SSRG International Journal of Electrical and Electronics Engineering
ISSN: 2348-8379/ https://doi.org/10.14445/23488379/IJEEE-V12I2P116

Volume 12 Issue 2, 146-156, February 2025

Review Article

A Survey on Matrix Based Error Detection and Correction Codes

Kavya Cherakula[1], Varadarajan Sourirajan[2]

[1,2]*Department of ECE, Sri Venkateswara University, Tirupati, Andhra Pradesh, India.*

[1]*Corresponding Author : kavyach.phd@gmail.com*

Received: 13 December 2024 Revised: 12 January 2025 Accepted: 10 February 2025 Published: 22 February 2025

Abstract - *Semiconductor memories are prone to various types of faults, such as stuck-at faults, memory faults, etc, that manifest as errors. As the data is usually stored in the memory in matrix form, the error correction capability is maximised by using Matrix codes with a minimal number of parity bits and improvement in code rate. The survey of codes extracted include MPC, 3D, HVD, HVDD, DMC, MDMC, PMC, HDMC, OPC, PrMC and MPrMC codes. The results are obtained by modeling in Verilog HDL using Xilinx Vivado Tool 28nm Zynq FPGA (XC7Z100-2FFG1156). These methods are evaluated for redundant bits, code rate, area in terms of LUTs, power dissipation, delay, etc. The MPrMC method – 2 Code uses reduced bit overhead by atleast 25.77% to 70.59%, increases code rate by 8.38% to 57.16%, decreases area occupied by 45.98% to 52.04% for encoder, 7.43% to 13.37% for decoder, decreases PDP by 19.69% to 51.74% for encoder and 33.67% to 40.97% for decoder. Hence, the MPrMC code proves to be a better choice in all aspects but trades off the area utilised.*

Keywords - *Code rate, Errors, Faults, Matrix codes, Radiation, Redundant bits.*

1. Introduction

As the technology scales down, integrating many devices within a single integrated circuit yields higher densities and miniaturisation. Due to this, radiation occurs due to ionization in the semiconductor memories, unlike the other essential components in the ICs. The radiation is a manifestation of high temperatures in the surrounding elements of semiconductor devices say in integrated circuits, the MOSFETs and the routed wires used for signal/ power transmission between the devices and I/O Blocks.

The radiation caused may be characterized by alpha particles, gamma rays, neutrons, electrons, etc. Memories are the most affected devices with ionizing radiation effects, which cause faults and manifest as errors. The radiation effects are either transient or permanent based on recovery time from radiation and restoration of its functionality. The transient radiation effects occur when a heavily charged particle passes through the element and creates Single Event Upsets (SEU) or Single Event Effects (SEEs). They cause false signals or logic states for a very short time that won't damage the device. But these induce soft errors. On the other hand, the permanent radiation effect or hard error caused by gamma rays results in altering the structure or functionality of the semiconductor device. These cause hard errors like latch-up, snapback, etc and can be resolved by turning off the device. The radiation effects on memories might alter the bit stored, resulting in erroneous data. These radiation effects affect semiconductor memories and form faults like destructive read faults, coupling faults, stuck-at faults, stuck-open faults, and address decoder faults. These faults manifest as errors, which may be single or multiple-bit errors. Multiple-bit errors [1] can be further classified as adjacent or burst errors and random errors. If the data read from memory has only a one-bit change, it is called a single-bit error. If the data read from memory has multiple changes in data bits that occur randomly at different data locations, then they are called random errors. If the data read from memory has multiple changes in data bits that occur in adjacent bits of data, they are called burst errors. There are many significant applications, such as medical and defense applications, where the reliability of data stored is highly necessary. For example, say the data stored in memory is related to launching a satellite.

Even a bit of change might result in the satellite being placed in a misplaced orbit, resulting in a waste of cost incurred in the design and development of the satellite [2]. Also, in medical applications, as in the present scenario, patient-related details are stored in databases for a long period of time. If errors occur while reading the data, then the patient might suffer from a wrong diagnosis, resulting in further damage to health and wealth. In robotic surgeries, even a bit of change might result in loss of life for the patient undergoing surgery. Hence, the concept of EDAC codes ensures reliability. The problem statement includes the necessity of low overhead codewords and a high code rate to detect and

correct errors caused by faults in semiconductor memories to facilitate the use of available bandwidth effectively to reduce data errors. Much of the research is found to have only a 65% code rate and atleast 70% bit overhead. This paper summarizes the EDAC codes developed with a focus towards ensuring the reliability of memories. The paper is ordered as a survey of literature in section II, the EDAC codes evaluated in this paper in section III, parameters used to assess EDAC codes in section IV and simulation results with discussion in section V. Finally, the paper is concluded.

2. Literature Survey

In [1], a new hybrid architecture for optimized performance of reconfiguration techniques, such as hardwired seed bits with interleaving capability based on compressed redundant information, is used to correct Multiple Cell Upsets (MCUs). Flexible unequal error control methodology is devised with column line code represented as matrix code with supported extended performing codes and parity checks [2]. In [3], using an auxiliary codeword, a two-level code based on low-density parity-check codes is developed for Non-Volatile Memories (NVM). In [4], a new optimised sub-expression elimination method is proposed to reduce area and power consumption without affecting speed and can detect double errors and correct a single error.

The linear block code with various data segments adaptive length is considered for higher reliability with a suitable information rate utilized for FPGA-based implementation [5]. Fault-tolerant encoders use logic-sharing blocks for every two adjacent parity bits for Single Error Correction (SEC) and Double Adjacent Error Correction (DAEC) codes [6]. The reordering of the hamming matrix along with a selective shortening scheme is proposed for SEC, Double Error Detection (DED), and Triple Adjacent Error Detection (TAED) codes to detect MCUs in SRAM memory designs [7].

A new methodology with minimum error probability along with bit interleaving is used for greater flexibility to optimize memory and to enhance protection from MCUs [8] is proposed. An optimized decoding method is proposed for SEC-DED-DAEC codes with constraints on H - Matrix suitable for different word lengths [9]. It offers a significant reduction in circuit area, delay and power. The scrubbing sequences improve the reliability of memories by mitigating MCU errors with optimal interleaving distance, which has been proven to improve Mean Time to Failure (MTTF) [10].

In [11], the focus is to improve manufacturing yield by using Matrix code that combines parity codes to ensure the reliability and yield of the memory chips. The matrix code capable of correcting 11 errors in 32-bit data size and 9 erroneous bits in 16-bit data size is proposed with a modified decoding algorithm [12]. In [13], new low redundant matrix Error Correction Codes (ECCs) that can correct adjacent errors with low redundancy in area, delay and power consumption are observed. A channel coding technique [14] that improves the reliability and efficiency of data transmission is proposed based on a multidirectional parity check code capable of correcting 4 error bits is developed. An EDAC method using a 3D parity check code capable of correcting 3 erroneous bits in data and parity bits is proposed in [15]. It achieves higher reliability with a trade-off in area and power consumption. In [16], The Decimal Matrix Code (DMC) enhances memory reliability with low delay overhead by dividing symbols. Further, the area overhead is minimized by using the Encoder Reuse Technique. In [17], The Decimal Matrix Code (DMC) is evaluated for mean time to failure, delay overhead, etc, but still, bit overhead remains high.

The Modified Decimal Matrix Code (MDMC) uses reconfigurable array XOR logic to compute decimal addition equivalent [18]. The Parity Matrix Code (PMC) is provided as an improved version of MDMC codes and proves to be better reliable for memories [19]. In [20], the 2-D code is named Horizontal Vertical Diagonal (HVD) code, where row, column, slash, and backslash diagonal parity bits are used to increase the correction capability. In [21], a Horizontal, Vertical Double-bit Diagonal (HVDD) code detects and corrects multi-bit soft errors using the comparatively low overhead. The ECC code uses modified hamming code to protect data from memory against 3-bit errors and reduce 4-bit error detection probability [22]. In [23], a low latency zero overhead burst error correction technique based on Decision Feedback Equalization (DFE) is proposed that works with less power consumption. A technique based on design rules and a search algorithm extends 3-bit Burst Error Correction (BEC) code and Quadruple Adjacent Error Correction (QAEC) [24].

An area-efficient matrix code using hardware redundancy and encoder reuse technique is presented in [25]. In [28], ultrafast error control codes are proposed, which work independently of word length to increase reliability with very low delays that combine DED and Adjacent Error Correction (AEC). A Double Error Correction (DEC) systematic (16,8) quasi-cycle code is used to detect Triple Adjacent Errors (TAE) with Triple Error Correction (TEC) and Quadruple Error Detection (QED) capability [29]. Hence, the major observations include the number of errors corrected is nearly only 1/4th of the erroneous data, the area utilization is more than 50% of the device LUTs only for either encoder or decoder, the power delay product remains a trade-off with error correction capability, the bit overhead is large which doesn't improve code rate, as data bits increase, there is a significant decrease in fault coverage which doesn't ensure reliability, etc.

3. EDAC Codes

The very basic EDAC is Hamming code [1] proposed by R.W. Hamming for linear block ECC with SEC-DED. The (12, 8) Hamming code is shown in Figure 1.

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

Fig. 1 The shortened hamming code (12,8)

The different and nonzero columns in the H matrix must have odd-weight with data columns whose weight is >1, and the sum of two adjacent columns must be nonzero [4]. The Extended Hamming Codes use an additional parity bit to get an even weight syndrome and ensure TAEC capability. Hsiao codes [5] represent optimized Hamming codes [6] that use a minimum odd number of 1s in each column, as shown in Figure 2.

b0	b1	b2	b3	b4 u0	b5 u1	b6 u2	b7 u3	Encoding formulas
1	0	0	0	0	1	1	1	b0=u1+u2+u3
0	1	0	0	1	0	1	1	b1=u0+u2+u3
0	0	1	0	1	1	0	1	b2=u0+u1+u3
0	0	0	1	1	1	1	0	b3=u0+u1+u2

r0	r1	r2	r3	r4 u0	r5 u1	r6 u2	r7 u3	Syndrome bits
1	0	0	0	0	1	1	1	s0=r0+r5+r6+r7
0	1	0	0	1	0	1	1	s1=r1+r4+r6+r7
0	0	1	0	1	1	0	1	s2=r2+r4+r5+r7
0	0	0	1	1	1	1	0	s3=r3+r4+r5+r6

Fig. 2 (8,4) Hsiao code

The Hsiao codes detect one random error but can correct up to 1/8th of adjacent errors. The other codes include SEC-DAED [7], based on extended hamming code [8]. The multidimensional parity-check codes use horizontal, vertical and diagonal parity bits for EDAC, which increase bit overhead [14]. There exist several codes that aim at TAED [19, 20] but still have a bit of overhead and a decreased code rate [21]. The HVD, i.e., Horizontal Vertical and Diagonal codes, also called 3-D HVD Codes [15], increase parity bits to ensure adjacent error corrections up to 1/4th of data bits [17].

As shown in Figure 3, the parity bits are calculated using modulo - 2 operation for encoding that corrects up to 1/4th of the adjacent errors. The 4-D Codes [22] use horizontal, vertical, forward slash diagonal and backward slash diagonal parity bits, which have the same error correction capability as that of 3-D HVD Code, as shown in Figure 4. The ultrafast codes [2, 3] modify Hsiao Codes with the hamming distance of 1 in each column and can correct up to 1/4th of erroneous data and 2 parity bit errors, as shown in Figure 5.

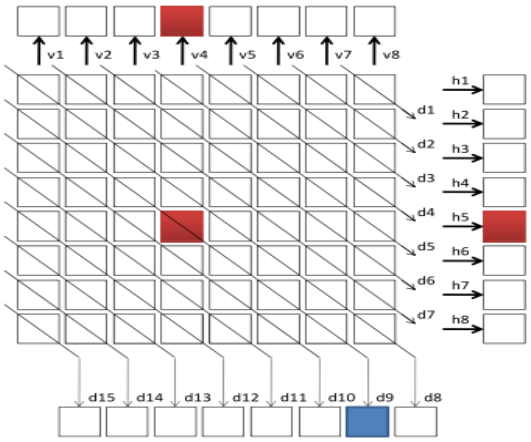

Fig. 3 HVD Code

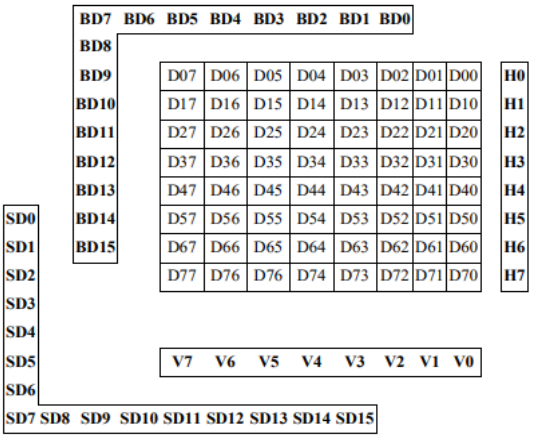

Fig. 4 4-D codes

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

Fig. 5 (16, 8) Ultrafast code

The Quaternary Adjacent Error Correction (QAEC) Codes [24] optimize decoder complexity and delay [9, 25]. The recursive backtracing algorithm is used to reduce run time costs and improve performance. The Decimal Matrix Code (DMC) [16, 17] divides symbols and uses the decimal integer addition and subtraction [25] along with the Encoder Reuse Technique (ERT), as shown in Figure 6. To maximise the correction capability with low overhead, the optimal choice of k and m must be ensured, as shown in Figure 7. The modified DMC (MDMC) [18] modifies DMC using higher-order adders and subtractors for H-bits. For N data bits, k-symbols of m-bits are subdivided as N = k x m [25], as shown in Figure 8.

Symbol 7				Symbol 2				Symbol 5				Symbol 0														
D_{15}	D_{14}	D_{13}	D_{12}	D_{11}	D_{10}	D_9	D_8	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0	H_9	H_8	H_7	H_6	H_5	H_4	H_3	H_2	H_1	H_0
D_{31}	D_{30}	D_{29}	D_{28}	D_{27}	D_{26}	D_{25}	D_{24}	D_{23}	D_{22}	D_{21}	D_{20}	D_{19}	D_{18}	D_{17}	D_{16}	H_{19}	H_{18}	H_{17}	H_{16}	H_{15}	H_{14}	H_{13}	H_{12}	H_{11}	H_{10}
V_{15}	V_{14}	V_{13}	V_{12}	V_{11}	V_{10}	V_9	V_8	V_7	V_6	V_5	V_4	V_3	V_2	V_1	V_0										

Fig. 6 DMC (k = 2 x 4, m = 4)

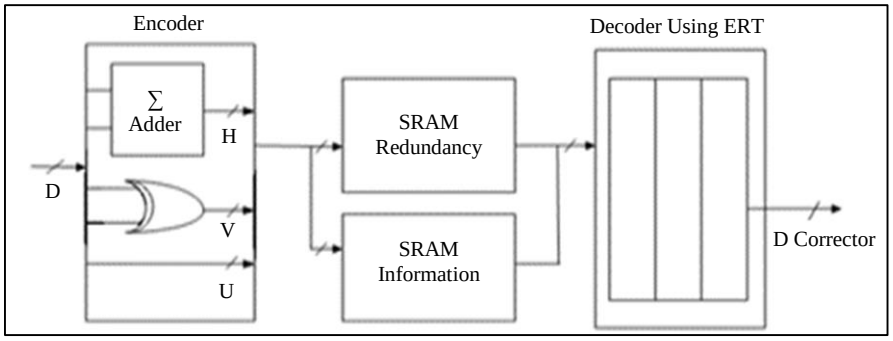

Fig. 7 DMC code

Symbol 7				Symbol 2				Symbol 5				Symbol 0														
D_{15}	D_{14}	D_{13}	D_{12}	D_{11}	D_{10}	D_9	D_8	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0	P_{11}	P_{10}	P_9	P_8	P_7	P_6	P_5	P_4	P_3	P_2	P_1	P_0
D_{31}	D_{30}	D_{29}	D_{28}	D_{27}	D_{26}	D_{25}	D_{24}	D_{23}	D_{22}	D_{21}	D_{20}	D_{19}	D_{18}	D_{17}	D_{16}												
V_{15}	V_{14}	V_{13}	V_{12}	V_{11}	V_{10}	V_9	V_8	V_7	V_6	V_5	V_4	V_3	V_2	V_1	V_0												

Fig. 8 MDMC

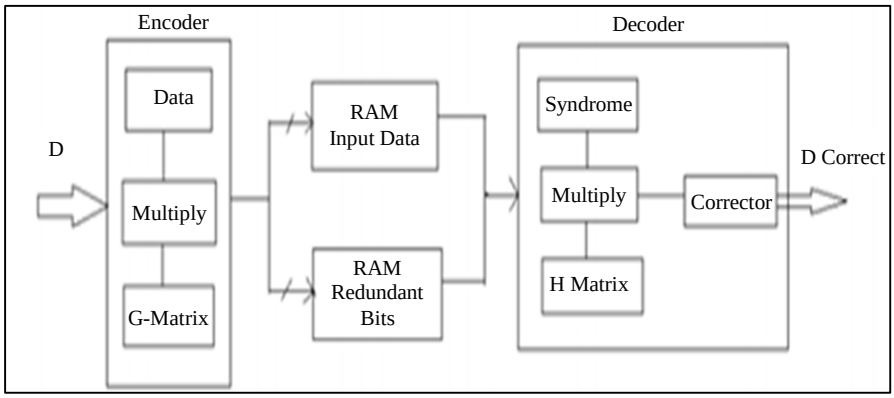

Fig. 9 PMC

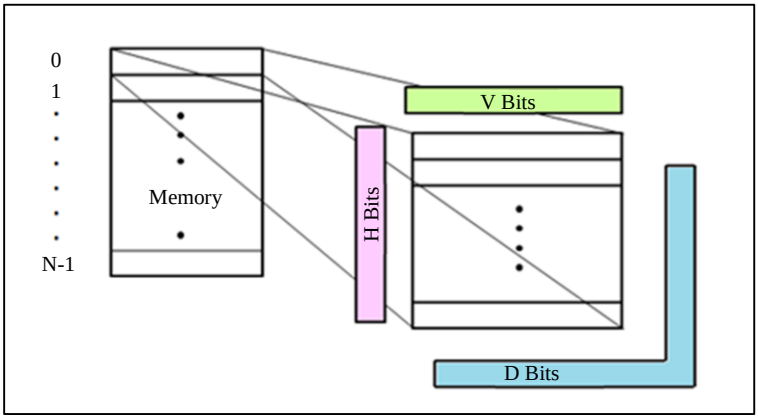

Fig. 10 3D Parity check

Parity Matrix Code (PMC) [19], as shown in Figure 9, ensures higher correction capability using an identity parity matrix than MDMC. The matrix-reordered codes aim to improve the number of bits corrected, and the 3D codes utilize the matrix representation, as shown in Figure 10. For 64-bit data, the matrix can be organized as 8 rows x 8 columns that use 8-H Bits, 8-V Bits and 15-D Bits, as shown in Figure 11. If the matrix is organized as 4 rows X 16 columns, then 4 H, 16 V and 19 D bits are used. If the matrix is organized as 2 rows X 32 columns, then 2 H, 32 V and 33 D bits are used.

	v_1	v_2	v_3	v_4	v_5	v_6	v_7	v_8	
h_1	m_0	m_1	m_2	m_3	m_4	m_5	m_6	m_7	
h_2	m_8	m_9	m_{10}	m_{11}	m_{12}	m_{13}	m_{14}	m_{15}	d_1
h_3	m_{16}	m_{17}	m_{18}	m_{19}	m_{20}	m_{21}	m_{22}	m_{23}	d_2
h_4	m_{24}	m_{25}	m_{26}	m_{27}	m_{28}	m_{29}	m_{30}	m_{31}	d_3
h_5	m_{32}	m_{33}	m_{34}	m_{35}	m_{36}	m_{37}	m_{38}	m_{39}	d_4
h_6	m_{40}	m_{41}	m_{42}	m_{43}	m_{44}	m_{45}	m_{46}	m_{47}	d_5
h_7	m_{48}	m_{49}	m_{50}	m_{51}	m_{52}	m_{53}	m_{54}	m_{55}	d_6
h_8	m_{56}	m_{57}	m_{58}	m_{59}	m_{60}	m_{61}	m_{62}	m_{63}	d_7
		d_{15}	d_{14}	d_{13}	d_{12}	d_{11}	d_{10}	d_9	d_8

(a) 8 x 8 Matrix

	v_1	v_2	v_3	v_4	v_5	v_6	v_7	v_8	v_9	v_{10}	v_{11}	v_{12}	v_{13}	v_{14}	v_{15}	v_{16}	
h_1	m_0	m_1	m_2	m_3	m_4	m_5	m_6	m_7	m_8	m_9	m_{10}	m_{11}	m_{12}	m_{13}	m_{14}	m_{15}	
h_2	m_{16}	m_{17}	m_{18}	m_{19}	m_{20}	m_{21}	m_{22}	m_{23}	m_{24}	m_{25}	m_{26}	m_{27}	m_{28}	m_{29}	m_{30}	m_{31}	d_1
h_3	m_{32}	m_{33}	m_{34}	m_{35}	m_{36}	m_{37}	m_{38}	m_{39}	m_{40}	m_{41}	m_{42}	m_{43}	m_{44}	m_{45}	m_{46}	m_{47}	d_2
h_4	m_{48}	m_{49}	m_{50}	m_{51}	m_{52}	m_{53}	m_{54}	m_{55}	m_{56}	m_{57}	m_{58}	m_{59}	m_{60}	m_{61}	m_{62}	m_{63}	d_3
	d_{19}	d_{18}	d_{17}	d_{16}	d_{15}	d_{14}	d_{13}	d_{12}	d_{11}	d_{10}	d_9	d_8	d_7	d_6	d_5	d_4	

(b) 4 x 16 Matrix

	v_1	v_2	v_3	v_4	v_5	v_6	v_7	v_8	v_9	v_{10}	v_{11}	v_{12}	v_{13}	v_{14}	v_{15}	v_{16}	v_{17}	v_{18}	v_{19}	v_{20}	v_{21}	v_{22}	v_{23}	v_{24}	v_{25}	v_{26}	v_{27}	v_{28}	v_{29}	v_{30}	v_{31}	v_{32}	
h_1	m_0	m_1	m_2	m_3	m_4	m_5	m_6	m_7	m_8	m_9	m_{10}	m_{11}	m_{12}	m_{13}	m_{14}	m_{15}	m_{16}	m_{17}	m_{18}	m_{19}	m_{20}	m_{21}	m_{22}	m_{23}	m_{24}	m_{25}	m_{26}	m_{27}	m_{28}	m_{29}	m_{30}	m_{31}	
h_2	m_{32}	m_{33}	m_{34}	m_{35}	m_{36}	m_{37}	m_{38}	m_{39}	m_{40}	m_{41}	m_{42}	m_{43}	m_{44}	m_{45}	m_{46}	m_{47}	m_{48}	m_{49}	m_{50}	m_{51}	m_{52}	m_{53}	m_{54}	m_{55}	m_{56}	m_{57}	m_{58}	m_{59}	m_{60}	m_{61}	m_{62}	m_{63}	d_1
	d_{33}	d_{32}	d_{31}	d_{30}	d_{29}	d_{28}	d_{27}	d_{26}	d_{25}	d_{24}	d_{23}	d_{22}	d_{21}	d_{20}	d_{19}	d_{18}	d_{17}	d_{16}	d_{15}	d_{14}	d_{13}	d_{12}	d_{11}	d_{10}	d_9	d_8	d_7	d_6	d_5	d_4	d_3	d_2	

(c) 2 x 32 Matrix

Fig. 11 3D Parity check code

D_{63} D_{62}	D_{61}	D_{60} D_{59}	D_{58} D_{57}	D_{56}	D_{55} D_{54}	D_{53} D_{52}	D_{51}	D_{50} D_{49}	D_{48}	D_{47} D_{46}	D_{45}	D_{44}	D_{43}	D_{42} D_{41}	D_{40}	D_{39} D_{38}	D_{37} D_{36}	D_{35}	D_{34} D_{33}	D_{32}	H_2
D_{31} D_{30}	D_{29}	D_{28} D_{27}	D_{26} D_{25}	D_{24}	D_{23} D_{22}	D_{21} D_{20}	D_{19}	D_{18} D_{17}	D_{16}	D_{15} D_{14}	D_{13}	D_{12}	D_{11}	D_{10} D_9	D_8	D_7 D_6	D_5 D_4	D_3	D_2 D_1	D_0	H_1
V_{31} V_{30}	V_{29}	V_{28} V_{27}	V_{26} V_{25}	V_{24}	V_{23} V_{22}	V_{21} V_{20}	V_{19}	V_{18} V_{17}	V_{16}	V_{15} V_{14}	V_{13}	V_{12}	V_{11}	V_{10} V_9	V_8	V_7 V_6	V_5 V_4	V_3	V_2 V_1	V_0	H_0

Fig. 12 HDMC encoding mechanism for 64-bit data size

The Half Diagonal Matrix Codes (HDMC), as shown in Figure 12, focus on eliminating the usage of diagonal parity bits and using horizontal bits themselves as bits to improve the code rate.

In the decoder, $\triangle H$ is calculated from modulo-2 addition, and the error is detected only if $\triangle H$ and S are nonzero numbers [28]. The Optimal Parity Code -1 (OPC – 1) is similar to DMC, but vector adjustment is performed S={S, 32'b0} for MSB and S = {32'b0, S} for LSB data correction to correct n/2 adjacent errors.

The process of EDAC by using total parity bits = 36 + 32 = 68 for 64 data bits and 8-bit adders is as shown below [29].

Step-1: Encoder

Inputs: En, Di

Outputs: H-Bits, V-Bits

If En = 1 then

$$H[8:0] = di[7:0] + di[23:16]$$
$$H[17:9] = di[15:8] + di[31:24]$$
$$H[26:18] = di[39:32] + di[55:48]$$
$$H[35:27] = di[47:41] + di[63:56]$$
$$V_i = di_i \oplus di_{i+32} \quad where\ i = 0,1,\dots,31$$

Else

H = 0 and V = 0

Step-2: Data Written to Memory with a given address

Step-3: Data Read from Memory from the same given address

Step-4: Decoder

Inputs: En, H-Bits, V-Bits, Dr

Output: Do

If En = 1 then

$$HD[8:0] = dr[7:0] + dr[23:16]$$
$$HD[17:9] = dr[15:8] + dr[31:24]$$
$$HD[26:18] = dr[39:32] + dr[55:48]$$
$$HD[35:27] = dr[47:41] + dr[63:56]$$
$$VD_i = dr_i \oplus dr_{i+32} \quad where\ i = 0,1,\dots,31$$

Syndrome Calculation

$$Hdiff = HD \oplus H$$
$$S = VD \oplus V$$

Error Location and Correction

If Hdiff = 0 and S = 0 then

Do = Dr

Else

Do = Dr ⊕ S

Else

Do = 0

The Optimal Parity Code-2 (OPC-2) uses 16-bit adders instead of 8-bit adders for H parity bits calculation, as shown below.

It uses total parity bits = 34 + 32 = 66 for 64 data bits. The changes in the encoder include

If En = 1 then

$$H[16:0] = di[15:0] + di[31:16]$$
$$H[33:17] = di[47:32] + di[63:48]$$
$$V_i = di_i \oplus di_{i+32} \quad where\ i = 0,1,\dots,31$$

Else

H = 0 and V = 0

and the decoder includes

If En = 1 then

$$HD[16:0] = dr[15:0] + dr[31:16]$$
$$HD[33:17] = dr[47:32] + dr[63:48]$$
$$VD_i = dr_i \oplus dr_{i+32} \quad where\ i = 0,1,\dots,31$$

The Optimal Parity Code-3 (OPC-3) uses hamming bits as H bits by considering each row of the matrix as 32-bit data, which yields 6 H bits for each row, i.e., 12 H bits for 64-bit data size, as shown below. It uses total parity bits = 12 + 32 = 44 for 64 data bits. The changes in the encoder include

If En = 1 then

$$H[0] = di[0] \oplus di[1] \oplus di[3] \oplus di[4] \oplus di[6] \oplus di[8] \oplus di[10] \oplus di[11] \oplus di[13] \oplus di[15] \oplus di[17] \oplus di[19] \oplus di[21] \oplus di[23] \oplus di[25] \oplus di[26] \oplus di[28] \oplus di[30]$$
$$H[1] = di[0] \oplus di[2] \oplus di[3] \oplus di[5] \oplus di[6] \oplus di[9] \oplus di[10] \oplus di[12] \oplus di[13] \oplus di[16] \oplus di[17] \oplus di[20] \oplus di[21] \oplus di[24] \oplus di[25] \oplus di[27] \oplus di[28] \oplus di[31]$$
$$H[2] = di[1] \oplus di[2] \oplus di[3] \oplus di[7] \oplus di[8] \oplus di[9] \oplus di[10] \oplus di[14] \oplus di[15] \oplus di[16] \oplus di[17] \oplus di[22] \oplus di[23] \oplus di[24] \oplus di[25] \oplus di[29] \oplus di[30] \oplus di[31]$$
$$H[3] = di[4] \oplus di[5] \oplus di[6] \oplus di[7] \oplus di[8] \oplus di[9] \oplus di[10] \oplus di[18] \oplus di[19] \oplus di[20] \oplus di[21] \oplus di[22] \oplus di[23] \oplus di[24] \oplus di[25]$$
$$H[4] = di[11] \oplus di[12] \oplus di[13] \oplus di[14] \oplus di[15] \oplus di[16] \oplus di[17] \oplus di[18] \oplus di[19] \oplus di[20] \oplus di[21] \oplus di[22] \oplus di[23] \oplus di[24] \oplus di[25]$$
$$H[5] = di[26] \oplus di[27] \oplus di[28] \oplus di[29] \oplus di[30] \oplus di[31]$$
$$H[6] = di[32] \oplus di[33] \oplus di[35] \oplus di[36] \oplus di[38] \oplus di[40] \oplus di[42] \oplus di[43] \oplus di[45] \oplus di[47] \oplus di[49] \oplus di[51] \oplus di[53] \oplus di[55] \oplus di[57] \oplus di[58] \oplus di[60] \oplus di[62]$$
$$H[7] = di[32] \oplus di[34] \oplus di[35] \oplus di[37] \oplus di[38] \oplus di[41] \oplus di[42] \oplus di[44] \oplus di[45] \oplus di[48] \oplus di[49] \oplus di[52] \oplus di[53] \oplus di[56] \oplus di[57] \oplus di[59] \oplus di[60] \oplus di[63]$$
$$H[8] = di[33] \oplus di[34] \oplus di[35] \oplus di[39] \oplus di[40] \oplus di[41] \oplus di[42] \oplus di[46] \oplus di[47] \oplus di[48] \oplus di[49] \oplus di[54] \oplus di[55] \oplus di[56] \oplus di[57] \oplus di[61] \oplus di[62] \oplus di[63]$$
$$H[9] = di[36] \oplus di[37] \oplus di[38] \oplus di[39] \oplus di[40] \oplus di[41] \oplus di[42] \oplus di[50] \oplus di[51] \oplus di[52] \oplus di[53] \oplus di[54] \oplus di[55] \oplus di[56] \oplus di[57]$$
$$H[10] = di[43] \oplus di[44] \oplus di[45] \oplus di[46] \oplus di[47] \oplus di[48] \oplus di[49] \oplus di[50] \oplus di[51] \oplus di[52] \oplus di[53] \oplus di[54] \oplus di[55] \oplus di[56] \oplus di[57]$$
$$H[11] = di[58] \oplus di[59] \oplus di[60] \oplus di[61] \oplus di[62] \oplus di[63]$$
$$V_i = di_i \oplus di_{i+32} \quad where\ i = 0,1,\dots,31$$

Else

H = 0 and V = 0

and the decoder includes

If En = 1 then

$$HD[0] = dr[0] \oplus dr[1] \oplus dr[3] \oplus dr[4] \oplus dr[6] \oplus dr[8] \oplus dr[10] \oplus dr[11] \oplus dr[13] \oplus dr[15] \oplus dr[17] \oplus dr[19] \oplus dr[21] \oplus dr[23] \oplus dr[25] \oplus dr[26] \oplus dr[28] \oplus dr[30]$$
$$HD[1] = dr[0] \oplus dr[2] \oplus dr[3] \oplus dr[5] \oplus dr[6] \oplus dr[9] \oplus dr[10] \oplus dr[12] \oplus dr[13] \oplus dr[16] \oplus dr[17] \oplus dr[20] \oplus dr[21] \oplus dr[24] \oplus dr[25] \oplus dr[27] \oplus dr[28] \oplus dr[31]$$
$$HD[2] = dr[1] \oplus dr[2] \oplus dr[3] \oplus dr[7] \oplus dr[8] \oplus dr[9] \oplus dr[10] \oplus dr[14] \oplus dr[15] \oplus dr[16] \oplus dr[17] \oplus dr[22] \oplus dr[23] \oplus dr[24] \oplus dr[25] \oplus dr[29] \oplus dr[30] \oplus dr[31]$$
$$HD[3] = dr[4] \oplus dr[5] \oplus dr[6] \oplus dr[7] \oplus dr[8] \oplus dr[9] \oplus dr[10] \oplus dr[18] \oplus dr[19] \oplus dr[20] \oplus dr[21] \oplus dr[22] \oplus dr[23] \oplus dr[24] \oplus dr[25]$$
$$HD[4] = dr[11] \oplus dr[12] \oplus dr[13] \oplus dr[14] \oplus dr[15] \oplus dr[16] \oplus dr[17] \oplus dr[18] \oplus dr[19] \oplus dr[20] \oplus dr[21] \oplus dr[22] \oplus dr[23] \oplus dr[24] \oplus dr[25]$$
$$HD[5] = dr[26] \oplus dr[27] \oplus dr[28] \oplus dr[29] \oplus dr[30] \oplus dr[31]$$
$$HD[6] = dr[32] \oplus dr[33] \oplus dr[35] \oplus dr[36] \oplus dr[38] \oplus dr[40] \oplus dr[42] \oplus dr[43] \oplus dr[45] \oplus dr[47] \oplus dr[49] \oplus dr[51] \oplus dr[53] \oplus dr[55] \oplus dr[57] \oplus dr[58] \oplus dr[60] \oplus dr[62]$$
$$HD[7] = dr[32] \oplus dr[34] \oplus dr[35] \oplus dr[37] \oplus dr[38] \oplus dr[41] \oplus dr[42] \oplus dr[44] \oplus dr[45] \oplus dr[48] \oplus dr[49] \oplus dr[52] \oplus dr[53] \oplus dr[56] \oplus dr[57] \oplus dr[59] \oplus dr[60] \oplus dr[63]$$
$$HD[8] = dr[33] \oplus dr[34] \oplus dr[35] \oplus dr[39] \oplus dr[40] \oplus dr[41] \oplus dr[42] \oplus dr[46] \oplus dr[47] \oplus dr[48] \oplus dr[49] \oplus dr[54] \oplus dr[55] \oplus dr[56] \oplus dr[57] \oplus dr[61] \oplus dr[62] \oplus dr[63]$$
$$HD[9] = dr[36] \oplus dr[37] \oplus dr[38] \oplus dr[39] \oplus dr[40] \oplus dr[41] \oplus dr[42] \oplus dr[50] \oplus dr[51] \oplus dr[52] \oplus dr[53] \oplus dr[54] \oplus dr[55] \oplus dr[56] \oplus dr[57]$$
$$HD[10] = dr[43] \oplus dr[44] \oplus dr[45] \oplus dr[46] \oplus dr[47] \oplus dr[48] \oplus dr[49] \oplus dr[50] \oplus dr[51] \oplus dr[52] \oplus dr[53] \oplus dr[54] \oplus dr[55] \oplus dr[56] \oplus dr[57]$$
$$HD[11] = dr[58] \oplus dr[59] \oplus dr[60] \oplus dr[61] \oplus dr[62] \oplus dr[63]$$
$$VD_i = dr_i \oplus dr_{i+32} \quad where\ i = 0,1,\dots,31$$

The Optimal Parity Code-4 (OPC-4) uses a modulo – 2 addition operation for H bits, as shown below. It uses total parity bits = 2 + 32 = 34 for 64 data bits. The changes in the encoder include

If Enable = 1 then

$$H[0] = di[0] \oplus di[1] \oplus di[2] \oplus di[3] \oplus di[4] \oplus di[5] \oplus di[6] \oplus di[7] \oplus di[8] \oplus di[9] \oplus di[10] \oplus di[11] \oplus di[12] \oplus di[13] \oplus di[14] \oplus di[15] \oplus di[16] \oplus di[17] \oplus di[18] \oplus di[19] \oplus di[20] \oplus di[21] \oplus di[22] \oplus di[23] \oplus di[24] \oplus di[25] \oplus di[26] \oplus di[27] \oplus di[28] \oplus di[29] \oplus di[30] \oplus di[31]$$

$$H[1] = di[32] \oplus di[33] \oplus di[34] \oplus di[35] \oplus di[36] \oplus di[37] \oplus di[38] \oplus di[39] \oplus di[40] \oplus di[41] \oplus di[42] \oplus di[43] \oplus di[44] \oplus di[45] \oplus di[46] \oplus di[47] \oplus di[48] \oplus di[49] \oplus di[50] \oplus di[51] \oplus di[52] \oplus di[53] \oplus di[54] \oplus di[55] \oplus di[56] \oplus di[57] \oplus di[58] \oplus di[59] \oplus di[60] \oplus di[61] \oplus di[62] \oplus di[63]$$

$$V_i = di_i \oplus di_{i+32} \quad where\ i = 0,1, \dots ,31$$

Else

H = 0 and V = 0

, and the decoder includes

If Enable = 1 then

$$HD[0] = dr[0] \oplus dr[1] \oplus dr[2] \oplus dr[3] \oplus dr[4] \oplus dr[5] \oplus dr[6] \oplus dr[7] \oplus dr[8] \oplus dr[9] \oplus dr[10] \oplus dr[11] \oplus dr[12] \oplus dr[13] \oplus dr[14] \oplus dr[15] \oplus dr[16] \oplus dr[17] \oplus dr[18] \oplus dr[19] \oplus dr[20] \oplus dr[21] \oplus dr[22] \oplus dr[23] \oplus dr[24] \oplus dr[25] \oplus dr[26] \oplus dr[27] \oplus dr[28] \oplus dr[29] \oplus dr[30] \oplus dr[31]$$

$$HD[1] = dr[32] \oplus dr[33] \oplus dr[34] \oplus dr[35] \oplus dr[36] \oplus dr[37] \oplus dr[38] \oplus dr[39] \oplus dr[40] \oplus dr[41] \oplus dr[42] \oplus dr[43] \oplus dr[44] \oplus dr[45] \oplus dr[46] \oplus dr[47] \oplus dr[48] \oplus dr[49] \oplus dr[50] \oplus dr[51] \oplus dr[52] \oplus dr[53] \oplus dr[54] \oplus dr[55] \oplus dr[56] \oplus dr[57] \oplus dr[58] \oplus dr[59] \oplus dr[60] \oplus dr[61] \oplus dr[62] \oplus dr[63]$$

$$VD_i = dr_i \oplus dr_{i+32} \quad where\ i = 0,1, \dots ,31$$

The proficient matrix codes utilise one way of encoding and three different ways of decoding. The bit overhead and code rate show a significant amount of improvement as the number of data bits increase that are considered for processing. The encoding mechanism is as depicted in Figure 13 [30] for 8-bit data where V[3...0], R[5...0] and H[1...0] represent vertical, hamming and extended hamming parity bits.

d[7]	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]				

Fig. 13 PrMC

Three decoding mechanisms, represented as method-1, method-2, and method-3, use the values of H and V, which are taken from the code word, but H' and V' are calculated from data read from memory. In PrMC decoding method-1, the extended parity bits and vertical bits for decoding are shown below.

If $\triangle$h[0] = 1
then
do[3:0] = dr[3:0] $\oplus$ s
else
if $\triangle$r[2:0] $\neq$ 0
then
do[0] = $\triangle r_{all}$[i] $\oplus$ $\triangle v_{all}$ [i]

In PrMC decoding method-2, the correction capability of up to 3 bits is enhanced by modifying the decoder and using hamming bits in addition to extended hamming bits and vertical parity bits, i.e., $do_i = \Delta r \oplus \triangle V$ for an even number of errors, as shown below.

If $\triangle$h[0] = 1 then
do[3:0] = dr[3:0] $\oplus$ s
else if $\triangle$r[2:0] $\neq$ 0 then
do[0] = $\triangle r_{corresponding}$[i] $\oplus$ $\triangle v_{all}$ [i]

In PrMC decoding method – 3, the decoder corrects 4 error bits by using hamming bits in addition to extended hamming bits and vertical parity bits to verify $\triangle$R and $do = dr \oplus \triangle V$ for an even number of errors, as shown below.

If $\triangle$h[0] = 1 then
do[3:0] = dr[3:0] $\oplus$ s
else if $\triangle$r[2:0] $\neq$ 0 then
do[0] = $\triangle r_{corresponding}$[i] $\oplus$ $\triangle v_{corresponding}$ [i]

Method 3 is more efficient as it is capable of correcting 4 adjacent errors in 8 data bits, but beyond that, the number of errors induced and observed remains the same. The Modified Proficient Matrix Codes (MPrMC), as shown in Figure 14, use vertical parity bits as modulo-2 addition of hamming bits encoded, which further yields a significant change in the way the data bits can be corrected from possible adjacent errors with a simplified decoding mechanism [31].

d[7]	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[2]	V[1]	V[0]

Fig. 14 MPrMC

The MPrMC decoding mechanism uses two methods represented by method-1 and method-2. In MPrMC method-1, the change used is $V'[i] = R[i] \oplus R[i+3]$ where the decoded hamming parity bits are calculated as

$R'[n] = dr[n] \oplus dr[n+1] \oplus dr[n+2]$ where n=0,1,2, …

and the decoded extended hamming parity bits are calculated using

$$H'[0] = dr[0] \oplus dr[1] \oplus dr[2] \oplus dr[3]$$

$$H'[1] = dr[4] \oplus dr[5] \oplus dr[6] \oplus dr[7]$$

The algorithm is represented as

If $\triangle$H or $\triangle$R $\neq$ 0

then

do[i] = $\triangle$Rall[i] $\oplus$ $\triangle$Vall[i]

The MPrMC method - 2 uses the following changes to evaluate higher-order data sizes.

If $\triangle H$ or $\triangle R \neq 0$

then

do[i] = $\triangle$Rcorresponding[i] ^ $\triangle$V corresponding [i]

These EDAC codes are used for Network on Chip [32] applications with data access through buffers as memories [33].

4. Evaluation Metrics

The EDAC codes are evaluated for parameters like bit overhead, code rate, correction capability, etc. The bit overhead is defined as the ratio of the number of parity bits to the number of data bits, which is usually the composition by which the bits are written into the memory. It must be as low as possible.

$$Bit\ Overhead = \frac{r}{k} \quad (1)$$

Where r is the number of parity bits used, and k is the number of data bits.

The Code Rate is defined as the ratio of a number of data bits to the number of codeword bits, again the combination of both data and parity bits. It must be as high as possible.

$$Code\ Rate = \frac{k}{n} \quad (2)$$

Where k is the number of data bits and n is the number of bits in the codeword, i.e., n = k + r.

The Correction Capability of any code is defined as the number of bits corrected from the detected number of errors. It must be the same as the number of erroneous bits detected.

The EDAC codes are further evaluated for technology-related parameters like area Slice Look-Up Tables (LUTs) occupied in the FPGA, combinational path delay, the power dissipated by the design, power delay product as the figure of merit, etc. Practically, these parameters must be as low as possible.

5. Results and Discussion

The EDAC codes are modeled in Verilog HDL and verified in Xilinx Vivado Tool for 28nm Zynq FPGA (XC7Z100-2FFG1156) for 8, 16, 32 and 64-bit data sizes processed at a time. Figure 15 shows the simulation result of MPrMC decoder methods, which are capable of correcting 4 erroneous bits in 8-bit data size. The comparison is shown in Table 1. For any communication system, the bit overhead must be as low as possible, but the code rate must be as high as possible. From Table 1, for 64-bit data, the bit overhead is less for PrMC decoding using method-1 and the other two methods have overhead increased by 50%. For n/2 correction capability, method - 3 evolves as a better choice with optimal code rate. Similarly, in MPrMC, the bit overhead is less, only 31.25%, with a code rate of 76.19%. Also, the MPrMC encoder and decoder method – 2 prove to be a better choice. The comparison of codes in terms of code rate and bit overhead is shown in Table 1. The results give insight as PrMC (method - 1, method – 2 or 3) and MPrMC codes (method – 1 or 2) show improvement in bit overhead by atleast 19.02% over other codes. The proposed MPrMC method – 2 proves to be a better choice as it requires 20 parity bits for 64 bits of data size, which yields 31.25% with a correction capability of 32-bit burst error. The proposed MPrMC method – 2 code proves to be a better choice, and it provides a code rate of 76.19%.

The area in terms of the number of slice LUTs occupied and delay should be kept at a minimum, as shown in Table 1 for encoders. The PrMC method - 1 code encoder proves to be a better choice, but still, for the area, HDMC and OPC – 4 codes remain a better choice. The PrMC method - 1 encoder proves to be a better choice by 28.88%, but still, for power delay products, HDMC code is a better choice. Among the proposed codes, the MPrMC (method – 1) code decoder proves to be a better choice by 13.37%, but still, for the area, HDMC, OPC – 3 and OPC - 4 codes are a better choice. The MPrMC method – 1 code is a better choice for less power delay products by 40.97% than other codes. Hence, proficient matrix codes have advantages like low bit overhead, high code rate, good correction capability, less power delay product, etc. However, the disadvantage is that it is unsuitable to use advanced techniques like parallel processing, pipelining, etc.

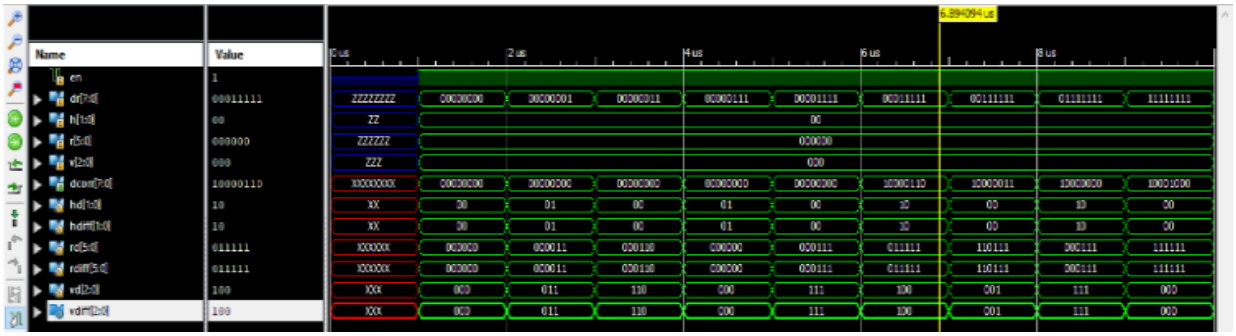

Fig. 15 Simulation result of MPrMC decoder methods

Table 1. Comparison of matrix codes in terms of bit overhead, code rate and correction capability

Codes/ Parameters	# Data Bits (k)	# Parity Bits (r)	# Codeword Bits (n=k+r)	Bit Overhead (r/k)	Code Rate (k/n)	Correction Capability (Bits)	Area in terms of Slice LUTs (out of 277400) for Encoder	Power Delay Product (pWs) for Encoder	Area in Terms of Slice LUTs (out of 277400) for Decoder	Power Delay Product (pWs) for Decoder
MPC	64	31	95	48.4%	67.3%	7	38	404.98	134	849.26
HVD	64	42	106	65.6%	60.3%	3	39	558.38	128	857.69
HVDD	64	27	91	42.1%	70.3%	3	39	403.44	121	844.13
3D	64	31	95	48.4%	67.3%	8	38	404.98	118	766.58
DMC	64	68	132	106.25%	48.48%	16	98	637.518	150	763.28
MDMC	64	66	130	103.125%	49.23%	16	97	671.40	175	796.63
PMC	64	44	99	68.75%	59.26%	16	87	590.68	187	793.81
4 x 16 MRC	64	39	103	60.9%	62.1%	16	43	533.20	119	897.44
2 x 32 MRC	64	67	131	104.6%	48.8%	32	45	599.17	175	984.65
HDMC	64	35	99	54.69%	64.65%	32	30	259.02	80	671.08
OPC-1	64	68	132	106.25%	48.48%	32	98	637.52	150	763.28
OPC-2	64	66	130	103.125%	49.23%	32	97	671.40	159	847.91
OPC-3	64	44	108	68.75%	59.25%	32	55	542.70	80	835.06
OPC-4	64	34	98	53.125%	65.31%	32	30	435.74	80	736.28
PrMC Method-1	64	34	98	53.12%	65.31%	31	32	288.04	112	588.01
PrMC Method-2	64	46	110	71.87%	58.18%	31	64	483.02	188	528.04
PrMC Method-3	64	46	110	71.87%	58.18%	32	64	329.03	155	539.02
MPrMC Method-1	64	20	84	31.25%	76.19%	32	47	324.00	109	506.32
MPrMC Method-2	64	20	84	31.25%	76.19%	32	47	324.00	162	506.32

6. Conclusion

This work aimed at maximizing the error correction capability using Matrix codes for critical applications. The methods considered focus on using a minimal number of redundant bits and improving the code rate. The HDMC code satisfies the area and PDP but trades off bit overhead and code rate. Also, OPC – 4 code satisfies code rate, area occupied, and PDP; still, bit overhead can be reduced. Even though the correction capability was retained, the MPrMC code used the least bit overhead of 31.25% with a code rate of 76.19%. Also, when compared with existing codes, the MPrMC method – 2 Code uses reduced bit overhead by atleast 25.77% to 70.59%, increases code rate by 8.38% to 57.16%, decreases area occupied by 45.98% to 52.04% for encoder, 7.43% to 13.37% for decoder, decreases PDP by 19.69% to 51.74% for encoder and 33.67% to 40.97% for decoder.

Hence, from the proposed codes, the MPrMC code proves to be a better choice in all aspects except in the area utilized. If area remains a concern, then HDMC code is a better choice but with a trade-off in bit overhead. If the area utilized and PDP are not a concern, then 8 x 8 PPMC remains a better choice. In the future, diagonal codes will be explored with a focus on quantum EDAC and machine learning approaches.

References

[1] Andrés Jiménez Olazábal, and Jorge Pleite Guerra, "Multiple Cell Upsets Inside Aircrafts. New Fault-Tolerant Architecture," *IEEE Transactions on Aerospace and Electronic Systems*, vol. 55, no. 1, pp. 332-342, 2019. [CrossRef] [Google Scholar] [Publisher Link]

[2] Joaquín Gracia-Morán et al., "Improving Error Correction Codes for Multiple-Cell Upsets in Space Applications," *IEEE Transactions on Very Large Scale Integration Systems*, vol. 26, no. 10, pp. 2132-2142, 2018. [CrossRef] [Google Scholar] [Publisher Link]

[3] "1890-2018 - IEEE Standard for Error Correction Coding of Flash Memory Using Low-Density Parity Check Codes," *IEEE*, pp. 1-51, 2019. [CrossRef] [Google Scholar] [Publisher Link]

[4] Jagannath Samanta, Jaydeb Bhaumik, and Soma Barman, "Compact and Power Efficient SEC-DED Codec for Computer Memory," *Microsystem Technologies*, vol. 27, pp. 359-368, 2021. [CrossRef] [Google Scholar] [Publisher Link]

[5] Abdullah Mohammed A. Hamdoon, Zaid Ghanim Mohammed, and Emad A. Mohammed, "Design and Implementation of Single Bit Error Correction Linear Block Code System Based on FPGA," *TELKOMNIKA Telecommunication, Computing, Electronics and Control*, vol. 17, no. 4, 2019. [CrossRef] [Google Scholar] [Publisher Link]

[6] Shanshan Liu et al., "Fault Tolerant Encoders for Single Error Correction and Double Adjacent Error Correction Codes," *Microelectronics Reliability*, vol. 81, pp. 167-173, 2018. [CrossRef] [Google Scholar] [Publisher Link]

[7] Alfonso Sánchez-Macián, Pedro Reviriego, and Juan Antonio Maestro, "Hamming SEC-DAED and Extended Hamming SEC-DED-TAED Codes through Selective Shortening and Bit Placement," *IEEE Transactions on Device and Materials* Reliability, vol. 14, no. 1, pp. 574-576, 2014. [CrossRef] [Google Scholar] [Publisher Link]

[8] Avijit Dutta, and Nur A. Touba, "Multiple Bit Upset Tolerant Memory Using a Selective Cycle Avoidance Based SEC-DED-DAEC Code," *25th IEEE VLSI Test Symposium*, Berkeley, CA, USA, pp. 349-354, 2007. [CrossRef] [Google Scholar] [Publisher Link]

[9] Pedro Reviriego et al., "A Method to Design SEC-DED-DAEC Codes with Optimized Decoding," *IEEE Transactions on Device Material Reliability*, vol. 14, no. 3, pp. 884-889, 2014. [CrossRef] [Google Scholar] [Publisher Link]

[10] Wei Zhou et al., "Designing Scrubbing Strategy for Memories Suffering MCUs through the Selection of Optimal Interleaving Distance," *International Journal of Computational Science and Engineering*, vol. 19, no. 1, pp. 53-63, 2019. [CrossRef] [Google Scholar] [Publisher Link]

[11] Costas Argyrides, Dhiraj K. Pradhan, and Taskin Kocak, "Matrix Codes for Reliable and Cost Efficient Memory Chips," *IEEE Transactions on Very Large Scale Integration Systems*, vol. 19, no. 3, pp. 420-428, 2011. [CrossRef] [Google Scholar] [Publisher Link]

[12] M.S. Sunita, and V.S. Kanchana Bhaaskaran, "Matrix Code Based Multiple Error Correction Technique for N-Bit Memory Data," *International Journal of VLSI Design & Communication Systems*, vol. 4, no. 1, pp. 29-37, 2013. [Google Scholar] [Publisher Link]

[13] Joaquín Gracia-Moran et al., "Correction of Adjacent Errors with Low Redundant Matrix Error Correction Codes," *Eighth Latin-American Symposium on Dependable Computing*, Foz do Iguacu, Brazil, pp.107-114, 2018. [CrossRef] [Google Scholar] [Publisher Link]

[14] Vishal Badole, and Amit Udawa, "Implementation of Multidirectional Parity Check Code Using Hamming Code for Error Detection and Correction," *International Journal of Research in Advent Technology*, vol. 2, no. 5, pp. 317-322, 2014. [Google Scholar] [Publisher Link]

[15] Shivani Tambatkar et al., "Error Detection and Correction in Semiconductor Memories Using 3D Parity Check Code with Hamming Code," *International Conference on Communication and Signal Processing*, Chennai, India, vol. 2, pp. 0974-0978, 2017. [CrossRef] [Google Scholar] [Publisher Link]

[16] T. Maheswari, and P. Sukumar, "Error Detection and Correction in SRAM Cell Using Decimal Matrix Code," *IOSR Journal of VLSI and Signal Processing*, vol. 5, no. 1, pp. 9-14, 2015. [Google Scholar] [Publisher Link]

[17] Jing Guo et al., "Enhanced Memory Reliability Against Multiple Cell Upsets Using Decimal Matrix Code," *IEEE Transactions on Very Large Scale Integration Systems*, vol. 22, no. 1, pp. 127-135, 2014. [CrossRef] [Google Scholar] [Publisher Link]

[18] A. Ahilan, and P. Deepa, "Modified Decimal Matrix Codes in FPGA Configuration Memory for Multiple Bit Upsets," *International Conference on Computer Communication and Informatics*, Coimbatore, India, pp. 1-5, 2015. [CrossRef] [Google Scholar] [Publisher Link]

[19] S. Manoj, and C. Babu, "Improved Error Detection and Correction for Memory Reliability Against Multiple Cell Upsets Using DMC and PMC," *IEEE Annual India Conference*, Bangalore, India, pp.1-6, 2016. [CrossRef] [Google Scholar] [Publisher Link]

[20] Shalini Sharma, and P. Vijayakumar, "An HVD Based Error Detection and Correction of Soft Errors in Semiconductor Memories Used for Space Applications," *International Conference on Devices, Circuits and Systems*, Coimbatore, India, pp. 563-567, 2012. [CrossRef] [Google Scholar] [Publisher Link]

[21] Md. Shamimur Rahman et al., "Soft Error Tolerance Using Horizontal-Vertical-Double-Bit Diagonal Parity Method," *International Conference on Electrical Engineering and Information Communication Technology*, Savar, Bangladesh, pp. 1-6, 2015. [CrossRef] [Google Scholar] [Publisher Link]

[22] Wael Toghuj, "Modifying Hamming Code and Using the Replication Method to Protect Memory Against Triple Soft Errors," *TELKOMNIKA Telecommunication, Computing, Electronics and Control*, vol. 18, no. 5, 2020. [CrossRef] [Google Scholar] [Publisher Link]

[23] Shovon Dey, Aurangozeb, and Masum Hossain, "Low-Latency Burst Error Detection and Correction in Decision-Feedback Equalization," *IEEE Open Journal of Circuits and Systems*, vol. 2, pp. 91-100, 2021. [CrossRef] [Google Scholar] [Publisher Link]

[24] Jiaqiang Li et al., "Extending 3-Bit Burst Error-Correction Codes with Quadruple Adjacent Error Correction," *IEEE Transactions on Very Large Scale Integration Systems*, vol. 26, no. 2, pp. 221-229, 2018. [CrossRef] [Google Scholar] [Publisher Link]

[25] J. Athira, and B. Yamuna. "FPGA Implementation of an Area Efficient Matrix Code with Encoder Reuse Method," *International Conference on Communication and Signal Processing*, Chennai, India, pp. 0254-0257, 2018. [CrossRef] [Google Scholar] [Publisher Link]

[26] Luis-J. Saiz-Adalid et al., "Ultrafast Codes for Multiple Adjacent Error Correction and Double Error Detection," *IEEE Access*, vol. 7, pp. 151131-151143, 2019. [CrossRef] [Google Scholar] [Publisher Link]

[27] T.A. Gulliver, and V.K. Bhargava, "A Systematic (16, 8) Code for Correcting Double Errors and Detecting Triple Adjacent Errors," *IEEE Transactions on Computers*, vol. 42, no. 1, pp. 109-112, 1993. [CrossRef] [Google Scholar] [Publisher Link]

[28] Neelima K, and C. Subhas, "Half Diagonal Matrix Codes for Reliable Embedded Memories," *International Journal of Health Sciences*, vol. 6, no. S2, pp. 11664-11677, 2022. [CrossRef] [Google Scholar] [Publisher Link]

[29] Neelima Koppala, and Chennapalli Subhas, "Low Overhead Optimal Parity Codes," *TELKOMNIKA Telecommunication Computing Electronics and Control*, vol. 20, no. 3, 2022. [CrossRef] [Google Scholar] [Publisher Link]

[30] Neelima K, and C. Subhas, "Proficient Adjacent Error Correcting Codes," *IEEE 3rd International Conference on Applied Electromagnetics, Signal Processing, & Communication*, Bhubaneswar, India, pp. 1-5, 2023. [CrossRef] [Google Scholar] [Publisher Link]

[31] Neelima K, C. Subhas, "Modified Proficient Adjacent Error Correcting Codes," *e-Prime - Advances in Electrical Engineering, Electronics and Energy*, vol. 5, 2023. [CrossRef] [Google Scholar] [Publisher Link]

[32] Neelima Koppala et al., "Proficient Matrix Codes for Error Detection and Correction in 8-Port Network on Chip Routers," *Indonesian Journal of Electrical Engineering and Computer Science*, vol. 29, no. 3, pp. 1336-1344, 2023. [CrossRef] [Google Scholar] [Publisher Link]

[33] Neelima K, C. Subhas, "Modified Matrix Codes for Shielding Memories Against Adjacent Errors," *ASEAN Engineering Journal*, vol. 14, no. 2, pp. 19-25, 2024. [CrossRef] [Google Scholar] [Publisher Link]