Generating Burst-error Correcting Codes from Orthogonal Latin Square Codes – a Graph Theoretic Approach

Rudrajit Datta and Nur A. Touba

Computer Engineering Research Center
University of Texas at Austin, Austin, TX 78712
rudrajit@utexas.edu, touba@ece.utexas.edu

Abstract

The paper proposes a scheme by which an Orthogonal Latin Square code (OLS) can be modified to correct burst-errors of specific length. The method discussed in this paper models it as a graph coloring problem where the goal is to resolve conflicts in the existing OLS code in order for it to correct burst-errors. Conflicts are resolved by reordering and/or reorganizing existing parity relations by inclusion of extra check bits. The graph coloring approach tries to minimize the number of additional check bits required. The final OLS code after reordering and/or reorganizing would be capable of correcting burst-errors of specific length in addition to its original error correction capabilities.

Keywords: graph coloring; burst error; Orthogonal Latin Square

1. Introduction

Single event upsets (SEU) and soft errors generated by ionizing particles or neutron interactions with semiconductor devices have been identified as a critical and possibly dominant failure mechanism in modern CMOS circuits**.** Error detection and correction schemes in memories and microprocessor caches are common and drastically reduce the externally observable error rate.

A key consideration for these protection schemes is the treatment of multiple bit errors that can be generated when adjacent bits fail as a result of a single strike. Studies have shown that 1-5% of single event upsets (SEUs) can cause multiple-bit errors (MBUs) [Satoh 00], [Makihara 00], [Kawakami 04]. Most MBUs will affect nearby cells.

These events can prevent the detection of an error in parity protected circuits, or make an error uncorrectable in spite of the use of Error Correction Codes (ECC). Bit interleaving is commonly used to minimize the error rate contribution of multi-bit errors. It refers to a memory layout architecture in which physically adjacent bits belong to different logic words. The result is that from an error detection and correction standpoint, two adjacent failing bits appear as two single bit errors rather than **as** a double bit error in the logic word. Bit interleaving rules are often defined as the minimum physical distance separating two bits belonging to the same logic word. The quantification of their effectiveness requires a detailed understanding of the multi-bit failure probabilities and operating parameter sensitivities which are generally not available in the open literature [Maiz 03]. However bit interleaving would be limited by the width of the memory bus.

Moreover with continuous voltage scaling multiple-bit errors pose an important challenge, especially for memory sub-systems. A dramatic increase in MCU rates relative to SBU is projected for geosynchronous orbits, where direct ionization by heavy-ions dominates [Seifert 08].

In the non-volatile memory space, multilevel cell (MLC) NAND flash memories, which are widely used in mobile and wireless systems, have inferior data retention times compared to single bit cell (SLC) NAND flash. Although multi-leveling cell (MLC) improves memory density and performance of memory storage systems in general, they would also be prone to a greater number of errors caused due to shift of threshold voltage during cell programming operations [Micheloni 06] [Chen 08]. Such systems would require stronger ECC than traditional single error correction codes. Therefore, the data reliability has become an important issue in most communication and storage systems for high speed operation and mass data process.

In this paper, a novel method is proposed for designing a multiple error correcting code specifically targeted towards correcting burst errors. This paper shows how, given an Orthogonal Latin Square code, it can be converted into a code capable of correcting burst errors of a specific length in addition to its original capacity with minimal overhead.

The paper is organized as follows, section two talks about other knows methods of multi-bit error detection., section three explains OLS codes, section four explains our proposed methodology followed by the results in section five. Finally section six concludes the paper along with experimental results.

2. Related Work

For high defect rates, memory repair schemes based on spare rows and columns are not effective. Much higher levels of redundancy are required that can tolerate multi-bit errors. The two-dimensional ECC proposed by [Kim 07] tolerates multiple bit errors due to non-persistent faults, but is slow and complicated to decode.

Conventional SEC-DED codes can only detect, but not correct, double-bit errors. In [Dutta 07], it was shown that by carefully selecting and ordering the columns in the H-matrix for an SEC-DED code, it is possible to correct all adjacent double-bit errors in addition to correcting all single bit errors thereby creating an SEC-DAEC code. Since the most likely double-bit errors will be adjacent, this is very useful. The limitation of SEC-DAEC codes is that they may not detect all non-adjacent double-bit errors. While scaling up conventional parity check code for correcting multi-bit errors requires less check bits, the additional number of syndromes that needs to be stored for correction purposes makes parity check matrices an unattractive solution for multi-bit errors.

In some cases, check bits are used along with spare rows and columns to get combined fault-tolerance. In [Stapper 92], interleaved words with redundant word lines and bit lines are used in addition to the check bits on each word. [Su 05] proposes an approach where the hard errors are mitigated by mapping to redundant elements and ECC is used for the soft errors. Such approaches will not be able to provide requisite fault tolerance under high bit error rates when there are not enough redundant elements to map all the hard errors.

[Micheloni 06] proposed a scheme that uses Bose- Chaudhuri-Hocquenghem (BCH) codes to correct multiple errors in NAND flash. However, NAND flash memory systems process with a large size of data such as a page or a block unit. Hence, BCH codes may not be appropriate for a NAND flash controller [Chen 08]. [Kim 10] proposed a product code using a Reed-Solomon code scheme for NAND flash memories, capable of correcting multiple bit errors. Although Reed-Solomon codes are good for burst errors, the decoding time would be enormous (> 500 clock cycles) [Kim 10].

The application of OLS codes for handling the high defect rates in low power caches as described in [Christi 09] provides a more attractive solution. While OLS codes require more redundancy than conventional ECC, the one-step majority encoding and decoding process is very fast and can be scaled up for handling large numbers of errors as opposed to BCH codes, which while providing the desired level of reliability requires multi-cycles for decoding [Lin 83].

Since most multi-bit errors are likely to result in adjacent bit failures, a burst error code seems like an optimal solution. In this paper we show how a regular OLS code can be converted to correct burst errors of specific lengths. This way we can combine the single-step decodable facet of OLS codes along with its high error correction capability. The capability of OLS codes to correct multiple errors in a single cycle is synergistic with a high performance memory system, in particular MLC NAND. This way even in the presence of multiple errors, a likely scenario in MLC NAND systems [Micheloni 06], the error detection and correction step would not be a bottleneck in the way of improved memory performance. Our proposed solution preserves this property of OLS codes while enabling it to correct burst errors with minimal overhead.

3. Orthogonal Latin Square Codes

A Latin square [Hsiao 70] of order (size) m is an m x m square array of the digits 0, 1, . . . , m - ***1,*** with each row and column a permutation of the digits 0,1, … , m - 1. Two Latin squares are orthogonal if, when one Latin square is superimposed on the other, every ordered pair of elements appears only once.

In general, a t-error correcting majority decodable code works on the principle that $2t + 1$ copies of each information bit are generated from $2t + 1$ independent sources. One copy is the bit itself received from memory or any transmitting device. The other $2t$ copies are generated from $2t$ parity relations involving the bit. By choosing a set of h Latin squares that are pair-wise orthogonal, one can construct a parity check matrix such that the number of 1's in each column is $2t = h + 2$. The orthogonality condition ensures that for any bit d, there exists a set of $2t$ parity check equations orthogonal on d_i, and thus makes the code self-orthogonal and one-step majority decodable. One-step majority decoding is the fastest parallel decoding method. The t-error correcting codes generated by OLS codes [Hsiao 70] have m^2 data bits and $2tm$ check bits per word.

Let the m^2 data bits be denoted by a vector:

$$D = [d_0, d_1, ...d_{m^2-1}] \quad \text{..... (1)}$$

Then the *2tm* check-bit equations for *t*-error correcting are obtained from the following parity check matrix H:

$$H = \left[\begin{array}{c|c} M_1 & \\ M_2 & \\ M_3 & I_{2tm} \\ \vdots & \\ M_{2t} & \end{array}\right] \quad \ldots\ldots\ldots\ldots (2)$$

$\boldsymbol{I_{2tm}}$ is an identity matrix of order *2tm* and $\boldsymbol{M_1}, \ldots, \boldsymbol{M_{2t}}$ are submatrices of size *m* x m^2. These submatrices $\boldsymbol{M_1}....\boldsymbol{M_{2t}}$ have the form

$$M_1 = \begin{bmatrix} 11\ldots1 & \ldots & \\ \vdots & 11\ldots1 & \vdots \\ \vdots & \ldots & 11\ldots1 \end{bmatrix}_{mxm^2} \quad \ldots\ldots\ldots (3)$$

$$M_2 = \begin{bmatrix} I_m & I_m & \ldots & \ldots & \ldots & I_m \end{bmatrix}_{mxm^2} \ldots (4)$$

The matrices $\boldsymbol{M_1}....\boldsymbol{M_{2t}}$ are derived from the existing set of orthogonal Latin squares $L_1, L_2, \ldots, L_{2t-2}$ of size *m* x *m*. Denote the set of Latin squares as,

$$L_1 = \left[l_{ij}^1\right]_{mxm}$$
$$L_2 = \left[l_{ij}^2\right]_{mxm}$$
$$\vdots$$
$$\vdots$$
$$L_{2t-2} = \left[l_{ij}^{2t-2}\right]_{mxm}$$

where $l_{ij} \in \{1, 2, \ldots, m\}$

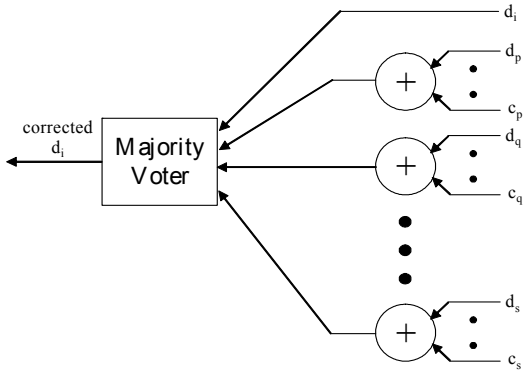

Figure 1. Decoding Data Bit d_i with Majority Voter

Once the *H*-matrix is constructed, the decoding for each data bit is done using a majority voter as illustrated in Fig. 1. When decoding data bit d_i, the set of bits in each of the *2t H*-matrix rows that d_i is present in are XORed together and serve as an input to a majority voter along with d_i itself giving a total of *2t+1* inputs. Since the set of inputs to the XOR gates are orthogonal, the OLS code will provide the correct output as long as the number of errors is less than half the number of inputs to each voter, i.e., *t* or less. Note that OLS coding does not need to generate a syndrome, but can "correct" errors directly from majority voting.

As an example, a single error correcting OLS code for 16 data bits is shown below. For 16 data bits, *m* = 4. Also, since this is a single error correcting code, *t* = 1. Therefore the total number of check bits will be *2*t*m* = 8. The H-matrix for the resulting (24, 16) code is shown below.

$$H = \begin{bmatrix} d_0 & d_1 & d_2 & d_3 & d_4 & d_5 & d_6 & d_7 & d_8 & d_9 & d_{10} & d_{11} & d_{12} & d_{13} & d_{14} & d_{15} & c_0 & c_1 & c_2 & c_3 & c_4 & c_5 & c_6 & c_7 \\ 1 & 1 & 1 & 1 & & & & & & & & & & & & & 1 & & & & & & & \\ & & & & 1 & 1 & 1 & 1 & & & & & & & & & & 1 & & & & & & \\ & & & & & & & & 1 & 1 & 1 & 1 & & & & & & & 1 & & & & & \\ & & & & & & & & & & & & 1 & 1 & 1 & 1 & & & & 1 & & & & \\ 1 & & & & 1 & & & & 1 & & & & 1 & & & & & & & & 1 & & & \\ & 1 & & & & 1 & & & & 1 & & & & 1 & & & & & & & & 1 & & \\ & & 1 & & & & 1 & & & & 1 & & & & 1 & & & & & & & & 1 & \\ & & & 1 & & & & 1 & & & & 1 & & & & 1 & & & & & & & & 1 \end{bmatrix}$$

As can be seen from the above matrix, each data bit, from d_0 to d_{15} can be reconstructed from two independent parity equations. For example, bit d_0 can be regenerated by XORing, d_1, d_2, d_3, c_0 and also d_4, d_8, d_{12}, c_4. These two independent equations along with bit d_0 itself would be the three inputs to the voter. Using such a scheme bit d_0 can be correctly decoded in the presence of one error. This shows the above H-matrix can tolerate a single error anywhere in the code.

4. Proposed Scheme

The proposed scheme is based on the fact that for decoding OLS codes, as long as each bit has requisite number of inputs feeding into the majority voter, multiple-bit errors can be corrected using a much smaller code. Say, for example bits d_x, d_{x+1}, d_{x+2} are in error. If each row of the OLS matrix contains at most only one of d_x, d_{x+1}, d_{x+2} and each of these bits occur in at least $2t$ different, orthogonal rows of the OLS matrix, then the OLS code is capable of handling any t error pattern and also a burst error pattern on the bits d_x, d_{x+1}, d_{x+2}. Moreover since by definition all rows of an OLS matrix are mutually orthogonal, each particular conflict can occur only once in the code. This helps limit the number of conflicts that needs to be resolved. To summarize, in this example, if the parity relations are chosen carefully then $2t + k$ bits should be sufficient to detect all t-bit error patterns and any burst of length at most three. So, for a OLS code to correct all burst error patterns of length b, no parity relation should be comprised of bits adjacent by a distance b or less.

As part of our scheme we try to resolve all "conflicts" in a given OLS code and convert it into one capable of correcting all burst errors of length b. "Conflicts" here are defined as any row in the original OLS matrix where any combination of b adjacent bits appear together. The OLS matrix cannot correct all burst errors as long as even one such row is present. The problem has been modeled as a graph coloring problem. All the bits which cause conflicts are modeled as graph nodes. Once the set of nodes have been determined, the next step is to formulate the constraints that would dictate which bits can be grouped together. So, here we have a problem where a set of nodes needs to be grouped into as few sets as possible while adhering to some specific rules. If, now, we think of the set of nodes as a set of vertices in a graph whose edges are the specific rules binding their grouping, the problem then almost perfectly lends itself as a graph coloring problem.

Graph coloring refers to a problem, where we seek to assign a color to each node of an undirected graph G so that if (u, v) is an edge, then u and v are assigned different colors; and the goal is to do this while using a small set of colors. More formally, a k-coloring of G is a function $f: V \rightarrow \{1, 2, \ldots, k\}$ so that for every edge (u, v), $f(u) \neq f(v)$. For the burst-error problem under consideration, the number of colors required to color the graph is equal to the number of extra check bits required. There is however, one key difference between the traditional graph coloring problem and our problem at hand. While in a conventional graph coloring problem, one would try to color all nodes of the graph using a minimum set of colors, all we need for our purposes is to select any one node from each conflicting pair and move them to a different line. The justification behind this is as follows. Each conflicting pair represents two bits, separated by a distance less than or equal to b. The structure of an OLS matrix is such that for all $b < m$, conflicting nodes will always be in pairs. Now if we choose any one node from each conflicting pair and move them to different lines, following pre-determined rules, we can convert the OLS code into a burst error correcting code. Hence once we single out the entire set of conflicting nodes (the set V of graph vertices) and determine the constraints binding their mutual positioning in a line of the OLS matrix, we trim the set of nodes by picking one from each conflicting pair. Then we have our final graph which we proceed to color in a manner described below with the final goal of using up as few colors as possible.

Since we expect the set of conflicting nodes to be fairly constrained as far as their mutual positioning in a single line goes, especially for smaller data sizes, we choose to represent the graph using an adjacency matrix. Identifying conflicting pair of nodes requires a single pass through each bit of the OLS matrix once. Since an OLS matrix

capable of $2t$ errors has $2tm$ check bits and m^2 data bits, identifying all conflicting pair of nodes requires $O(m^3)$ time. Once all the conflicting pair of nodes has been recognized, the edges are determined following the rules listed below,

i) if any two nodes u, v of the graph appear in a row of the OLS matrix which produced neither u nor v, then (u, v) is an edge of G.

ii) if any two nodes u, v of the graph are separated by a distance less than or equal to b, then (u, v) is an edge of G. This takes $O(n^2)$, n being the number of nodes in the graph [Cormen 01].

Once the graph G has been created, there are two separate problems which need to be solved in order. First we need to trim the graph, choosing one node from each conflicting pair. While choosing any one from the conflicting pair would preserve the functionality of our goal, choosing the node with fewer edges incident upon it helps us in the next step, where we do the coloring. Since two nodes sharing an edge between them needs to be colored differently, fewer edges would allow us to achieve our coloring goals using fewer colors which translate to fewer extra check bits. The argument rings true intuitively as well, since fewer edges mean less constraints and subsequently more freedom in coloring the graph.

Once the set of nodes, V, and the set of edges, E, has been trimmed, we are left with the final step in our problem i.e. to color G. It has been shown that k-coloring, for $k > 2$ is a NP complete problem [Kleinberg 06]. In this paper we try solving the graph coloring problem using an underlying Breadth First Search (BFS) structure. In order to adapt the traditional BFS algorithm for the purposes of graph coloring we needed to use some additional data structures. Each node maintains an array listing what are the forbidden colors for that node. Thereafter the BFS algorithm is applied as follows,

i) start with a source node, s
ii) add s to the processing queue Q
iii) while $Q \neq \emptyset$,
    a) $u \leftarrow$ *dequeue (Q)*
    b) make a single pass through the array listing forbidden colors, selecting the first color, say c_u not listed as forbidden for u
    c) for each node $v \in Adj\ (u)$
        1) set c_u as a forbidden color
        2) *enqueue (v)*
    d) set *u.visited* $\leftarrow$ 1
iv) go through set of nodes, if *u.visited* $\neq$ 1, run steps i)-iii) on it (after traversing the queue Q, if any node has *u.visited* $\neq$ 1, it would mean that node is disconnected from the rest of the graph, hence a separate modified BFS algorithm needs to be run on that node to ensure that all nodes of the graph are covered)

In addition to traversing each node and each edge of the graph as part of our coloring algorithm, one has to go through an array for each node to determine what should be the correct color for that node. The cost for that traversal is of $O(k)$, for an array of size k. Hence, using our chosen adjacency-matrix representation of the graph, the complexity of the algorithm described above is bounded by $O(n^2k)$.

The number of colors used, k, signifies the number of extra check bits required to convert the original OLS code into a code capable of correcting all burst errors of length b or less. For our experiments, the array in each node keeping track of forbidden colors would be initialized to size k. If at any node, all k colors are designated as forbidden, that would mean the graph cannot be colored using k colors. Subsequent calls to the coloring function would be made with increasing values of k unless a valid solution was obtained. The next section lists some of the experimental results we obtained as proof of concept for our proposed scheme.

5. Results

Table 1 shows experimental results where OLS codes that were originally double error correcting were converted to double error correcting and 3-bit burst error correcting code. Experiments were performed for different sizes of m. Although a double error correcting OLS code would include the sub-matrices M_1, M_2, M_3 and M_4, the structure of M_1 is such that there would be too many conflicts due to bit adjacency. Hence without any loss of generalization, a

double error correcting OLS code was formed out of sub-matrices M_2, M_3, M_4 and M_5. This ensured a minimal number of conflicts and subsequently a better solution.

As shown in Table 1, the overhead of extra check-bits diminish with increasing size of code. This can be explained by virtue of the fact that for a larger m there is more freedom in the placement of bits giving rise to fewer conflicts.

Table 1. Check-bit overhead for 3-bit burst-error protection and Double Error Correcting OLS code

m	Original check-bits	Data Bits	Extra added check-bits	Percentage Overhead
4	16	16	4	25%
8	32	64	4	12.50%
16	64	256	3	4.69%

In Table 2, we show how attempting to build a code capable of handling longer burst errors affects check bit overhead. As expected we see that a larger number of check bits is necessary for stronger codes. We see from Fig. 1 that the increase in the number of extra check-bits follows a piecewise linear relationship with the length of burst-errors to be corrected.

Table 2. Check-bit overhead for DEC OLS code with m=16 while burst-error length is varied

Burst Error Length	Extra added check-bits	Percentage Overhead
3	3	4.69%
5	3	4.69%
7	4	6.25%
9	5	7.81%
11	5	7.81%

6. Concluding Remarks

In this paper we have presented a scheme for generating one-step decodable burst-error correction codes. Our experimental results show that the check-bit overhead is a decreasing function for increasing code size, which makes it this an attractive solution to counter the worsening problem of multiple-bit errors in memory systems.

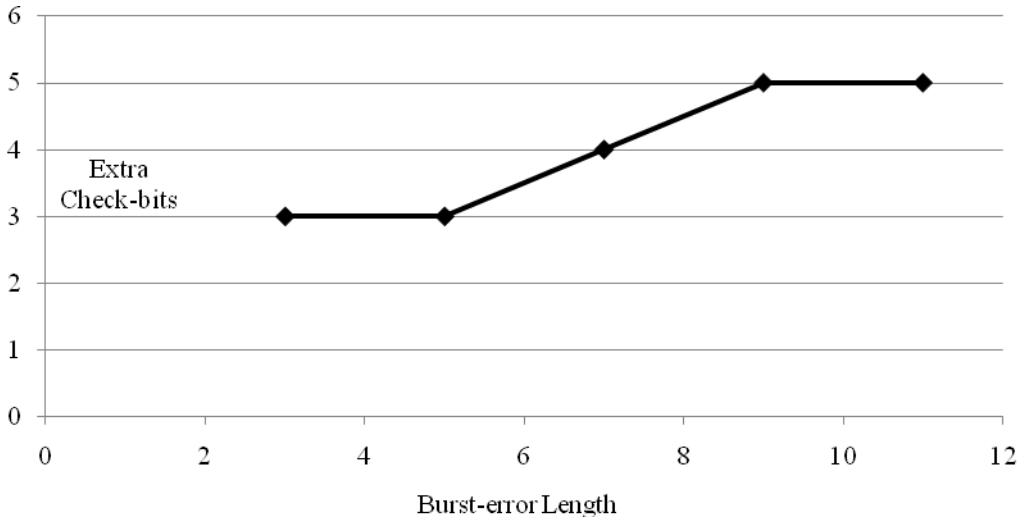

Figure 2. Burst-error length vs check-bit overhead

References

[Chen 08] B. Chen and X. Zhang, "Error Correction for Multi-Level NAND Flash Memory Using Reed-Solomon Codes," *IEEE Workshop on Singal Processing Systems (SiPS)*, pp. 94-99, Oct. 2008.

[Chisti 09] Chishti Zeshan, Alameldeen Alaa R., Wilkerson Chris, Wu Wei and Lu S.-L., "Improving Cache Lifetime Reliability at Ultra-low Voltages", *Proc. of International Symposium on Microarchitecture*, Dec. 2009.

[Cormen 01] Cormen, T., Leiserson, C., Rivest, R., Stein, C., "Introduction to Algorithms", *MIT Press*, 2001

[Dutta 07] Dutta, A., and N.A. Touba, "Multiple-Bit Upset Tolerant Memory Using a Selective Cycle Avoidance Based SEC-DED-DEAC Code," *Proc. of VLSI Test Symopsium*, pp. 349-354, 2007.

[Hsiao 70] Hsiao, M. Y., "A Class of Optimal Minimum Odd-weight-column SEC-DED codes", *IBM Journal of Research and Development*, Vol. 14, pp. 395-401, 1970.

[Kawakami 04] Kawakami, Y., et al., "Investigation of Soft Error Rate Including Multi-Bit Upsets in Advanced SRAM Using Neutron Irradiation Test and 3D Mixed-mode Device Simulation", *Proc. of IEEE Int'l Electronic Device Meeting*, pp. 945-948, Dec. 2004.

[Kim 98] Kim, I., Y. Zorian, G. Komoriya, H. Pham, F.P. Higgins, and J.L. Lewandowski, "Built In Self Repair for Embedded High Density SRAM," *Proc. of International Test Conference*, pp. 1112-1119, 1998.

[Kim 07] Jangwoo Kim, Hardavellas, N., Ken Mai, Falsafi, B., Hoe, J.C, "Multi-bit Error Tolerant Caches Using Two-Dimensional Error Coding", *Proc. of International Symposium on Micro-architectures*, December 2007.

[Kim 10] Kim, C., Rhee, S., Kim, J., Jee, Y., "Product Reed-Solomon Codes for Implementing NAND Flash Controller on FPGA Chip", *Second International Conference on Computer Engineering and Application*, pp 281-285, 2010

[Kleinberg 06] Kleinberg, J., Tardos, E., "Algortihm Design", *Pearson/Addison-Wesley*, pp/ 485-490, 2006.

[Lin 83] Lin, S. Costello, D., *.Error Control Coding: Fundamentals and Applications*. Prentice Hall, 1983

[Makihara 00] Makihara, A., et al., "Analysis of Single-Ion Multiple-Bit Upset in High-Density DRAMS", *IEEE Trans. on Nuclear Science*, Vol. 47, No. 6, Dec. 2000.

[Micheloni 06] R. Micheloni, R. Ravasio, A. Marelli, E. Alice, V. Altieri, A. Bovino, L. Crippa1, E. Di Martino, L. 'Onofrio, A. Gambardella, E. Grillea, G. Guerra, D. Kim, C. Missiroli, I. Motta, A. Prisco, G. Ragone, M. Romano, M. Sangalli, P. Sauro, M. Scotti, S. Won, "A 4Gb 2b/cell NAND Flash Memory with Embedded 5b BCH ECC for 36MB/s System Read Throughput", *IEEE Inter. Solid-State Circuits Conf*, pp. 497-506, February 2006.

[Maiz 03] Maiz, J., Hareland, S., Zhang, K., Armstrong, P., "Characterization of multi-bit soft error events in advanced SRAMs", *Proc. of IEEE International Electron Devices Meeting*, pp. 21.4.1-21.4.4.

[Satoh 00] Satoh, S., Y. Tosaka, S.A. Wender, "Geometric Effect of Multiple-bit Soft Errors Induced by Cosmic-ray Neutrons on DRAMs", *Proc. of IEEE Int'l Electronic Device Meeting*, pp. 310-312, Jun. 2000.

[Seifert 08] Seifert, N., Gill, B., Foley, K., Relangi, P., "Multi-cell upset probabilities of 45nm high-k + metal gate SRAM devices in terrestrial and space environments", *IEEE International Reliability Physics Symposium*, pp. 181-186, 2008

[Su 05] Su, Ching-Lung, Yeh, Yi-Ting, Wu, Cheng-Wen, "An Integrated ECC and Redundancy Repair Scheme for Memory Reliability Enhancement," *Proc. of Defect and Fault Tolerance in VLSI Systems*, pp. 81-89, 2005

[Stapper 92] Stapper, Charles H., Hsing-san Lee, "Synergistic Fault-Tolerance for Memory Chips", *Proc. of IEEE Transactions on Computers*, Vol. 41, No. 9, pp 1078-1087, Sep. 1992.