Soft Error Tolerance using Horizontal-Vertical-Double-Bit Diagonal Parity Method

Md. Shamimur Rahman,
Muhammad Sheikh Sadi,
Sakib Ahammed
Dept. of Computer Science &Engineering
Khulna University of Engineering & Technology
Bangladesh
e-mail:shamimur052@gmail.com,
sheikhsadi@gmail.com, sakib.kuet10@gmail.com

Jan Jurjens
Dept. of Computer Science
TU Dortmund
Germany
e-mail:jan.juerjens@isst.fraunhofer.de

***Abstract*—The likelihood of soft errors increase with system complexity, reduction in operational voltages, exponential growth in transistors per chip, increases in clock frequencies and device shrinking. As the memory bit-cell area is condensed, single event upset that would have formerly despoiled only a single bit-cell are now proficient of upsetting multiple contiguous memory bit-cells per particle strike. While these error types are beyond the error handling capabilities of the frequently used error correction codes (ECCs) for single bit, the overhead associated with moving to more sophisticated codes for multi-bit errors is considered to be too costly. To address this issue, this paper presents a new approach to detect and correct multi-bit soft error by using Horizontal-Vertical-Double-Bit-Diagonal (HVDD) parity bits with a comparatively low overhead.**

***Keywords—Soft Error Tolerance, Horizontal Parity, Vertical Parity, Double-Bit Diagonal Parity*.**

I. INTRODUCTION

The advent of new technologies for implementation, along with non-functional constraints (dependability, timeliness, power, cost and time-to-market), has seen the design of embedded systems become more challenging and complex [1]. The complexity of such systems is growing at a high pace through the integration of mixed-criticality applications (both safety-critical and non-safety-critical), through the high interaction of distributed application. Embedded systems engineering has evolved from designing single CPU systems (System-on-a-Chip (SoC) concept) to concurrent computing structures (Multi-Processor-System-on-a-Chip (MPSoC) and Network-on-a-Chip (NoC) concepts) often in the form of a distributed network [2], [3]. When designing 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 soft errors [4]. 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 soft error is a critical issue [5], [6], [7].

The impact of soft errors is such that action is needed to increase a system's tolerance or to lower the risk of soft errors in the system. Prior research into soft errors has focused primarily on circuit level solutions, logic level solutions, spatial redundancy and temporal redundancy [8]. However, in all cases, the system is vulnerable to soft error problems in key areas. Further, in software-based approaches, the complex use of threads presents a difficult programming model. Hardware and software duplication suffers not only from overheads due to synchronizing duplicate threads, but also from inherent performance overheads due to additional hardware. Hardware-based protection techniques based on duplication often suffers from high area, time and power overheads [9], [10], [11].

Various types of error detection and correction codes are used in computers. For example, for satellite applications, Hamming code [12] 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 [13]. There are other methods for error detection and correction such as parity codes, rectangular parity codes, BCH codes [14], N-Dimensional parity code [15], and Golay codes [16] whose error detection and correction rate varies from method to method. However, majority of these methods are still facing low error detection and correction rate and/or high information overhead. For this reason, there is a great need of further research to increase the error detection and correction rate with minimal overhead.

In this paper, a high-level error detection and correction method to protect against soft errors is proposed. This method is based on parities for each row, column and double bit diagonal parity in backward slash directions. The HVDD method provides high error detection and correction rate that can correct up to 3 bit upsets with low bit overhead in a data block.

The rest of this paper is organized as follows. We provide some related work in Section II. The proposed methodology is

presented in Section III. We give an example in Section IV. In Section V, Experimental analysis is shown. Conclusions are drawn in Section VI.

II. RELATED WORK

Several approaches are made to increase error detection and correction rate.

Pflanz et al. [17] 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 [18] 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. [13] proved that maximum 3 bit of any combination in any position can be detected and corrected by this method.

Anne et al. [19] 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 bit errors and correct 2 bit error. Anne et al. [15] described some undetectable error pattern such as 6 bit hexagonal shaped error pattern.

Aflakian et al. [20] proposed two methods for error detection and correction. In the first schema authors used horizontal, vertical and diagonal parity in both directions. This scheme can detect 7 bit errors and correct up to 4 bit errors. In the second scheme, cube of data bits are formed. Though this scheme can detect up to 15 bit errors and correct 4 bit errors, the complexity of this method is higher than existing methods.

Kishani et al. [13] 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. For this reason, their bit overhead is 73.12% which is very high. The method proposed by Kishani et al. [13] is shown in Fig. 1.

However, our correction method uses three directions of parity instead of four as shown by Kishani et al. [13]. And it can detect 100% errors in any combination and correct up to three bit errors for almost all combinations. Even the few combinations which were out of the scope of the proposed first approach are also corrected by the proposed second approach. The information overhead of our approach is much lower than that of Kishani et al. [13].

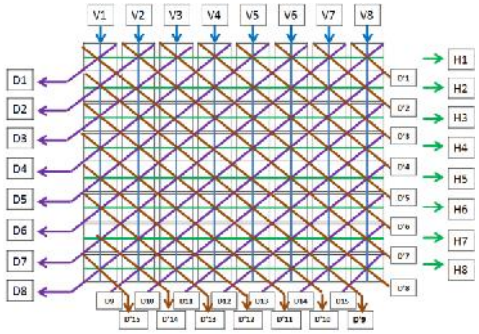

Fig. 1. The structure of the proposed method by Kishani et al. [13].

So the main differences between the proposed method and that of Kishani et al. [13] are:

1) The proposed method is relatively less complex than that of Kishani et al. [13] since it uses three directions for parity bits generations whereas Kishani et al. [13] uses four directions for parity bits generations.

2) The proposed method consumes lesser memory (for lesser check bits) than that of Kishani et al. [13].

III. THE PROPOSED METHODLOGY TO DETECT & CORRECT ERROR

The proposed method detects and corrects the soft error by using the check bits that are smeared on the double bit diagonal (DD) along with horizontal (H) and vertical (V) parity in forward direction.

A. Detection Algorithm

We take a datablock of 64 bit and in a format of 8*8 array. In order to detect bit upsets in the codeword, all parity bits in the receiver side should be calculated again. All the mentioned parity bits can be calculated independently in parallelize.While computing the horizontal parity bits, vertical parity bits can be computed simultaneously. This property is very operational in real-time and relatively high speed applications. After detecting the misapprehend between the received parity bits and the calculated ones, the detection algorithm will be stopped. Along with horizontal and vertical dimensions, double bit diagonal parity bits are calculated by using the following algorithm.

For double-bit diagonal parity, the matrix should be evenly ordered (row and column sizes should be even). Here, every two bit of a same row will take part in double-bit diagonal parity calculation. In Fig. 3, the diagonals are indexed from top right corner towards bottom left corner. The number of diagonal parity bits will be (number of rows+ (number of column-2)/2).

Step 1: Initialize number of rows and number of column from double-bit diagonal array
Step 2: Select two cells from the top right hand corner
Step 3: Select two cells from the left hand side of the previous selection in the first row
Step 4: Select two cells from the next row and next column (next to starting column of Step 3) and continue till last coulmn is reached in the same double-bit diagonal(D2) as shown in Fig. 3.
Step 5: Go to Step 3 untill the first column is selected and generate all double-bitdiagonals (D3-D4 in Fig. 3).
Step 6: Select two cells from the next row and continue till last coulmn is reached in thesame double-bit diagonal (D5) as shown in Fig. 3.
Step 7: Go to Step 6 untill last row is selected and generate all double-bit diagonals (D6-D11 inFig. 3).

Fig. 2. Double-Bit Diagonal parity detection algorithm.

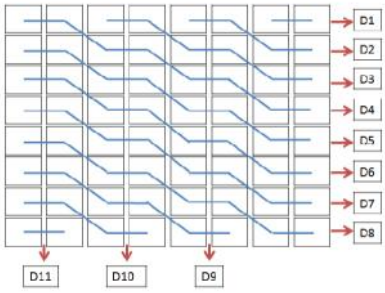

Fig. 3. Parity in Diagonal (Double bit) direction.

B. Correction Algorithm

Any difference in parity bits in horizontal, vertical and diagonal directions will set the corresponding syndrome bit to one. Syndrome bit is a special kind of bit which is determined by comparing sent parity and received parity. We need four arrays which clutch horizontal, vertical, and diagonal parity bits and syndrome bits respectively. Among the first three arrays, two are chosen and all ordered pairs of them are produced. Through every ordered pair, there may be found maximum two candidate bits. It is also mentioned that no candidate bit may be determined from some case of the ordered pair. These status are shown in Fig. 4. In the case of Fig. 4(b), two parity bits are erroneous. So, the horizontal and diagonal bits are not intersected and there is no candidate bit.

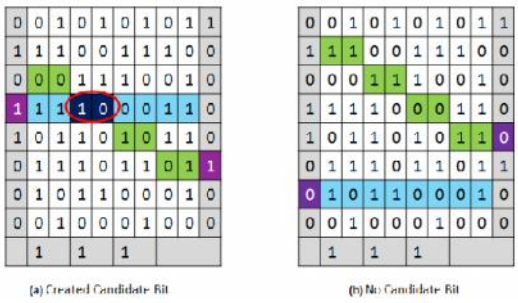

Fig. 4. The created candidate bit and no candidate bit of the intersection.

According to Kishani et al. [13], all of the factual bit upsets are included in candidate bits and if a candidate bit is alone in one row of horizontal or column of vertical or diagonal dimension (i.e. there is not any other candidate bit in the same row of horizontal or column of vertical or diagonal dimension) and if there is a syndrome bit which is set to one in a row or column or diagonal, the candidate bit is faulty and must be corrected. There is always a candidate bit that can be selected to be eliminated [13]. In Fig. 5 We show a pattern where three bit errors occur but can detect two error bits, one error bit is missing. Hence, to detect this type of error, we use a Lemma as follows.

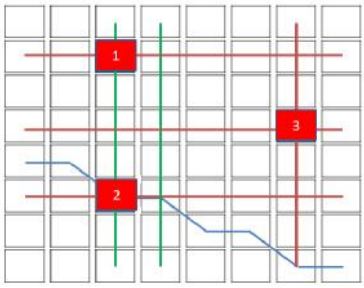

Fig. 5. Error pattern which was detected by the Lemma.

However, it is possible to detect this type of error if we can consider the following lemma.

Lemma If there are errors in the line of intersection between diagonal and horizontal, then take a vertical line to detect those candidate bits. And if there are errors in the line of intersection between diagonal and vertical, then take a horizontal line to detect those candidate bits.

Proof

Assume that there are three bits upsets which are in the pattern as shown in Fig. 6. As a result there are two horizontal, two vertical and three diagonal parity mismatch. Using normal procedure we can detect error 2 & 3 but error 1 can't. To detect error 1 we take two vertical lines from the intersection between diagonal andhorizontal which represent as green color shown in Fig. 5. So, all errors are detected as candidate bits.

C. HVDD Implementation

The proposed implementation of HVDD coding arrangement is shown in Fig. 6. The figure shows 91 bit codeword resided of 64 data bits and 27 supplementary bits for parity bits. This planning is decided to have better error correction treatment. Note that, the overhead of a codeword depends on the number of additional bits compared to the whole memory block size. The memory segregated into appropriate number of slice and the codeword is calculated for the slice.

IV. AN EXAMPLE OF ERROR DETECTION AND CORRECTION

In this section, we consider 3 errors occurrence in a codeword and show how the proposed methodology corrects it. The code word architecture with errors (the red (dark)

squares) is shown in Fig. 7. The detection and correction method of HVDD are shown as follows.

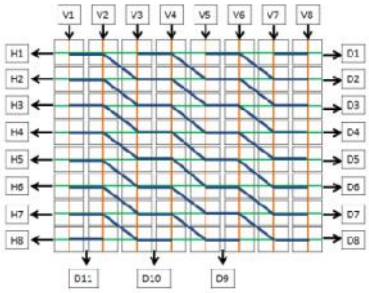

Fig. 6. Parity bits for a given memory block.

A. *Regenerate Parity Bit*

We can compare regenerating parity bits with their original ones. Any differences is a sign of error on bits that generate parity. In Fig. 8, we indicate the mistaken parity bits. The parity bits of horizontal, vertical and diagonal parities are shown in the architecture.

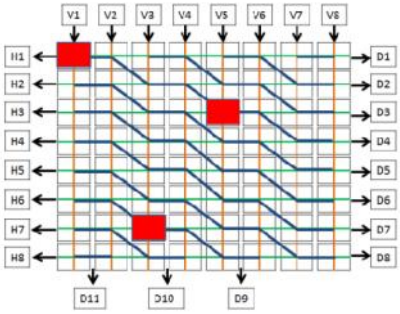

Fig. 7. Code word architecture with 3 errors.

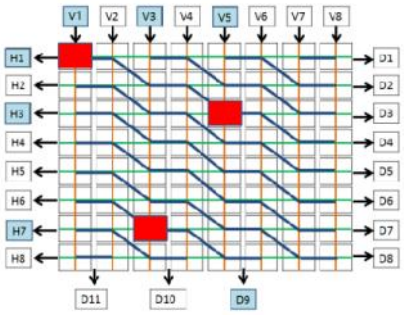

Fig. 8. Code word with parity codes.

B. *Mark Cadidate Bits*

Each parity bit has a corresponding line of codeword bits that generate it. Lines of each mistaken parity bit are considered. When we intersect two lines, if there is a conjunction then it is a candidate bit. Candidate bit represent the mismatched parity positions. After finding candidate bits, we need to store the address of each bit in an array. The candidate bits are marked in the Fig. 9.

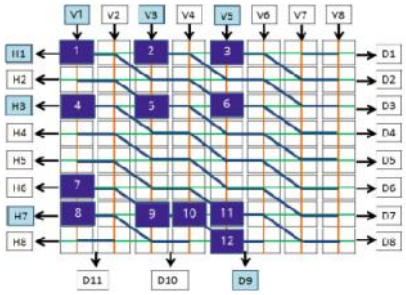

Fig. 9. Candidate bits..

C. *Refine Candidate Bits To Find Error*

All the candidate bits are not erroneous but all erroneous bits are candidate bits. To refine candidate bits, we use the condition: in any direction (i.e. row or column or diagonal), if there is any candidate bit found already but the corresponding syndrome bit is not set then that candidate bit is eliminated. If there is a syndrome bit which is set to one in a row or column or diagonal, the candidate bit is erroneous and must be corrected.

For example, consider candidate 7 in Fig. 9. It is the only candidate bit in line of H6 (parity 6 in horizontal direction) and H6 is not a erroneous parity. Therefore, candidate 7 is not an error. So, we remove this bit from candidate bits array. According to the rules, candidates 10 and 12 are also removed. After this step, the architecture is shown in Fig. 10.

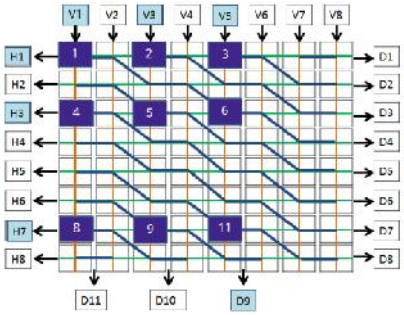

Fig. 10. Refining candidate bits: First Step.

Now consider candidate bits 1, 3, 8 and 11 from Fig. 10. Candidate 3, 8 and 11 are the only bits in direction D2, D10 and D8. There is no mistaken parity in that direction. So, these candidate are not erroneous and should be removed. But candidate 1 is not the only bit of diagonal D4 as well as there is syndrome bit set in both horizontal and vertical direction. So we don't take any decision for candidate bit 1. After this step the memory cell is shown in Fig.11.

Candidate bits 6 and 9 are alone (Fig. 11) in direction of V_5 and H_7 respectively. V_5 and H_7 are mistaken parity bits, so candidate bits 6 and 9 are errors and should be corrected and

corresponding syndrome bit are changed to 0. The architecture's status at this time is shown in Fig. 12.

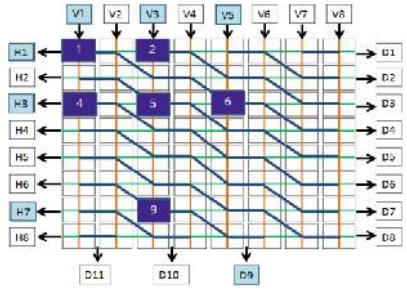

Fig. 11. Refining candidate bits: Second Step.

Candidate bits 6 and 9 are alone (Fig. 11) in direction of V_5 and H_7 respectively. V_5 and H_7 are mistaken parity bits, so candidate bits 6 and 9 are errors and should be corrected and corresponding syndrome bit are changed to 0. The architecture's status at this time is shown in Fig. 12.

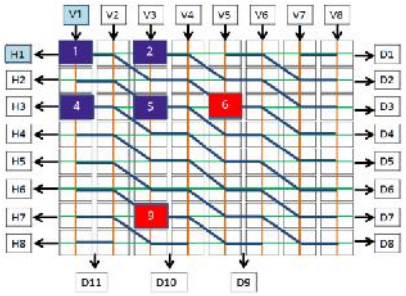

Fig. 12. Refining candidate bits: Third Step.

Now we have candidate 1, 2, 4 and 5 (Fig. 12). Candidate 2, 4 and 5 are the only bit of D3, D5 and D6 respectively. But there is no mistaken parity in those direction. So, these candidate bits are not erroneous and should be removed. Then only candidate 1 is remaining. Because of setting syndrome bit in both horizontal and vertical directions, the candidate bit is erroneous and should be corrected. After that the final memory block without any databit error is shown in Fig. 13.

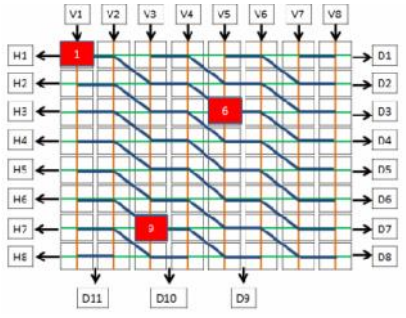

Fig. 13. Refining candidate bits, final step.

V. EXPERIMENTAL ANALYSIS

The proposed HVDD method can detect all combinations of errors and correct up to 3 bit errors. Table 1 compares the HVDD method with existing dominant methods. This comparison is based on bit overhead and code rate. The typical formulas to calculate the bit overhead and code rate is shown as follows.

$$\text{Bit Overhead} = \frac{\text{Number of Parity Bits}}{\text{Number of Data Bits}} = \frac{27}{64} = 42.19\% \quad (1)$$

$$\text{Code Rate} = \frac{\text{Number of Data Bits}}{\text{Number of Codeword Bits}} = \frac{64}{91} = 70.33\% \quad (2)$$

As shown in Table 1, HVDD performs better than Golay[14], BCH[16], and HVD[13] with respect to bit overhead. The first column of Table. 1 shows existing dominant methods, second and third column show the dataword sizes and corresponding parity bits. The fourth column presents the codeword sizes which is the summation of datawords and parity bits. The fifth column and sixth column show the bit overhead and code rate respectively that are calculated using Equation 1 &Equation2. In the first column,Golay(23,12,7) represents that in the method proposed by Golay[14], they used the examples which have the codeword of size 23 bits, parity bits of size 12 bits and hamming distance of 7. BCH (31, 16, 7) represents same meaning as Golay(23,12,7). HVD (64) and HVDD (64) represent the dataword sizes of 64 bits.

The bit overhead of Golay [14], and BCH [16] are 91.67% and 93.75% respectively which shows higher overhead than the proposed method. InHVD [13], there are 60 parity bits for 64 bits data block whereas the proposed method requires only 27 bits parity bits for the same dataword. So, the bit overhead of the proposed method is 42.19%, whereas that of HVD [13] is 78.12%.

TABLE1. Comparison of triple bit error correcting codes

Error Correcting Method	# of Data Bits	# of parity bits	# of codeword bits	Bit Overhead (%)	Coderate (%)
Golay(23,12,7)	11	12	23	91.67	52.17
BCH(31,16,7)	15	16	31	93.75	51.61
HVD(64)	64	60	124	78.12	56.14
HVDD(64)	64	27	91	42.19	70.33

The information supplied in Table. 1 are plottedin Fig. 14 for better visual understanding bit overhead and coderate between the error correcting methods shown in Table 1. X-axis represents the triple bit correcting methods and Y-axis represents the percentage of parameter. The blue and red polygon are representing the bit overhead and code rate respectively.

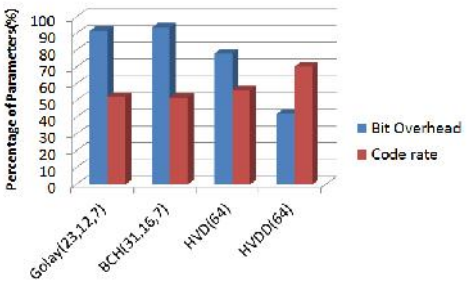

Fig. 14. Comparison of triple bit error correcting codes

From Fig .14, It can be observed that Golay (23, 12, 7) performs error detection & correction with 12 data bits, 11 parity bits and hamming distance of 7. They form a set of codeword with 23-bits long with high bit overhead and low code rate than HVDD. BCH (31, 16, 7) is a form of 16 data bits, 15 parity bits with 7 hamming distance. Encode and decode process of BCH codes have a high delay compared to others. So, this correction scheme has also complex hardware implementation. HVD, which developed the error detection & correction scheme based on parity method, have high bit overhead and low code rate compared to the proposed method.

From the experimental analysis it can be observed that the proposed method can detect and correct up to three bits errors in any combinations. To detect candidate bits, we take cross-section of two directions among the three. This method can't detect any intersection where four bits errors are in rectangle pattern. We faced problem to correct errors if these are in a closed triangle pattern. We solved this problem in this particular situation that may rarely happen by using our previous method as shown in Sadi et al. [21].

VI. CONCLUSIONS

This paper proposes a high-level error detection and correction method (HVDD) to protect against soft errors. This method is based on parities for each row, column and double-bit diagonal (in backward) directions. The HVDD method provides error detection capabilities for all possible errors and can correct up to 3 bit errors in a data word. Though it faces problem to detect a particular error pattern (which is later solved by our previous method), it has a significant contribution in reducing information overhead than existing dominant methods to detect same amount of erroneous bits. To solve that particular problem can be captured as an extension of the current research.

REFERENCES

[1] Manoochehri**,**Annavaram**,** Dubois**,** "Extremely Low Cost Error Protection with Correctable Parity Protected Cache,"*Computers, IEEE Transactions on* (Volume:63, Issue: 10), pp. 2431-2444, September 2014.

[2] Sanchez-Macian,Reviriego, Maestro, "Hamming SEC-DAED and Extended Hamming SEC-DED-TAED CodesThroughSelective Shortening and Bit Placement,"*Device and Materials Reliability, IEEE Transactions on* (Volume:14, Issue: 1), pp. 574-576, March 2014

[3] Reviriego, Flanagan, Pontarelli, Maestro, "An Efficient Technique to Protect Serial Shift Registers Against Soft Errors,"*Circuits and Systems ll: Express Briefs. IEEE Transactions on*(Volume:60, Issue: 8), pp. 512-516, August 2013.

[4] Neale, Sachdev, "A New SEC-DED Error Correction Code Subclass for Adjacent MBU Tolerance in Embedded Memory,"*Device and Materials Reliability, IEEE Transactions on* (Volume:13, Issue: 1),pp. 223 – 230, March 2013

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

[6] Ne'amHashemIbraheem "Error-Detecting and Error-Correcting Using Hamming and Cyclic Codes," *IEEE Transactions on Information Theory*, vol. 51, no. 9, pp. 3347–3353, September 2005.

[7] Ferreyra PA, Marques CA, Ferreyra RT, Gaspar JP,"Failure map functions and accelerated meantime to failure tests: new approaches for improving the reliability estimation in systems exposed to single event upsets," *IEEE Trans NuclSci* 52(1), pp. 494–500 .

[8] ArgyridesC, Zarandi HR, Pradhan DK "Multiple upsets tolerance in SRAM memory," *International symposium on circuits and system*, New Orleans, LA, May 2007

[9] A. A. Jerraya, "Long term trends for embedded system design," *Digital System Design*,Euromicro Symposium on, 2004, pp. 20-26

[10] Hyun Yoon D, Erez M (2009) "Memory mapped ECC: low-cost error protection for last level caches," *ACM SIGARCHComputArchit News* 37(3):116–127 .

[11] M. S. Sadi, D. G. Myers, C. O. Sanchez, "A Design Approach for Soft Error Protection in Real-Time Embedded Systems," *19th Australian Conference on Software Engineering, ASWEC* 2008, pp. 639-643

[12] Hentschke R, Marques R, Lima F, Carro L, Susin A, Reis R, "Analyzing area and performancepenalty of protecting different digital modules with hamming code and triple modular redundancy," *International symposium on integrated circuits and systems design*,pp. 95–100

[13] Mostafa Kishani, Hamid R. Zarandi, HosseinPedram, AlirezaTajary, Mohsen Raji, BehnamGhavami "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.

[14] 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.

[15] Rubinoff M "N-dimensional codes for detecting and correcting multiple errors,"*ComunACM* 545–551 .

[16] http://mathworld.wolfram.com/GolayCode.html

[17] M. Pflanz et al.,"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.

[18] 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.pp. 563 – 567.

[19] NaveenBabu Anne, UtthamanThirunavukkarasu, 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)*, pp. 267-282, 2004.

[20] DanialAflakian, Dr.TamannaSiddiqui, Najeeb Ahmad Khan, DavoudAflakian "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.

[21] M.S. Sadi, M.S. Rahman, K.M. ImrulKayesSikdar, "Error detection & correction using horizontal-vertical-diagonal-shift method," *Electrical Engineering and Information & Communication Technology (ICEEICT), 2014 International Conference on*, pp. 1-7 , April 2014