Error Detection & Correction using Horizontal-Vertical-Diagonal-Shift Method

Muhammad Sheikh Sadi, Muhammed Saifur Rahman, K.M. Imrul Kayes Sikdar
Department of Computer Science & Engineering
Khulna University of Engineering & Technology
Khulna, Bangladesh
e-mail: sheikhsadi@gmail.com, saifur.nishat@gmail.com, kayes.shamil@gmail.com

Abstract- **This paper proposes a novel approach to detect and correct multi bit upsets using horizontal-vertical-diagonal-shift (HVDS) parity. Many of the errors occur when information is transmitted from one node to another node. Detection and correction of these errors is a must for many systems e.g. safety critical systems. Existing work on error detection and correction cover up to three bit errors (as far reviewed) in a data word. The proposed method can detect and correct up to 7 bit errors. Experimental studies show the effectiveness of the proposed method in comparison to existing dominant work.**

Keywords: Error Detection and Correction, Horizontal Parity, Vertical Parity, Diagonal Parity, Shift Parity, Fault Tolerance.

I. INTRODUCTION

The high level of complexity and the fact that the software and hardware are so intricately linked means that the system may be very sensitive to errors [1], [2], [3]. When designing a high availability systems that are used in electronic-hostile environment, errors are a matter of great concern. Space programs, patient condition monitoring system in ICU where a system cannot afford a malfunction, are vulnerable to errors. Nuclear power monitoring systems, where a single failure may cause severe destruction and real-time systems, where a missed deadline can constitute an erroneous action and a possible system failure, are a few other examples where errors are a critical issue [4], [5]. A momentary failure in banking transactions due to errors may cause a huge difference in balances. If an error causes 1→0 bit flip in the most significant bit of the register storing the amount of money deposited in a bank account, then the effect might be an unexpected change in balance [6]. A corrupted intermediate value can corrupt all subsequent computations and the executions of the whole program. Real-time embedded systems are now a central part of our lives. Reliable functioning of real-time systems is of paramount concern to the millions of users that depend on these systems every day [7]. Hence fault tolerance is essential for real-time systems [8]. To design a fault tolerant system, error detection and correction is very important. Software errors may occur when information is transmitted from one node to another node. Detection and correction of error at this particular moment is necessary to mitigate its consequence.

Various types of error detection and correction codes are used in computers. For example, for satellite applications, Hamming code and different types of parity codes are used to protect memory devices. Complex codes are not applied due to time constraints and limited resources on board [9]. There are other methods for error detection and correction such as parity codes, rectangular parity codes, three-dimensional parity codes, four-dimensional parity codes, Golay codes and BCH codes[10] whose error detection and correction rate varies from method to method. These methods have some drawbacks either low error detection and correction rate or high redundant bits. For this reason, research is needed to increase error detection and correction rate with minimum overhead.

In this paper, a high-level error detection and correction method to protect against errors is proposed. This method is based on parities for each row, column, diagonal in forward slash and backslash directions, shift parity in forward and backward direction. This method provides high error detection and correction rate that can correct up to 7 bit upsets in a data block.

The paper is organized as follows. Related work is explained in Section II. The proposed methodology is presented in Section III. Experimental analysis and results are shown in Section IV. Conclusions are drawn in Section V.

II. RELATED WORK

Several approaches are made to increase error detection and correction rate. Kishani et al. [9] corrected 3 bit errors in any combination in any position where 60 redundant bits are required for 64 data bits and can detect 100% multiple bit errors. The authors used diagonal parity in both directions along with row and column parity to detect and correct error. However in this scheme square shaped error pattern of 4 bit cannot be corrected.

PFLANZ et al. [11] proposed a method which can detect and correct 5 bit error using 3 dimensional parity codes.

This method cannot detect and correct all combination of 5 bit error in the data bits and it ignored the possibility of error occurrence in parity bits.
Sharma and Vijayakumar [12] proposed a method which utilizes horizontal, vertical and diagonal parity to detect and correct soft error in semiconductor memories. Here, authors described that their proposed system can detect and correct up to 5 bit errors. But Kishani et al. [9] proved that maximum 3 bit of any combination in any position can be detected and corrected by HVD method by Sharma and Vijayakumar's [12] method.

Anne et al. [13] introduces a method where data bits are arranged in different layers one over another to obtain a cube. The outer surfaces are made of the parity bits. This method forms a three dimensional structure of bits. It can detect 7 errors and correct 2 bit error. Another method by the authors uses 4 dimension parity bits in horizontal, vertical and diagonal in both directions which can detect 5 bit error and correct 2 bit error. Anne et al. [13] described some undetectable error pattern such as 6 bit hexagonal shaped error pattern.

Aflakian et al. [14] proposed 2 methods for error detection and correction. In the first method authors used horizontal, vertical and diagonal parity in both directions. The four bits in the corner colored red are used as parity bit for other parity bits. This scheme can detect 7 bit errors and correct up to 4 bit errors. Aflakian et al. [14] showed an undetectable 8 bit octagonal error pattern.

In the second scheme, cube of data bits are formed [14]. By dividing the data bits into planes of squares, parity is calculated for each squares using scheme of two-dimensional and two-diagonal parity-check code. A cube has 6 planes, 4 of which are covered by parity. Authors considered the forward facing face (Black colored) of the cube as 5^{th} plane. Each bit in this parity plane is calculated as the parity bit for the depth bits, except for the left column and top row [14]. The left column parity bits are obtained by considering each bit as parity bit for its row in the parity plane. Similarly, each parity bit in the top row is obtained by considering it as parity bit of its column in the parity plane.

Through this scheme can detect up to 15 bit errors and correct 4 bit errors, the complexity of this method is higher than existing methods.

III. THE METHODOLOGY TO DETECT & CORRECT ERROR

The proposed method can detect and correct error using the principle of rectangular parity codes. This paper processes right shift and left shift along with horizontal, vertical and diagonal parity in both forward & backward direction. Parity check method consists of adding a single parity bit, to make the number of 1's even.
Suppose the data block is of 64 bit and in a format of 8*8 array. The row, column, right diagonal (top left towards bottom right), left diagonal (top right towards bottom left), right shift and left shift are represented by the symbols C, R, D, D`, RS and LS. The point (R0, C0) represents the intersection between 1st row and 1st column, (C0, D0) represents the intersection between 1st column and 1st right diagonal and so on.

The data bits are sent along with check bits which are known as code word. Then receiver generates parity for received data bits. Generation of different dimensional parity occurs in parallel as it is useful in real-time and high speed applications.

Determination process of row, column, diagonal and shift parity are as follows.

A. Row & Column Parity: Each data bit in a row take part in horizontal parity code. The number of 1 is counted in a row. Then 1 or 0 is used to make the number of 1 even. This is a row parity bit. The same process is used for column parity.

0	1	2	3	4	5	6	7
1	0	0	0	0	0	0	1
1	0	0	0	0	0	1	0
0	1	0	0	0	0	1	1
0	0	0	1	0	1	0	0
1	0	0	0	0	1	0	1
1	0	1	0	1	0	1	0
1	0	0	0	0	0	1	0
0	0	1	1	1	0	0	0

Fig. 1: Parity in Vertical (Column) Direction.

0	1	0	0	0	0	0	0	1
1	1	0	0	0	0	0	1	0
2	0	1	0	0	0	0	1	1
3	0	0	0	1	0	1	0	0
4	1	0	0	0	0	1	0	1
5	1	0	1	0	1	0	1	0
6	1	0	0	0	0	0	1	0
7	0	0	1	1	1	0	0	0

Fig. 2: Parity in Horizontal (Row) Direction.

The dashed boxes represent the sequence of row and column parity.

B. Diagonal Parity: For diagonal parity, every bit of a same diagonal will take part. The number of diagonal parity bits will be (number of row + number of column - 1). There are two types of diagonal parity. One is left diagonal (top right to bottom left) and another one is right diagonal (top left to bottom right).

Left diagonal parity bits are arranged in the format as shown in Fig. 3.

0	1	2	3	4	5	6		
1	0	0	0	0	0	0	1	7
1	0	0	0	0	0	1	0	8
0	1	0	0	0	0	1	1	9
0	0	0	1	0	1	0	0	10
1	0	0	0	0	1	0	1	11
1	0	1	0	1	0	1	0	12
1	0	0	0	0	0	1	0	13
0	0	1	1	1	0	0	0	14

Fig. 3: Parity in Left diagonal (top left to bottom right direction).

The dashed boxes in Fig. 3 represent the parity in left diagonal direction. First left diagonal starts from the top left position i.e. [0][0] and it is the only bit that will take part in consisting first i.e. 0^{th} parity. For the 7^{th} left diagonal parity bit the starting bit is [0][7]. The next bit will be [row+1][column-1] i.e. [1][6]. This process will continue until column<0 or row+1>row_size.

Right diagonal parity bit are arranged in the pattern as shown in Fig. 4

	6	5	4	3	2	1	0
7	1	0	0	0	0	0	0	1
8	1	0	0	0	0	0	1	0
9	0	1	0	0	0	0	1	1
10	0	0	0	1	0	1	0	0
11	1	0	0	0	0	1	0	1
12	1	0	1	0	1	0	1	0
13	1	0	0	0	0	0	1	0
14	0	0	1	1	1	0	0	0

Fig. 4: Parity in Right diagonal (Top right to Bottom left).

The dashed boxes represent the parity in right diagonal direction. The only bit that takes part in consisting 0^{th} right diagonal parity is [0][col_position-1] i.e. [0][7]. Starting bit for 7^{th} right diagonal parity is [0][0]. The next bit will be [row+1][col+1]. This process will continue until row+1>row_size or col+1>col_size.

C. Shift Parity: It uses shift property to select bit for parity generation. There are also two types of shift parity. These are (i) Right shift parity and (ii) Left shift parity.

1) Right Shift Parity: This Paper uses 3 bit right shift parity. The pattern in right shift parity are determined using the following algorithm.

a) Step 1: start.
b) Step 2: determine row and column number
c) Step 3: row:= row+1;
   if row+1>row_size
   then stop.
d) Step 4: col:=(col+3) % col_size
e) Step 5: go to step 2.

This process is repeated for 8 rows. If number of row is n the complexity for this algorithm is O(n).
The starting bit is the first data bit i.e. [0][0]. The next bit will be [1][3]. The next one will be [2][6] and so on until last row i.e. 7^{th} row. These bits will take part in parity bit of position [0][0]. Then we will determine parity bit for [0][1] and repeat the process.

0	1	2	3	4	5	6	7
5	6	7	0	1	2	3	4
2	3	4	5	6	7	0	1
7	0	1	2	3	4	5	6
4	5	6	7	0	1	2	3
1	2	3	4	5	6	7	0
6	7	0	1	2	3	4	5
3	2	1	0	7	6	5	4

Fig. 5: Data Bit Selection for Right Shift parity

2) Left Shift Parity: This paper uses 2 bit left shift parity. In the 5^{th} row from bottom we used 5 bit shift (only for that row) as it will prevent from picking same bit position for parity i.e. [7][7] and [3][3] bit. The next bit that will take part in left shift parity is determined by the following algorithm.

a) Step 1: start
b) Step 2: determine row and column number
c) Step 3: row:=row-1
   if row<0
   then stop
d) Step 4: if row==5 && col-2<0
   then col:=col+ col_size-5
e) Step 5: else if col-2<0
   then col:=col+ col_size-2

f) Step 6: else
col:=col-2
g) Step 7: go to step 2.

If number of row is n the complexity for this algorithm is O(n). Starting bit for this process is last data bit of last row i.e. [row_position-1][col_position-2]. The first bit will be [7][7]. The next bit will be [6][5]. The next one is [5][3] and the next is [4][1]. The bit [4][1] is in the 4th row. The next will be one 5 bit left shifted which is [3][4].

The reason to shift 5 bit instead 2 bit in this certain position is to avoid mirroring of 7th row in 5th row duplicity between 7th row and 4th row.

The bit will be shifted 2 bit as usual until 0th row. Then we will determine parity bit for [7][6] and repeat this process until [7][0].

3	4	5	6	7	1	0	2
0	2	6	5	4	3	2	1
2	1	0	7	6	5	4	3
4	3	2	1	0	7	6	5
1	0	7	6	5	4	3	2
3	2	1	0	7	6	5	4
5	4	3	2	1	0	7	6
7	6	5	4	3	2	1	0

Fig. 6: Data Bit Selection for Left Shift Parity

In this paper, bits are shifted two positions towards left and three positions towards right to prevent same set of bits getting selected for right shift and left shift parity. The starting bits are different in these two parity scheme.

All these parity bits are stored in 6 arrays. Then respective parity bits are compared with the received parity bits. If there is no mismatch among them then there is no error in data bits.

The sender first sends the data bits along with check bits to the receiver. Then the receiver generates parity bits for the received data bit. These two conditions are compared to determine the erroneous row, column etc. If there is any mismatch in any row and/or column then there is an error. This error is indicated by syndrome bit. Syndrome bit is a special kind of bits which will determine by comparing sent parity and received parity bits. When we find a mismatch in a row or a column or a diagonal or a left shift or a right shift then we set syndrome bit 1 for those dimensions and take an array for storing the syndrome bits.

Two steps are used to detect and correct error. These are (i) Mark candidate bits and (ii) Refine Candidate Bits to find Error.

D. Mark Candidate Bits: For finding candidate bits, we have to consider the conjunction of data lines (at least two) in different dimensions. When a row or a column or a diagonal or a left shift or a right shift in one dimension intersects with a row or a column or a diagonal or a left shift or a right shift in another dimension then the intersecting point is a candidate bit. Candidate bit represents the mismatched parity positions. After finding candidate bits we take an array to store them. If two dimensions are not intersected then we are unable to find candidate bits. Fig. 7 shows that we are unable to find mismatched parity because they are not intersected.

1	0	0	0	0	0	0	1
1	0	0	0	0	0	1	0
0	1	0	0	0	0	1	1
0	0	0	1	0	1	0	0
1	0	0	0	0	1	0	1
1	0	1	0	1	0	1	0
1	0	0	0	0	0	1	0
0	0	1	1	1	0	0	0

Fig. 7: Erroneous Data Block

1	0	0	0	0	0	0	1
1	0	0	0	0	0	1	0
0	1	0	0	0	0	1	1
0	0	0	1	0	1	0	0
1	0	0	0	0	1	0	1
1	0	1	0	1	0	1	0
1	0	0	0	0	0	1	0
0	0	1	1	1	0	0	0

Fig. 8: Erroneous Data Block

In Fig. 7 there is error in D3 & D`11 position but there is no intersection of mismatched parity in top left to bottom right and top right to bottom left parity direction. So no candidate bit is found by this ordered pair.

In Fig. 8 there is an intersection between R1 & D3. That intersected bit is a candidate bit.

E. Candidate Bits Elimination & Refine Candidate Bits to eliminate Error: Not all candidate bits are erroneous. But all erroneous bits are among candidate bits [9]. Candidate bits are refined using the condition: In any

direction (i.e. row or column or diagonal or left shift or right shift), if there is any candidate found already but the corresponding syndrome bit is not set then that candidate selection is eliminated. The real erroneous bits are flagged. Then these bits are corrected accordingly**.**

There is always a candidate bit that can be selected to be eliminated [9]. After eliminating the candidate bits, the whole process is performed continuously to ensure error detection and correction.

IV. EXPERIMENTAL ANALYSIS & RESULTS

Any odd number of errors can be detected and corrected using odd parity or even parity methods. In this section, we have explained the reason of failure to correct 4 bit square shaped error pattern using existing parity methods. The efficiency of the proposed method is validated in this section.

Here we consider a code word with 4 bit error. The code word architecture with errors is shown in Fig. 9. The chosen error bits are in the positions [2][1], [2][5], [6][1] and [6][5].

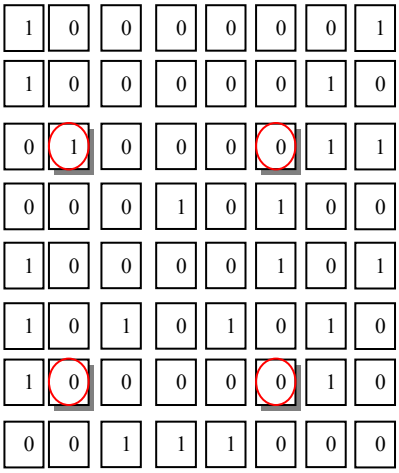

Fig. 9: Erroneous Data Block

Error correction using overlapping property requires crossing of n dimensions in one point. But the chosen erroneous bits in Fig. 9 are undetectable by overlapping propoerty.

Using the method of Kishani et al. [9], all generated candidate bits cannot be eliminated to correct the erroneous bits. The detail working principle of the proposed method is outlined brifly as follows.

A. Parity Generation & Mark Candidate Bits:

The error pattern shown in Fig. 9 has mismatched parity in D4, D12 in right diagonal parity, D`3, D`11 in left diagonal parity and LS0, LS1, LS4, LS5 in left shift parity directions. We will set syndrome bit 1 for D`3, D4, D`11, D12, LS0, LS1, LS4 and LS5 dimensions.

Now we move to mark candidate bits. We know that, Candidate bits are those bits that lie in the intersection of two mismatched parity of different dimensions. Fig. 10 shows that right diagonal dimension D4 and left diagonal dimension D`3 are intersecting in [0][3] position. That is our first Candidate bit. Then we consider right diagonal D4 and left diagonal D`11 dimensions. They cut together in [4][7] position. That is our second candidate bit. Right diagonal D12 is remaining unused until left shift dimensions are used. Star symbol represents the position of Candidate bits. Now we consider right shift parity direction. But there is no mismatched parity. In left shift parity direction the mismatched parities are LS0, LS1, LS4 and LS5.Now if we cross the left and right diagonal mismatched parity dimensions with left shift dimensions separately, we will get all the candidate bits which are shown in Fig. 10.

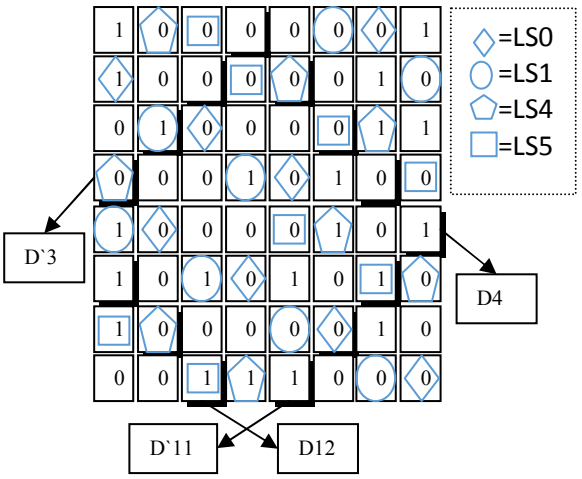

Fig.10: Crossing Left and Right Diagonals with Left shift

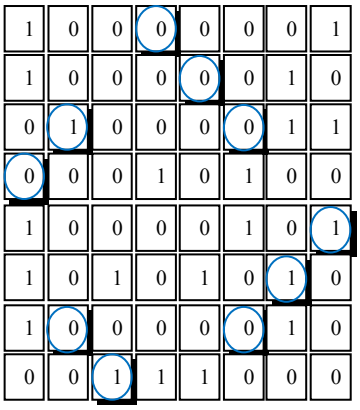

Fig. 11: Candidate Bits

From Fig. 10, we can see that the D4 dimension and left shift dimensions are intersected together. D4 intersects LS1 and LS5 separately in [1][4] and [2][5] positions. They are 3rd and 4th candidate bits. Left diagonal D`3 intersects left shift dimensions, LS4 and LS5 separately in [2][1] and [3][0] positions. These are 5th and 6th

Candidates. Using the same procedure, we can mark Candidate bits in [5][6],[6][1],[6][6] and [7][2] positions respectively.

Fig. 11 shows the positions of Cadidate bits.

B. Candidate Bits Elimination & Refine Candidate Bits to eliminate Error:

In the first row, considering the first Candidate bit, there is no mismatched parity and no syndrome bit is set. So 1st candidate bit is eliminated. There is no other candidate bit in that direction. Considering 2nd Candidate bit, which is in the 1st row, is in the intersection of D4 and LS4. That bit is alone in row direction and has no syndrome bit set. For this reason, 2nd candidate bit is eliminated. In the same process candidate bits 5, 6, 7 and 10 are eliminated. After eliminating candidate bits, the situation is shown in Fig. 12.

1	0	0	0	0	0	0	1
1	0	0	0	0	0	1	0
0	1	0	0	0	0	1	1
0	0	0	1	0	1	0	0
1	0	0	0	0	1	0	1
1	0	1	0	1	0	1	0
1	0	0	0	0	0	1	0
0	0	1	1	1	0	0	0

Fig. 12: Candidate Bits after Elimination

Candidate bit 3 & 9 are the only candidate bits in D4, D12 directions respectively & syndrome bits are set in those directions. Same process goes for candidate bit 4 & 8 as they are the only candidate bits in D`3, D`11 directions respectively where syndrome bits are set. So they are erroneous and these data are corrected accordingly.

The proposed method detects and corrects error up to 7 bits of any combination. To determine error correction coverage rate, error injection is crucial. The data are in the format of 8*8, 16*16, 32*32 and 64*64 arrays. Then all possible error combinations are generated. A simulator has been developed to inject the error in a fixed data set. Then error is detected and corrected using the proposed method. The comparison between the proposed method and existing dominant methods is shown in Table 1.

The proposed method have slight higher bit overhead compared to existing methods such as Golay[10], BCH[10] & RS[10] method but with low code rate. As we can see, bit overhead is higher for the proposed method. On the other hand code rate is low.

TABLE I. THE COMPARISON BETWEEN THE PROPOSED METHOD AND EXISTING METHODS.

Error Detection & Correction Method	**Golay(23,12,7)**	**BCH(31,16,7)**	**RS(64,57,7)**	**Proposed Method**
# of Data Bits	11	15	7	64
# of Parity Bits	12	16	57	62
# of Codeword Bits	23	31	64	126
Bit Overhead (%)	91.67	93.75	12.28	96.86
Code Rate (%)	52.17	51.61	90.74	51.62
Error Bit Correction	3	3	3	7

We know that, these two terms are inversely proportional, when one increases other one de-creases. However, the proposed method can detect and correct up to 7 bit errors which is much higher than existing methods.

V. CONCLUSIONS

The paper proposes a novel error correction coding scheme with a high error correction rate. The proposed method works well for large number of data. While sending a large number of data, the possibility of error occurrence increases. However, the existing dominant error correcting coding approaches could detect and correct up to three bit errors. The proposed method shows the ability to detect and correct up to seven bit errors by incurring a little bit overhead. For the systems where reliability is a matter of concern, the proposed method is effective. Further studies can be performed to find the scope or minimizing bit overhead with the increase of correction rate.

REFERENCES

[1] Muhammad Sheikh Sadi, D.G. Myers, and Cesar Ortega Sanchez, "Component Criticality Analysis to Minimizing Soft Errors Risk*," In International* Journal *of Computer Systems Science and Engineering,* CRL Publishing, vol. 25, No. 5, 2010.

[2] S. Sankaranarayanan and B. Vasic, "Iterative decoding of linear block codes: A parity-check orthogonalization approach," IEEE Transactions on Information Theory, vol. 51, no. 9, pp. 3347–3353, September 2005.

[3] V. Kumar and O. Milenkovic, "On graphical representations of algebraic codes suitable for iterative decoding," IEEE Communications Letters, vol. 9, no. 8, pp. 729–731, August 2005.

[4] Muhammad Sheikh Sadi, Mizanur Rahman Khan, Nazim Uddin, and Jan Jürjens, "An Efficient Approach towards Mitigating Soft Errors Risks," *Signal& Image Processing: An International Journal (SIPIJ),* vol. 2, No. 3, September 2011.

[5] Ne'am Hashem Ibraheem "Error-Detecting and Error-Correcting Using Hamming and Cyclic Codes," IEEE Transactions on

Information Theory, vol. 51, no. 9, pp. 3347–3353, September 2005.

[6] R. W. HAMMING, “Error Detecting and Error Correcting Codes,” Bell System Tech. Jour., 29 (1950): 147–160.

[7] Xue Liu, Hui Ding, Kihwal Lee, LuiSha, Marco Caccamo “Feedback Fault Tolerance of Real-Time Embedded Systems – Issues and Possible Solutions” Newsletter ACM SIGBED, Volume 3 Issue 2, Pages 23-28, April 2006

[8] D. Rubenstein, J. Kurose, D. Towsley ``Real-Time Reliable Multicast Using Proactive Forward Error Correction,'' Proceedings of NOSSDAV *'98*, (Cambridge, UK, July 1998).

[9] MostafaKishani, Hamid R. Zarandi, Hossein Pedram, Alireza Tajary, Mohsen Raji, Behnam Ghavami “HVD: horizontal-vertical-diagonal error detecting and correcting code to protect against with soft errors” Des Autom Embed Syst (2011) 15:289–310DOI 10.1007/s10617-011-9078-2

[10] Muhammad Imran, Zaid Al-Ars, Georgi N. Gaydadjiev “Improving Soft Error Correction Capability of 4-D Parity Codes” 14th IEEE European Test Symposium, May 2009

[11] M. PFLANZ, K. WALTHER, C. GALKE AND H.T. VIERHAUS “On-Line Techniques for Error Detection and Correction in Processor Registers with Cross-Parity Check” Journal of Electronic Testing October 2003, Volume 19, Issue 5, pp 501-510

[12] Shalini Sharma, Vijayakumar P. “An HVD Based Error Detection and Correction of Soft Errors in Semiconductor Memories Used for Space Applications” International Conference on Devices, Circuits and Systems (ICDCS), 2012. Print ISBN 978-1-4577-1545-7, pp 563 - 567

[13] Naveen Babu Anne, Utthaman Thirunavukkarasu, Dr. ShahramLatifi “Three and Four-dimensional Parity-check Codes for Correction and Detection of Multiple Errors” Proceedings of the International Conference on Information Technology: Coding and Computing (ITCC’04) 0-7695-2108-8/04, pp. 267-282, 2004.

[14] Danial Aflakian, Dr.Tamanna Siddiqui, Najeeb Ahmad Khan, Davoud Aflakian “Error Detection and Correction over Two-Dimensional and Two-Diagonal Model and Five-Dimensional Model” (IJACSA) International Journal of Advanced Computer Science and Applications, Vol. 2, No. 7, 2011