IOSR Journal of Electronics and Communication Engineering (IOSR-JECE)
e-ISSN: 2278-2834, p- ISSN: 2278-8735.
PP 31-39
www.iosrjournals.org

Multiple Upset Tolerant Memory Using Modified Hamming Code

Babitha Antony, Divya S
[1]*(PG Student, Department of ECE, Sree Narayana Gurukulam College of Engineering, Ernakulam, India)*
[2]*(Assistant Professor, Department of ECE, Sree Narayana Gurukulam College of Engineering, India)*

Abstract: *Error correction codes are used in semiconductor memories to protect information against soft errors. Soft errors are a major concern for memory reliability especially for memories that are used in space applications. In such cases, to protect the data from soft errors, simple error correction codes are preferred due to their simple encoding/decoding logic, low redundancy and low encoding/decoding latency. Hamming codes are attractive as they are simple to construct for any word length and the encoding/decoding can be done with low delay. But they only allow single error correction or double error detection, so a multiple error can lead to a wrong decoding. Nowadays multiple errors are becoming more frequent as integration scale increases. Multiple errors occurs mainly to adjacent bits. In this paper modified Hamming code that can correct single errors, double-adjacent errors, triple-adjacent errors and double-almost-adjacent errors is presented. The double and triple adjacent errors are precisely the types of errors that an MCU would likely cause, and therefore, the modified Hamming code will be useful to provide MCU tolerant memories.*

Keywords *- Adjacent errors, Check matrix, Hamming code, Multiple cell upsets, Syndromes*

I. Introduction

The theory of error detecting and correcting codes is that branch of engineering and mathematics which deals with the reliable transmission and storage of data. Information media are not 100% reliable in practice, in the sense that any radiations or noise (any form of interference) frequently causes data to be distorted. To deal with this undesirable but inevitable situation, some form of redundancy is incorporated in the original data.With this redundancy, even if errors are introduced (up to some tolerance level), the original information can be recovered, or at least the presence of errors can be detected.

For memories that are used in radiation environments especially those that are used in space applications the main cause of error is the radiation induced soft error [1]. The radiation induced soft error occurs when a radiation particle hits the device and changes the logic value. Error Correction Codes (ECCs) are used to prevent soft errors from causing data corruption in memories and registers [2], [3]. The codes used range from simple codes such as Hamming codes [4] to more powerful and complex codes like Bose–Chaudhuri–Hocquenghem (BCH) codes, matrix codes and Euclidean Geometry (EG) codes [5], [6], [7]. In all cases, the data is encoded when it is written into the memory and decoded when it is read. Therefore, the encoding and decoding latency directly impact the memory access time. To minimize this effect ECCs for which decoding is simple are used in most cases. One example of codes for which decoding can be done with low delay is Single Error Correction (SEC) codes. Hamming codes are Single error correction codes. Hamming codes are attractive as they are simple to construct for any word length and the encoding and decoding can be done with low delay.

.SEC codes have a minimum distance of three and therefore a double error can be mistaken for a single error and erroneously corrected. To avoid this issue Single Error Correction Double Error Detection (SEC-DED) codes are preferred in memory applications [2], these codes have a minimum distance of four. Hamming codes can be extended with a parity bit covering all bits to implement a SEC-DED code [4]. Therefore, they are suitable for memory applications and also to protect registers in digital circuits.

A SEC-DED code [4] is capable of correcting one error and detecting all possible double errors. It is commonly used in memories and caches, but cannot correct more than a 1-bit error in a word. As technology scales, it is more likely that a radiation particle upsets more than one memory cell or register causing multiple errors [9]. This is known as a Multiple Cell Upset [10]. Recent studies characterizing different bit errors arising from an SEU suggest that 1–5% of the SEUs can cause multiple bit upsets (MBUs) [8]. The cells affected by the MCU are physically close and in many cases adjacent [11]. This is because errors are created along the path that the particle traverses. MCUs can therefore cause multiple errors on a given word causing a failure even when a SEC-DED code is used.

In order to correct the most commonly occurring MBUs, this paper proposes a low cost ECC methodology to correct single errors, double adjacent bit errors, triple adjacent errors and double almost

adjacent errors. The proposed methodology introduces small area, power and delay overheads.

The rest of the paper is organized as follows section 2 presents related work. In section 3 an overview of Hamming codes is presented. Section 4 describes the proposed code and in Section 5 the simulation results are presented. Section 6 provides the conclusion of the paper

II. Related Work

A number of approaches for extending the basic SEC-DED Hamming code [4] have been previously proposed. A special class of SEC-DED codes known as Hsiao codes [12] was proposed to improve the speed, cost, and reliability of the decoding logic. The codes constructed in the proposed methodology can be thought of as a special class of Hsiao codes. Another class of SEC-DED codes [13], [14] was proposed to detect any number of errors affecting a single byte. These codes are known as single-error-correcting double-error-detecting single-byte-error-detecting (SEC-DED- SBD) codes. For protecting byte-organized memories, SEC-DED-SBD codes are more suitable than the conventional SEC-DED code.

To provide complete double error correction capability, a double-error-correcting triple-error-detecting (DEC-TED) code may be used at the cost of much larger overhead in terms of both the check bits and more complex hardware to implement the error correction and detection [15], [16], [17].

The Reed-Solomon (RS) code and Bose-Chaudhuri- Hocquenghem (BCH) codes are able to detect and correct multiple bytes of errors with very low overhead in terms of additional check bits required. However, these codes typically work at the block level and are applied to multiple words at a time. The general drawbacks with these methods are latency and speed. Most of these codes require several cycles to correct the first error. Moreover, the encoding and decoding are much more complex and require several table lookups for multiplication in higher order fields.

Another class of multiple error-correcting approaches combines coding with circuit level techniques to sense multiple errors in a memory. In [18] and [19], an asynchronous built in current sensor (BICS) on the vertical power lines of a memory along with a parity bit per memory word is used. A conventional SEC-DED code and the BICS approach are combined in [20] to detect multiple bit upsets affecting the same memory word.

Even though several powerful error correcting codes exist, the SEC-DED code has remained an attractive choice mainly because of its fast and simple encoding/ decoding and low hardware overhead. One of the most commonly used techniques to minimize the probability of multiple bit upsets in a single word is bit interleaving which is a memory layout architecture in which physically adjacent bits are assigned to different logical words. For k-way interleaving, k adjacent failing bits appear as k single bit errors in k different logical words rather than as a k-bit error in a single logical word. A simple SEC-DED code can be used along with bit interleaving to help protect from multiple bit upsets. However, there can be some limitations/drawbacks for bit interleaving. In some cases, it may negatively impact floor planning, access time, and/or power consumption. The proposed code based on [21] introduces very little overheads in terms of area, power and delay and can be used instead of or in addition to bit interleaving to provide greater flexibility for optimizing a memory design.

III. Hamming Codes

Hamming codes are linear block error-correcting codes that were proposed by R.W. Hamming [4]. They provide single error correction or double error detection. For any positive integer $m \geq 3$ they have the following parameters :

$$n = 2^m - 1; \; k = n - m; \; d_{min} = 3 \qquad (1)$$

where n is the block size, m the parity check bits, k the number of information bits and dmin the minimum distance of the code. The parity bits are organized in a special way so different incorrect bits produce different error results when decoding. Some possible values of the parameters are illustrated in Table I. For memory applications, the number of information bits k is commonly a power of two and Hamming codes are shortened to fit that word length as illustrated in Table II.

TABLE I
Hamming Codes Parameters

k	n
4	7
11	15
26	31
57	63
120	127
247	255
503	511

TABLE II
Shortened Hamming Codes Parameters

K	n
8	12
16	21
32	38
64	71
128	136
256	265

Hamming codes are linear codes and, consequently, can be generated and decoded using the generator and parity-check matrices, respectively. These matrices have a canonical form, but also equivalent non-systematic code matrices can be obtained by column permutations and other row operations.

An algorithm to generate Hamming code words from information bits is as follows:

a) positions are numbered from 1 to n;
b) positions are written in their binary form (1, 10, 11, etc.);
c) bits in positions 2r are parity bits for those other positions where the binary form of those positions has the bit r + 1 set to one.

For instance, in the Hamming code (7,4) with n = 7, k = 4 and m = 3, positions c1, c2 and c4 are parity bits (p1, p2 and p3) and the information bits (d1, d2, d3 and d4) are placed in order in the rest of positions as shown in Table III.

TABLE III
Generation algorithm for Hamming Code (7,4)

Position	c1	c2	c3	c4	c5	c6	c7
Binary	001	010	011	100	101	110	111
Content	p1	p2	d1	p3	d2	d3	d4
c1 (p1)	--		011		101		111
c2 (p2)		--	011			110	111
c3 (p3)				--	101	110	111

Parity bits are calculated as follows:

c1 = c3 ^ c5 ^ c7 or p1 = d1 ^ d2 ^ d4

c2 = c3 ^ c6 ^ c7 or p2 = d1 ^ d3 ^ d4

c4 = c5 ^ c6 ^ c7 or p3 = d2 ^ d3 ^ d4

As an example, with this scheme the code word for data bits (1 0 1 0) is (1 0 1 1 0 1 0). This approach can be represented as a non-canonical generator matrix where rows are relocated to place the bits in the appropriate

position.

In order to check the code word, the parity bits can be recalculated again from the information bits and compared to the original set of parity bits. If they match, then no error was introduced (or it is not detected), otherwise, an error is detected and the non-matching parity bits can provide us with the information of the bit that was flipped so that the error can be corrected.

From a linear algebra approach, the parity-check matrix can be used to detect an error. The product of this matrix by the current value of the code word results in a vector called syndrome. If this vector is the null vector, then the current value of the word is an actual code word. In any other case, an error occurred in the code word.

There is a special parity-check matrix where column i contains the binary representation of i. It is called the lexicographic check matrix. The lexicographic matrix for the shortened Hamming code (12, 8) is shown in (2)

$$H_{Lex} = \begin{pmatrix} 0&0&0&0&0&0&0&1&1&1&1&1 \\ 0&0&0&1&1&1&1&0&0&0&0&1 \\ 0&1&1&0&0&1&1&0&0&1&1&0 \\ 1&0&1&0&1&0&1&0&1&0&1&0 \end{pmatrix}$$

(2)

If a single bit error occurs in the code word, the syndrome vector that results from the product of the lexicographic matrix with the error code word gives the binary representation of the position where the error was inserted. Using as an example Hamming code (12, 8), data bits (01010100) are coded as (000010110100). When an error occurs and, for instance, the third bit is changed the code word turns into (001010110100). The product of this vector by the lexicographic check matrix results in the syndrome vector (1100) corresponding to the binary representation of three.

A Hamming code can be used to correct single errors or, alternatively, to detect single and double errors. As the minimum distance between two words is three, it is not possible to distinguish between single and double errors. Coming back to the previous example, if there is a double error in the original word in positions 3 and 4 we get the vector (001110110100). Syndrome in this case is (1110) corresponding to the binary representation of 7. In this case, code word would be corrected into (001110010100) instead of the right word.

An option is to use an extended version of the Hamming code that includes an additional parity bit that covers all the bits in the code word. This solution increases the minimum distance to four and allows performing single error correction and double error detection (SEC-DED) simultaneously. Alternatively, it can be used to detect triple errors. In this case, the extended Hamming code (13, 8) encodes (01010100) into (0000101101000). The double error in positions 3 and 4 turns the code word into (0011101101000). This produces a no zero valid syndrome with no error in the parity bit. Therefore, a double error is detected and no correction is done since single errors have a valid syndrome and error in the parity bit.With a minimum distance of 4, a triple error in the extended Hamming code can be miscorrected. In the previous example, errors in bits 3, 4 and 7 produce a zero syndrome and error in the parity bit. This would be interpreted as a single error in the parity bit and miscorrected.

IV. Proposed Code

The characteristics of a linear block code are completely determined by its H-matrix. The proposed code has the following properties:

1) All single bit errors can be corrected.
2) All adjacent double bit errors can be corrected.
3) All triple adjacent bit errors can be corrected.
4) All double almost adjacent bit errors can be corrected.

The following error correction code analysis is based on [21]. Consider a code with k data bits and c check bits. The code can represent 2^k binary values. The code word contains k+c bit positions that could have a single error, k+c-1 bit positions for double adjacent bit error, k+c-2 bit positions for triple adjacent bit error and k+c-2 bit

positions for double-almost-adjacent bit error respectively. Including the set of correct values, there are (k+c) + (k+c-1) + (k+c-2) + (k+c-2)+1 sets of 2^k binary values that must be represented in the k+c bit code word. This implies

$$2^k (k+c+k+c-1+k+c-2+k+c-2+1) \leq 2^{k+c} \qquad (3)$$

Eliminating the common term from both sides of (3), the following relationship can be derived.

$$k+c-1 \leq 2^{c-2} \qquad (4)$$

where c (the number of check bits) should be minimized to reduce the code length.
Given 16 data bits (d0–d15), the minimal number of check bits should be 7 according to (4). Supposing that check bits are c0–c6 and the 23-bit code word is d0–d15c0–c6, the following equations are used to correct a single error, a double-adjacent error, a triple-adjacent error or a double-almost-adjacent error.

d0 ^ d3 ^ d4 ^ d5 ^ d8 ^ d12 ^ d13 ^ c0 = r0

d1 ^ d4 ^ d7 ^ d8 ^ d11 ^ d13 ^ d14 ^ c1 = r1

d2 ^ d5 ^ d6 ^ d9 ^ d10 ^ d11 ^ d14 ^ c2 = r2

d0 ^ d4 ^ d9 ^ d12 ^ d15 ^ c3 = r3

d1 ^ d5 ^ d8 ^ d10 ^ d11 ^ d12 ^ d14 ^ c4 = r4

d2 ^ d7 ^ d9 ^ d10 ^ d11 ^ d12 ^ d15 ^ c5 = r5

d3 ^ d6 ^ d8 ^ d11 ^ d12 ^ d13 ^ d15 ^ c6 = r6 (5)

where ^ represents XOR and c0–c6 are set to make r0–r6 equal to zero under fault-free conditions. Eq.

(5) can be transformed into a matrix form as shown in (6) , where the leftmost matrix is called check matrix (Hamming matrix) that determines which data bits and check bits are XORed, and the rightmost row vector [r0–r6] is a syndrome that is a zero vector under fault-free conditions and a non-zero vector under faulty conditions.

$$\begin{bmatrix} 10011100100011001000000 \\ 01001001100101100100000 \\ 00100110011100100010000 \\ 10001000010010010001000 \\ 01000100101110100000100 \\ 00100001011110010000010 \\ 00010010010111010000001 \end{bmatrix} \begin{bmatrix} d0 \\ d1 \\ d2 \\ d3 \\ d4 \\ d5 \\ d6 \\ d7 \\ d8 \\ d9 \\ d10 \\ d11 \\ d12 \\ d13 \\ d14 \\ d15 \\ c0 \\ c1 \\ c2 \\ c3 \\ c4 \\ c5 \\ c6 \end{bmatrix} = [r0\,r1\,r2\,r3\,r4\,r5\,r6]$$

(6)

Table IV provides syndromes [r0–r6] that correspond to single errors, double-adjacent errors, triple-adjacent errors or double almost-adjacent errors according to (5) or (6). In Table IV, x represents a single error in bit position x, (x, x+1) represents a double-adjacent error in bit positions x and x+1 , (x, x+2) represents a double-almost-adjacent error in bit positions x and x+2, and (x, x+1, x+2) represents a triple-adjacent error in bit positions x, x+1 and x+2 . As seen from Table IV, a single error in bit position x generates a syndrome that matches the x^{th} column in the check matrix. For example, a single error in bit d0 generates a syndrome [1001000] that matches the first column in the check matrix (see (6)). A single error in bit d1 generates a syndrome [0100100] that matches the second column in the check matrix. A double-adjacent error in bit positions x and x+1 generates a syndrome that matches the XOR of x^{th} and $x+1^{th}$ columns in the check matrix. For example, a double-adjacent error in bits d0 and d1 generates a syndrome [1101100] that matches the XOR of the first and the second columns in the check matrix. As seen from Table IV, a double-almost-adjacent error in bit positions x and x+2 generates a syndrome that matches the XOR of the x^{th} and $x+2^{th}$ columns in the check matrix. For example, a double-almost-adjacent error in bits d0 and d2 generates a syndrome [1011010] that matches the XOR of the first and the third columns in the check matrix. A triple-adjacent error in bit positions x, x+1 and x+2 generates a syndrome that matches the XOR of the x^{th}, $x+1^{th}$ and $x+2^{th}$ columns in the check matrix. For example, a triple-adjacent error in bits d0, d1 and d2 generates a syndrome [1111110] that matches the XOR of the first, the second and the third columns in the check matrix.

As seen from Table IV, the syndromes for single, double and triple errors are unique with respect to each other in order to correct these errors. Therefore, in the check matrix (in (4)), the individual columns, the XOR of double adjacent columns, the XOR of double-almost-adjacent columns, and the XOR of triple adjacent columns are also unique with respect to each other. Additionally, in the check matrix, the 17th to 23rd columns that correspond to check bits c0–c6 (i.e. the syndromes for single errors in check bits) are one-hot codes such as $[1000000]^T$, $[0100000]^T$,…. $[0000001]^T$.

Hence the proposed scheme is performed using an exhaustive search algorithm as follows.

At step 1, given k data bits , the minimal number of check bits c is obtained according to (2). Supposing check bits are $c_0 - c_{c-1}$ and the code word is $d_0 - d_{k-1}$ $c_0 - c_{c-1}$,the check matrix is a matrix with c rows and k+c columns, where the last c columns corresponding to check bits $c_0 - c_{c-1}$ are set to one-hot codes $[1000000]^T$, $[0100000]^T$,…. $[0000001]^T$ respectively.

TABLE IV
Syndromes for determining Single, Double and Triple errors

Faulty Bit X	Syndromes for single, double and triple errors			
	Single	Double		Triple
	X	x, x+1	x, x+2	x, x+1, x+2
d0	1001000	1101100	1011010	1111110
d1	0100100	0110110	1100101	1110111
d2	0010010	1010011	1111010	0111011
d3	1000001	0101001	0010101	1111101
d4	1101000	0111100	1111001	0101101
d5	1010100	1000101	1110110	1100111
d6	0010001	0110011	1110101	1010111
d7	0100010	1000110	0111001	1011101
d8	1100100	1111111	1110010	1101001
d9	0011011	0001101	0101100	0111010
d10	0010110	0100001	1011001	1101110
d11	0110111	1111000	1010110	0011001
d12	1001111	0101110	1111011	0011010
d13	1100001	1010101	1101010	1011110
d14	0110100	0111111	1110100	1111111

d15	0001011	1001011	0101011	1101011
c0	1000000	1100000	1010000	1110000
c1	0100000	0110000	0101000	0111000
c2	0010000	0011000	0010100	0011100
c3	0001000	0001100	0001010	0001110
c4	0000100	0000110	0000101	0000111
c5	0000010	0000011	-	-
c6	0000001	-	-	-

At step 2, the available column value set is initialized with all possible column values except for zero. The c one-hot codes($[1000000]^T$, $[0100000]^T$,…. $[0000001]^T$) are removed from the available column value set because they have been used by the columns for check bits. In the columns for check bits, the XOR of double adjacent columns, the XOR of double almost adjacent columns, and the XOR of triple-adjacent columns are also removed from the available column value set.

At step 3, the remaining empty columns of the check matrix are filled sequentially. An empty column is filled with a valid column value in the available column value set. The valid column value should ensure that no duplicate values exist for individual filled columns (i.e. the syndromes for single errors), the XOR of two adjacent filled columns (i.e. the syndromes for double-adjacent errors), the XOR of two almost-adjacent filled columns (i.e. the syndromes for double-almost-adjacent errors), and the XOR of three adjacent filled columns (i.e. the syndromes for triple-adjacent errors). So the valid column value can ensure that the syndromes for single, double and triple errors are unique with respect to each other and hence these errors are correctable. Then this column value, the XOR of double-adjacent filled columns, the XOR of double-almost-adjacent filled columns and the XOR of triple-adjacent filled columns are all removed from the available column value set. When an empty column cannot find a valid value in the available column value set, a backtracking is performed on neighbouring columns one by one by returning removed values to the available column value set and replacing the filled columns with different valid values. The backtracking may help this empty column find a valid value.

At step 4, if the check matrix is completely filled, the search algorithm may end, or go back to step 3 to continue looking for a better solution by filling columns with different valid values.

V. Fpga Implementation And Simulation Results

The code for the proposed Hamming code was written in Verilog Hardware Description Language and the code was simulated on ModelSim SE 6.3f for a 16 bit data. It was tested for correct functionality by giving various inputs. The proposed code is synthesized on Xilinx 13.3 version and has been implemented on Spartan 3E FPGA kit. The simulation results of the encoder and decoder section of the proposed code is shown in Fig. 1 and Fig. 2. The input to the encoder is a 16 bit data and the output is a 23 bit code word. The 23 bit code word is the input to the decoder.

Fig. 1. Simulation result for encoder of the proposed code

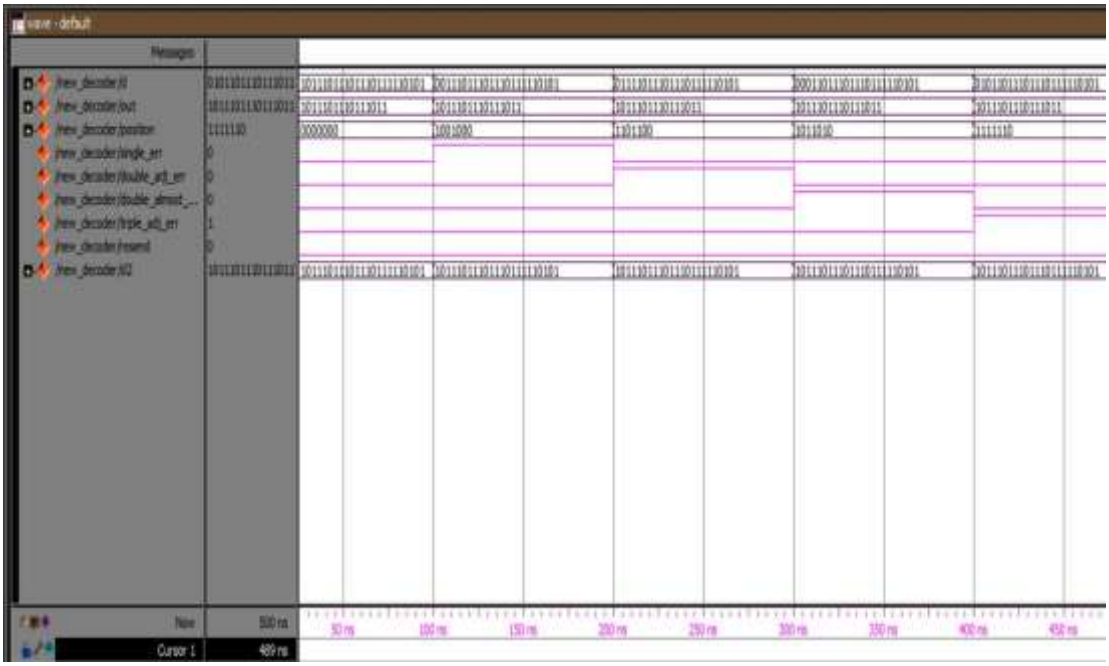

Fig. 2. Simulation result for decoder of the proposed code

VI. Conclusion

The proposed code described in this paper has the ability to correct single error, double adjacent errors, double almost adjacent errors and triple adjacent errors. The proposed scheme will correct upsets for up to 3 bits as long as they are within a span of 3 bits . The main drawback is that it will not correct even a double bit upset if the two bits occur outside that range. Further, with this scheme there is no way to detect a non-correctable double bit upset. The proposed methodology is flexible and can be used for the correction of adjacent errors which are the ones usually caused by Multiple Cell Upsets (MCUs) in semiconductor memories.

References

[1] R. C. Baumann, "Radiation-induced soft errors in advanced semiconductor technologies," *IEEE Trans. Device Mater. Rel., vol. 5, no. 3, pp. 301– 316, Sep. 2005.*

[2] C. L. Chen and M. Y. Hsiao, "Error-correcting codes for semiconductor memory applications: A state-of-the-art review," *IBM J. Res. Develop., vol. 28, no. 2, pp. 124–134, Mar. 1984.*

[3] R. Leveugle, "Optimized state assignment of single fault tolerant FSMs based on SEC codes," *in Proc. 30th Conf. Des. Autom., Jun. 1993, pp. 14–18.*

[4] R. W. Hamming, "Error detecting and error correcting codes," *Bell Syst. Tech. J., vol. 29, no. 2, pp. 147–160, Apr. 1950.*

[5] Anjum Asma and Gihan Nagib,'Energy Efficient Routing Algorithms for Mobile Ad Hoc Networks–A Survey', *International Journal of Emerging Trends & Technology in computer Science, Vol.3, Issue 1, pp. 218-223, 2012.*

[6] P. Ankolekar, S. Rosner, R. Isaac, and J. Bredow, "Multi-bit error correction methods for latency-constrained flashmemory systems," *IEEE Trans. Device Mater. Rel., vol. 10, no. 1, pp. 33–39, Mar. 2010.*

[7] Costas Argyrides, Dhiraj K. Pradhan and Taskin Kocak, "Matrix Codes for Reliable and Cost Efficient Memory Chips", *IEEE Transactions On Very Large Scale Integration (Vlsi) Systems, Vol. 19, No. 3, pp.420-428, March 2011.*

[8] H. Naeimi and A. DeHon, "Fault secure encoder and decoder for nanoMemory applications," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst., vol. 17, no. 4, pp. 473–486, Apr. 2009.*

[9] Maiz, J., S. Hareland, K. Zhang, and P. Armstrong, "Characterization of Multibit Soft Error Events in Advanced SRAMs", *Proc. of IEEE Int'l Electronic Device Meeting, pp. 519-522, Dec. 2003.*

[10] E. Ibe, H. Taniguchi, Y. Yahagi, K. Shimbo, and T. Toba, "Impact of scaling on neutron-induced soft error in SRAMs from a 250 nm to a 22 nm design rule," *IEEE Trans. Electron Devices, vol. 57, no. 7, pp. 1527–1538, Jul. 2010.*

[11] R. K. Lawrence and A. T. Kelly, "Single event effect induced multiple-cell upsets in a commercial 90 nm CMOS digital technology," *IEEE Trans. Nucl. Sci., vol. 55, no. 6, pp. 3367–3374, Dec. 2008.*

[12] S. Satoh, Y. Tosaka, and S. A. Wender, "Geometric effect of multiple-bit soft errors induced by cosmic ray neutrons on DRAM's," *IEEE Electron Device Lett., vol. 21, no. 6, pp. 310–312, Jun. 2000.*

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

[14] Reddy, S.M., "A Class of Linear Codes for Error Control in Byte-per-Package Organized Memory Systems", *IEEE Trans. On Computers, Vol. C-27, pp. 455-458, May. 1978.*

[15] Chen, C. L.," Error Correcting Codes with Byte Error Detection Capability", *IEEE Trans. On Computers, Vol. 32, pp. 615-621, May 1983.*
[16] Lin, S., and D. J. Costello, Jr., Error Control Coding: Fundamentals and Applications, Prentice-Hall, 1983.
[17] Berlekamp, E. R., Algebraic Coding Theory, McGraw-Hill, New York, 1968.
[18] Lala,P.K.,"An Adaptive Double Error Correction Scheme for Semiconductor Memory Systems,"*Digital Processes,Vol.4, pp 237-243, 1978.*
[19] Vargas, F. L., and M. Nicolaidis," SEU-Tolerant SRAM Design Based On Current Monitoring", *Proc. Int. Symposium on Fault Tolerant Computing, pp. 106-115, June 1994.*
[20] Calin, Th., F. L. Vargas, and M. Nicolaidis,"Upset Tolerant CMOS Using Current Monitoring: Prototype and Test Experiments",*Proc. Int. Test Conference, pp. 45-53, 1995.*
[21] Gill, B., M. Nicolaidis, and C. Papachristou," Radiation Induced Single-Word Multiple-bit Upsets Correction in SRAM" *Proc. of Int. Online Test Symposium, pp. 266-271, Jul. 2005.*
[22] X. She, N. Li, and D. W. Jensen, "SEU tolerant memory using error correction code," *IEEE Trans. Nucl. Sci., vol. 59, no. 1, pp. 205–210, Feb. 2012.*