & \oplus D_6 & D_5 & D_4 & \oplus & D_3 & \oplus & D_2 & \oplus & D_1 & \oplus & D_0 & \oplus & V_3 & \oplus & V_2 & \oplus & V_1 & \oplus & V_0 \\
\end{align*}
\]

If the same code word is read from memory, then there is no error in data. Suppose a single bit error exists say in $D_0 = 1$, then the code word changes as "00010000000000011110". As the error is reflected in parity bits $H_0$ and $V_0$, the decoding mechanism will be as $H_0 = 1$ and $V[3...0] = "0001"$, $D_{out}[3...0] = D_{read}[3...0] \oplus V[3...0] = "0000"$ and then $D_{out}[7...4] = D_{read}[7...4]$ then $D_{out} = "00000000"$, which is error free data. Suppose two adjacent bits are in error say in $D[1...0] = 1$, then the code word changes as "0011000000000011110". As the error is reflected in parity bits $V_0$ and $V_1$, the decoding mechanism will be as $H = "00"$ but $V[3...0] = "0011"$, the error may exist in either upper half or lower half of data read from memory then $D_{out}[3...0] = "00100011"$ and $D_{out}[7...4] = D_{read}[7...4] \oplus V[3...0] = "00001011"$, then $D_{out} = "00000000"$, which is error free data. Suppose two adjacent bits are in error say in $D[1...0] = 1$, then the code word changes as "0011000000000011110". As the error is reflected in parity bits $V_0$ and $V_1$, the decoding mechanism will be as $H = "00"$ but $V[3...0] = "0011"$, the error may exist in either upper half or lower half of data read from memory then $D_{out}[3...0] = "00100011"$ and $D_{out}[7...4] = D_{read}[7...4] \oplus V[3...0] = "00001011"$, then $D_{out} = "00000000"$, which is error free data.
Suppose three adjacent bits are in error say in \(D[2...0] = 1\), then the code word changes as “01110000000010110010”. As the error is reflected in parity bits \(V_0\), \(V_1\), \(V_2\), and \(V_3\), the decoding mechanism will be as \(\Delta H = 1\) and \(\Delta V[3...0] = "0011"\) and then \(D_{\text{out}}[7...4] = D_{\text{read}}[7...4]\) = “00000000”, which is error free data.

Suppose four adjacent bits are in error say in \(D[3...0] = 1\), then the code word changes as “111100000000111111”. As the error is reflected in parity bits \(V_0\), \(V_1\), \(V_2\), and \(V_3\), the decoding mechanism will be as \(\Delta H = 1\) and \(\Delta V[3...0] = "0011"\), the error may exist in either upper half or lower half of data read from memory then verify \(\triangle R = 000110\). As only LSB bits are changed then \(D_{\text{out}}[0] = D_{\text{out}}[1] = D_{\text{out}}[2] = D_{\text{out}}[3] = \Delta R[0] \oplus \Delta R[1] \oplus \Delta R[2] \oplus \Delta V[3] \oplus \Delta V[2] \oplus \Delta V[1] \oplus \Delta V[0] = 1\) and then \(D_{\text{out}}[7...4] = D_{\text{read}}[7...4] = "0000"\) \(\triangleright\) \(D_{\text{out}} = "00000000"\), which is error free data.

Suppose three adjacent bits are in error say in \(D[1...0] = 1\), then the code word changes as “001100000000111111”. As the error is reflected in parity bits \(V_0\), \(V_1\), \(V_2\), and \(V_3\), the decoding mechanism will be as \(\Delta H = 00\) but \(\Delta V[3...0] = "0011"\), the error may exist in either upper half or lower half of data read from memory then verify \(\triangle R = 000110\). As only LSB bits are changed then \(D_{\text{out}}[0] = D_{\text{out}}[1] = D_{\text{out}}[2] = D_{\text{out}}[3] = \Delta R[0] \oplus \Delta R[1] \oplus \Delta R[2] \oplus \Delta V[3] \oplus \Delta V[2] \oplus \Delta V[1] \oplus \Delta V[0] = 1\) and then \(D_{\text{out}}[7...4] = D_{\text{read}}[7...4] = "0000"\) \(\triangleright\) \(D_{\text{out}} = "00000000"\), which is error free data.

Suppose three adjacent bits are in error say in \(D[2...0] = 1\), then the code word changes as “01110000000010110010”. As the error is reflected in parity bits \(V_0\), \(V_1\), \(V_2\), and \(V_3\), the decoding mechanism will be as \(\Delta H = "00"\) but \(\Delta V[3...0] = "0011"\), the error may exist in either upper half or lower half of data read from memory then verify \(\triangle R = 000110\). As only LSB bits are changed then \(D_{\text{out}}[0] = D_{\text{out}}[1] = D_{\text{out}}[2] = D_{\text{out}}[3] = \Delta R[0] \oplus \Delta R[1] \oplus \Delta R[2] \oplus \Delta V[3] \oplus \Delta V[2] \oplus \Delta V[1] \oplus \Delta V[0] = 1\) and then \(D_{\text{out}}[7...4] = D_{\text{read}}[7...4] = "0000"\) \(\triangleright\) \(D_{\text{out}} = "00000000"\), which is error free data.

Suppose three adjacent bits are in error say in \(D[3...0] = 1\), then the code word changes as “111100000000111111”. As the error is reflected in parity bits \(V_0\), \(V_1\), \(V_2\), and \(V_3\), the decoding mechanism will be as \(\Delta H = "00"\) but \(\Delta V[3...0] = "1111"\), the error may exist in either upper half or lower half of data read from memory then verify \(\triangle R = 000110\). As only LSB bits are changed then \(D_{\text{out}}[0] = D_{\text{out}}[1] = D_{\text{out}}[2] = D_{\text{out}}[3] = \Delta R[0] \oplus \Delta R[1] \oplus \Delta R[2] \oplus \Delta V[3] \oplus \Delta V[2] \oplus \Delta V[1] \oplus \Delta V[0] = 1\) and then \(D_{\text{out}}[7...4] = D_{\text{read}}[7...4] = "0000"\) \(\triangleright\) \(D_{\text{out}} = "00000000"\), which is error free data.

Suppose four adjacent bits are in error say in \(D[3...0] = 1\), then the code word changes as “111100000000111111”. As the error is reflected in parity bits \(V_0\), \(V_1\), \(V_2\), and \(V_3\), the decoding mechanism will be as \(\Delta H = "00"\) but \(\Delta V[3...0] = "1111"\), the error may exist in either upper half or lower half of data read from memory then verify \(\triangle R = 000110\). As only LSB bits are changed then \(D_{\text{out}}[0] = D_{\text{out}}[1] = D_{\text{out}}[2] = D_{\text{out}}[3] = \Delta R[0] \oplus \Delta R[1] \oplus \Delta R[2] \oplus \Delta V[3] \oplus \Delta V[2] \oplus \Delta V[1] \oplus \Delta V[0] = 1\) and then \(D_{\text{out}}[7...4] = D_{\text{read}}[7...4] = "0000"\) \(\triangleright\) \(D_{\text{out}} = "00000000"\), which is error free data.
estimated is found to be even. This enables correction of 4 - erroneous bits.

Let R', H' and V' be the hamming, extended hamming and vertical parity bits calculated from data read from memory respectively. Then ΔR = R - R', ΔH = H - H' and ΔV = V - V'.

Suppose a single bit error exists say in D₀ = 1, then the code word changes as “00010000000010001111”. As the error is reflected in parity bits H₀', R₀', R₀' and V₀', the decoding mechanism will be as ΔH₀ = 1 and ΔV[3...0] = “0001”, D₀out[3...0] = D₀read[3...0] ⊕ ΔV[3...0] = “0000” and then D₀out[7...4] = D₀read[7...4] then D₀out = “00000000”, which is error free data.

Suppose two adjacent bits are in error say in D[1...0] = 1, then the code word changes as “001100000000000111111100”. As the error is reflected in parity bits H₁', R₁', R₀' and V₀', the decoding mechanism will be as ΔH[1] = “00” but ΔV[3...0] = “0001”, the error may exist in either upper half or lower half of data read from memory then verify ΔR = 0000011. As only LSB bits are changed then D₀out[3...0] = D₀read[3...0] ⊕ ΔV[3...0] = “0000” and then D₀out[7...4] = D₀read[7...4] = “0000” then D₀out = “00000000”, which is error free data.

Suppose three adjacent bits are in error say in D[2...0] = 1, then the code word changes as “00111000000000001111111111”. The error is reflected in parity bits H₂', R₂', R₁' and V₁', the decoding mechanism will be as ΔH[2] = “00” but ΔV[3...0] = “0001”, the error may exist in either upper half or lower half of data read from memory then verify ΔR = 00000011. As only LSB bits are changed then D₀out[3...0] = D₀read[3...0] ⊕ ΔV[3...0] = “0000” and then D₀out[7...4] = D₀read[7...4] = “0000” then D₀out = “00000000”, which is error free data.

Suppose four adjacent bits are in error say in D[3...0] = 1, then the code word changes as “01111000000000011111111111”. As the error is reflected in parity bits H₃', R₃', R₂' and V₂', the decoding mechanism will be as ΔH[3] = “00” but ΔV[3...0] = “0001”, the error may exist in either upper half or lower half of data read from memory then verify ΔR = 000000011. As only LSB bits are changed then D₀out[3...0] = D₀read[3...0] ⊕ ΔV[3...0] = “0000” and then D₀out[7...4] = D₀read[7...4] = “0000” then D₀out = “00000000”, which is error free data.

From five adjacent bits onwards being in error, the Dout can't correct errors and instead this method induces errors. The complete process is shown in table 3. From Table 3, it is clear that this method can correct up to four adjacent errors in 8-bit erroneous data read from memory.

<table>
<thead>
<tr>
<th>D₀read</th>
<th>H</th>
<th>R</th>
<th>V</th>
<th>Number of Errors Induced</th>
<th>D₀out</th>
<th>Number of Errors Corrected</th>
</tr>
</thead>
<tbody>
<tr>
<td>00000000</td>
<td>00</td>
<td>000000</td>
<td>0</td>
<td>0</td>
<td>000000000000</td>
<td>0</td>
</tr>
<tr>
<td>00000001</td>
<td>01</td>
<td>000011</td>
<td>0001</td>
<td>1</td>
<td>0000000000</td>
<td>0</td>
</tr>
<tr>
<td>00000011</td>
<td>00</td>
<td>0000110</td>
<td>0011</td>
<td>2</td>
<td>0000000000</td>
<td>0</td>
</tr>
<tr>
<td>00000111</td>
<td>01</td>
<td>000000</td>
<td>0111</td>
<td>3</td>
<td>0000000000</td>
<td>0</td>
</tr>
<tr>
<td>00001111</td>
<td>00</td>
<td>0000111</td>
<td>1111</td>
<td>4</td>
<td>0000000000</td>
<td>0</td>
</tr>
<tr>
<td>00111111</td>
<td>10</td>
<td>011111</td>
<td>1110</td>
<td>5</td>
<td>0111100000</td>
<td>0</td>
</tr>
<tr>
<td>00111110</td>
<td>00</td>
<td>111011</td>
<td>1000</td>
<td>6</td>
<td>111101111</td>
<td>0</td>
</tr>
<tr>
<td>01111111</td>
<td>10</td>
<td>000111</td>
<td>1000</td>
<td>7</td>
<td>111101111</td>
<td>0</td>
</tr>
<tr>
<td>11111111</td>
<td>00</td>
<td>111111</td>
<td>0000</td>
<td>8</td>
<td>111111111</td>
<td>0</td>
</tr>
</tbody>
</table>

### 3.0 RESULTS AND DISCUSSION

The designs are modeled in Verilog HDL and are verified in Xilinx ISE 14.5 Tool for 28nm Zynq FPGA with part number XC7Z100-2FFG1156.

The assessment of these methods is done for 8, 16, 32 and 64 – Bit Data. The comparison is as shown in table 4.

From table 4(a), method-3 of decoding is capable of correcting a maximum of 4 adjacent errors i.e., either in upper half or lower half of data read from memory with a code rate of 40%.

<table>
<thead>
<tr>
<th>Parameter/ Methods</th>
<th>Method-1</th>
<th>Method-2</th>
<th>Method-3</th>
</tr>
</thead>
<tbody>
<tr>
<td># Data Bits, k</td>
<td>8</td>
<td>8</td>
<td>8</td>
</tr>
<tr>
<td># Parity Bits, r</td>
<td>6</td>
<td>12</td>
<td>12</td>
</tr>
<tr>
<td># Code Word, n/k+r</td>
<td>14</td>
<td>20</td>
<td>20</td>
</tr>
<tr>
<td>Bit Overhead, r/k</td>
<td>0.75</td>
<td>1.5</td>
<td>1.5</td>
</tr>
<tr>
<td>Code Rate, k/n</td>
<td>0.57</td>
<td>0.4</td>
<td>0.4</td>
</tr>
<tr>
<td>Code Efficiency, r/n</td>
<td>0.43</td>
<td>0.6</td>
<td>0.6</td>
</tr>
<tr>
<td>Correction Capability</td>
<td>Only odd number up to 3 Adjacent Errors</td>
<td>Up to 3 Adjacent Errors</td>
<td>Up to 4 Adjacent Errors</td>
</tr>
</tbody>
</table>

From table 4(b), method-3 of decoding is capable of correcting a maximum of 8 adjacent errors i.e., either in upper half or lower half of data read from memory with a code rate of 47%.

<table>
<thead>
<tr>
<th>Parameter/ Methods</th>
<th>Method-1</th>
<th>Method-2</th>
<th>Method-3</th>
</tr>
</thead>
<tbody>
<tr>
<td># Data Bits, k</td>
<td>16</td>
<td>16</td>
<td>16</td>
</tr>
<tr>
<td># Parity Bits, r</td>
<td>10</td>
<td>18</td>
<td>18</td>
</tr>
<tr>
<td># Code Word, n/k+r</td>
<td>26</td>
<td>34</td>
<td>34</td>
</tr>
<tr>
<td>Bit Overhead, r/k</td>
<td>0.625</td>
<td>1.125</td>
<td>1.125</td>
</tr>
<tr>
<td>Code Rate, k/n</td>
<td>0.515</td>
<td>0.47</td>
<td>0.47</td>
</tr>
<tr>
<td>Code Efficiency, r/n</td>
<td>0.385</td>
<td>0.53</td>
<td>0.53</td>
</tr>
<tr>
<td>Correction Capability</td>
<td>Only odd number up to 7 Adjacent Errors</td>
<td>Up to 7 Adjacent Errors</td>
<td>Up to 8 Adjacent Errors</td>
</tr>
</tbody>
</table>

From table 4(c), method-3 of decoding is capable of correcting a maximum of 16 adjacent errors i.e., either in upper half or lower half of data read from memory with a code rate of 53%.

<table>
<thead>
<tr>
<th>Parameter/ Methods</th>
<th>Method-1</th>
<th>Method-2</th>
<th>Method-3</th>
</tr>
</thead>
<tbody>
<tr>
<td># Data Bits, k</td>
<td>32</td>
<td>32</td>
<td>32</td>
</tr>
<tr>
<td># Parity Bits, r</td>
<td>18</td>
<td>28</td>
<td>28</td>
</tr>
<tr>
<td># Code Word, n/k+r</td>
<td>50</td>
<td>60</td>
<td>60</td>
</tr>
<tr>
<td>Bit Overhead, r/k</td>
<td>0.56</td>
<td>0.875</td>
<td>0.875</td>
</tr>
<tr>
<td>Code Rate, k/n</td>
<td>0.64</td>
<td>0.53</td>
<td>0.53</td>
</tr>
<tr>
<td>Code Efficiency, r/n</td>
<td>0.36</td>
<td>0.47</td>
<td>0.47</td>
</tr>
<tr>
<td>Correction Capability</td>
<td>Only odd number up to 15 Adjacent Errors</td>
<td>Up to 15 Adjacent Errors</td>
<td>Up to 16 Adjacent Errors</td>
</tr>
</tbody>
</table>

From table 4(d), method-3 of decoding is capable of correcting a maximum of 32 adjacent errors i.e., either in upper half or lower half of data read from memory with a code rate of 58%.


Table 4(d) Comparison Table for Decoding Methods for 64-Bit Data

<table>
<thead>
<tr>
<th>Parameter/Methods</th>
<th>Method-1</th>
<th>Method-2</th>
<th>Method-3</th>
</tr>
</thead>
<tbody>
<tr>
<td># Data Bits, k</td>
<td>64</td>
<td>64</td>
<td>64</td>
</tr>
<tr>
<td># Parity Bits, r</td>
<td>34</td>
<td>46</td>
<td>46</td>
</tr>
<tr>
<td># Code Word, n=k+r</td>
<td>98</td>
<td>110</td>
<td>110</td>
</tr>
<tr>
<td>Bit Overhead, r/k</td>
<td>0.53</td>
<td>0.72</td>
<td>0.72</td>
</tr>
<tr>
<td>Code Rate, k/n</td>
<td>0.65</td>
<td>0.58</td>
<td>0.58</td>
</tr>
<tr>
<td>Code Efficiency, r/n</td>
<td>0.35</td>
<td>0.42</td>
<td>0.42</td>
</tr>
<tr>
<td>Correction Capability</td>
<td>Only odd number up to 31 Adjacent Errors</td>
<td>Up to 31 Adjacent Errors</td>
<td>Up to 32 Adjacent Errors</td>
</tr>
</tbody>
</table>

The bit overhead is less for decoding using method-1 and the other two methods share the same overhead by 50%, 44.44%, 36% and 26.38% for 8, 16, 32 and 64 bit data respectively. Even though the code rate is optimal, method-3 proves to be a better choice as the number of bits that can be corrected is N/2 in an N-bit Data. The code rate improves as the number of data bits are increased and has improved from 40% to 58% i.e., an improvement of 31%.

The simulation Results are shown in figures 4, 5 and 6 for the three methods of decoding respectively. The results show that the method-1 corrects only odd number of N/2 adjacent errors. The method-2 corrects up to N/2 - 1 adjacent errors and the method-3 corrects up to N/2 Adjacent Errors in an N-Bit Data.

From figure 4, if the data is “00000000”, then only 1 or 3 bits are corrected, i.e., for “00000001” and “00000111”, the correct output is obtained by using method-1 of decoding.

From figure 5, if the data is “00000000”, then only 1, 2 or 3 bits are corrected, i.e., for “00000001”, “00000011” and “00000111”, the correct output is obtained by using method-2 of decoding.

From figure 6, if the data is “00000000”, then only 1, 2, 3 or 4 bits are corrected, i.e., for “00000001”, “00000011”, “00000111” and “00001111”, the correct output is obtained by using method-3 of decoding.

Further the complexity of developed Encoders and Decoders are evaluated in terms of area and power-delay product as shown in figures 7 and 8 respectively.

In figure 7, the red colour bars represent the encoders without hamming bits and with hamming bits. Also the blue colour bars represent the decoders using method-1 corresponding to encoder-1, method-2 and method-3 corresponding to encoder-2 respectively. It shows that both in encoder and decoder, the area is increased to increase the reliability in correcting the adjacent errors.

The decoder-3 shows optimum results in terms of area for various data sizes by at least 16.36% reduction in area occupied when compared to decoder-2 and increase in area occupied by a minimum of 9.6% to a maximum of 56%. So area remains a compromise for higher order reliability.

In figure 8, the orange color bars represent the encoders without hamming bits and with hamming bits. Also the green color bars represent the decoders using method-1 corresponding to encoder-1, method-2 and method-3 corresponding to encoder-2 respectively. It shows that both in
encoder and decoder, the power delay product has increased slightly to increase the reliability in correcting the adjacent errors. The decoder-3 shows optimum results in terms of power delay product for various data sizes by at least 32.77% to 39.88% reduction when compared to decoder-2 and increase by a minimum of 5% to a maximum of 12.72%. So power delay product also increases slightly to achieve higher order reliability.

4.0 CONCLUSION

As the technology scales down, the soft errors are caused in memories due to radiation effects. This paper concentrates on correcting adjacent errors in memories using indirect decoding mechanisms. The hamming, extended hamming and vertical parity bits are used in correcting errors. Three decoding mechanisms are proposed. The designs are modelled in Verilog using 3D parity check code with hamming code. The assessment of these methods is done for 8, 16, 32 and 64 – Bit Data. The method-3 based decoder proves to be more reliable which is capable of correcting N/2 adjacent errors either in lower half or upper half of N-Bit Data enabling it to be used in image processing applications. But method-3 compromises with decrease in code rate, increase in bit overhead, area and power delay product by at least 26.38%, 10.76%, 9.6% and 5%.

Acknowledgement

This research is supported by JNTUA Ananthapuramu.

References


