International Journal of Applied Information Systems (IJAIS) – ISSN : 2249-0868
Foundation of Computer Science FCS, New York, USA
Volume 5– No.3, February 2013 – www.ijais.org

Multi-Bit Upset Deduction/Correction for Memory Applications

X. Jushwanth Xavier
M.E Student,
Applied Electronics,
Velammal Engineering College
Chennai, India.

Lakshmi Kantham
Assistant Professor,
Dept. of Electronics and communication,
Velammal Engineering College
Chennai, India.

ABSTRACT

In electronics memories are the widely used elements. As the transistor size shrinks multiple-bit upset (MCUs) are increasing due to radiation effects in memories. This affects the reliability of memories. Interleaving and built-in current sensors (BICS) have been success in the case of single event upset (SEC). The process is taken one step further by proposing specific error correction codes to protect memories against multiple-bit upsets and to improve yield have been proposed. The method is evaluated using fault injection experiments. The results are compared with known techniques such as Hamming codes. The proposed codes provide a better performance compared to that of the hamming codes in terms of Single Event Upset. In the case of the Multi Bit Upset it provides better coverage in error deduction and correction schemes.

General Terms
Error correction, VLSI

Keywords
Multi-bit error correction, Single event upset, hamming codes.

1. Introduction

CMOS scaling process provides high-density, low cost, low power, high-speed integrated circuits with a small noise margin. Due to this features susceptible temporary faults will be increased [1]. In very deep sub-micron technologies due to atmospheric neutrons and alpha particles the device's field-level reliability is severely impacted by single-event upset (SEU) and multi-bit event upset (MBU). When these particles hit the devices silicon bulk, they produce minority carrier, which produces voltage change at the nodes. Due to this not only memories but logic are also affected.

In combinational circuits soft error rate has drawn a major attention as the numbers of fault in the devices have increased significantly. Effective solutions in protecting memories are also provided in [2]. Circuit latch up at output due to neutron effect have become second point, not many techniques cope with this problem. Transient faults in space applications are potential consequences for the space craft that includes loss of information, functional failure of the craft [3]. Although SEUs are major concern, multiple-bit upset (MBU) has become important problem in the design processes of the memories. The probabilities of multiple errors due to technology shrinkage have been already discussed in [4] and [5]. As the size of the memories increases the probability of having multiple bits upset increase since large number of memory cells has been discussed in [6] and [7].

Unfortunately packing and shielding cannot be effective against SEUs and MBUs since the neutrons can easily penetrate through the shield packages [5], [8].

Common approach is to use memory interleaving, in which the cell that belonging to the same logical word are placed at different positions during the design. Since the MBU errors are caused to the cells that are closer discussed in [9]. However this method cannot be used in larger memories because if the high accesses time, power consumption and floor plan discussed in [10].

An alternative to protect memories is by using built-in current sensors (BICS) that can deduct errors by detecting changes in the current as in [11], [12]. The protection can be optimized with the error correction codes (ECC) to cope up with MBUs. This is the objective of this paper proposing a new ECC to overcome MBUs.

2. Error correcting codes

Error correcting codes are widely used in protecting memories against the soft errors that are occurring due to the changes in the environment and the operating point of the devices. Hamming codes are widely used to protect memories against SEU because of the reduced area and performance. The hamming code implementation is composed by a combinational encoder block, this includes extra latches or flip-flops since the parity bits are included and another combinational decoder block. The encoder block calculates parity and it can be implemented by a set of 2-input XOR gates. The decoder block is more complex because it needs not only to detect the fault, but also correct it. The decoder block can also be composed of a set of 2-input XOR gates and some INVERTER gates. In order to improve the efficiency of the error correction, Triple modular redundancy (TMR) is used. But TMR uses poling methods that increase the area along with hamming code can correct only one error. Hence BICS are used along with ECC with a trade-off with area. Different methods are proposed that depends on redundancy that gradually increases the area.

Error deduction and correction in memories should be simple since accesses time is a major criteria. Due to high bandwidth used in memories in SOC applications the efficiency of repairing the memories decreases and redundant methods cannot be used. Examples of such applications are presented in [13] [14]. In order to cope up with the errors during the

manufacturing processes certain times half of the device is used this is done by setting the MSB of the memory to be 0 or 1. Divide by half technique to cope with this problem has been proposed [15], to improve the efficiency of the memories novel techniques are required in the error correction. The technique that is used here gradually can correct more number of the errors with improving the overall system reliability.

3. Error Detection/Correction Scheme

In this detection/correction scheme the message bits are arranged in a matrix format. This is a combination of the parity codes and the hamming codes. The *n*-bit code word is divided into n_l sub-words of width n_2 (i.e.$k = n_1 * n_2$). A (n_1, n_2) matrix is formed where n_1 and n_2 represents the numbers of rows and columns, respectively. For each of the n_1 rows, the check bits are added for single error correction/double error detection. Another n_2 bits are added as vertical parity bits.

M_0	M_1	M_2	M_3	M_4	M_5	M_6	M_7	C_0	C_1	C_2	C_3	C_4
M_8	M_9	M_{10}	M_{11}	M_{12}	M_{13}	M_{14}	M_{15}	C_5	C_6	C_7	C_8	C_9
M_{16}	M_{17}	M_{18}	M_{19}	M_{20}	M_{21}	M_{22}	M_{23}	C_{10}	C_{11}	C_{12}	C_{13}	C_{14}
M_{24}	M_{25}	M_{26}	M_{27}	M_{28}	M_{29}	M_{30}	M_{31}	C_{15}	C_{16}	C_{17}	C_{18}	C_{19}
P_0	P_1	P_2	P_3	P_4	P_5	P_6	P_7					

Fig 1: 32-bit logical organization

The technique is explained by considering a 32-bit word length memory, which is divided into a matrix format as shown in Fig. 1, where $n_1 = 4$ and $n_2 = 8$, M_0 through M_{31} are the data bits, C_0 through C_{19} are the horizontal check bits, $p_0 - p_7$ are the vertical parity bits. Hamming codes are applied to each row. For an 8-bit data, 5 Hamming check bits are required. Hence 5 check bits are added at the end of the 8 bits.

As mentioned above the horizontal bits $P_0 - P_7$ are calculated using the ordinary parity generators. While the entire right side bits $C_0 - C_{19}$ are calculated as follows:

$$C_0 = M_0 \oplus M_1 \oplus M_3 \oplus M_4 \oplus M_6 \quad (1)$$

$$C_1 = M_0 \oplus M_2 \oplus M_3 \oplus M_5 \oplus M_6 \quad (2)$$

$$C_2 = M_1 \oplus M_2 \oplus M_3 \oplus M_7 \quad (3)$$

$$C_3 = M_4 \oplus M_5 \oplus M_6 \oplus M_7 \quad (4)$$

$$C_4 = M_0 \oplus M_1 \oplus M_2 \oplus M_3 \oplus M_4 \oplus M_5 \oplus M_6 \oplus M_7 \quad (5)$$

Accordingly, we calculate all check bits for all rows using $C_{New} = C_{j+(cb*o)}$ and $M_{new} = M_{i+(n2*o)}$, where cb is the position of check bit in the row, o is the row number where is the corresponding check bit's position in the first row and i is the corresponding data bit's position in this first row. For the parity row we use the following formula:

$$Pi = M_i \oplus M_{i+8} \oplus M_{i+16} \oplus M_{i+24} \quad (6)$$

Where is column number from 0 to 7 for eight parity bits.

A Hamming decoder is used to decode each row. This process is carried out in two steps. First, the horizontal check bits are calculated using the saved data bits and compared with the saved horizontal check bits. This procedure is called syndrome bit generation and C_1 is called syndrome bit of check bit C_1. Second, using syndrome bits S_i, the single error detection (SED)/double error detection (DED)/no error (NE) signals are generated for each row. If DED is activated (double error is detected in a row), we use the vertical syndrome bits SP_i and the saved value of the bit we can correct any single or double erroneous bits in each row using (7)

$$M_{icorrect} = (M_{ierr} \oplus 0) \oplus (DED_j * SP_n) \quad (7)$$

where M_{ierr} is the erroneous bit, O represents the decoder output corresponding to the erroneous bit i, DED_j is the DED signal of row j and SP_n the syndrome parity of the corresponding parity of the bit, e.g., for M_{10} , we have SP_2.

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

Fig 2: proposed 32bit code

It is important to mention that if more than two errors are present in the code word, this technique can correct errors in any row assuming that no error in the same column . If only two errors occur, they these can be corrected without any restriction. Algorithm 1 shows the procedure of detection and correction in the proposed method which is applied on a code word M, where C_i' and P_i' are the check bits and the parity bits that are calculated using the saved data bits in the memory. These are then compared with saved memory check bits and parity bits to calculate the syndrome bits SC and SP.

Algorithm 1: code verification algorithm (M: data)

1: *Read the saved data bits of M*

2: *Generate check bits using saved data bits* $(C'_0 - C'_{19})$

3: *Generate syndrome bits of check bits* $(SC_1 - SC_{19})$

4: *Generate parity bits using saved data bits* $(P'_0 - P'_7)$

5: *Generate syndrome bits of parity bits* $(SP_1 - SP_7)$

6: *Correct every saved bit if it is erroneous using*(7)

7: *Output the corrected word*

Let us give an example of the technique. Suppose the code word "11011011 10110001 00110101 10111000" is saved to the memory. Check bits $C_0 - C_{19}$ will be equal to "11110 01110 11100 11010" and according to parity equations, parity bits $P_0 - P_7$ will be equal to "11100111". The physical layout of this code word with check and parity bits is shown in Fig. 2. Suppose that while reading the code word from the memoryM_0, M_1 and M_2 are erroneous. M_0 Changed from 1 to 0, M_1 from 1 to 0 and value M_2 is changed from0 *to* 1. Using the decoding algorithm one can easily correct the two erroneous bits. This procedure is shown in Example 1.With

this technique we can correct any kind of single or double errors in each row.

Example 1: The procedure of error detection/correction

1. $Read\ the\ saved\ data\ bits\ =$ **00**111011 10110001 00110101 10111000
2. $Calculate\ check\ bits\ using\ saved\ data\ bits:$

   $C'_0 - C'_{19}$ = "11111 01110 11100 11010"
3. $Generate\ syndrome\ bits\ for\ check\ bits:$

   $(SC_0 - SC_{19})$ = "00001000000000000000"
4. $Generate\ parity\ bits\ using\ saved\ data\ bits:$

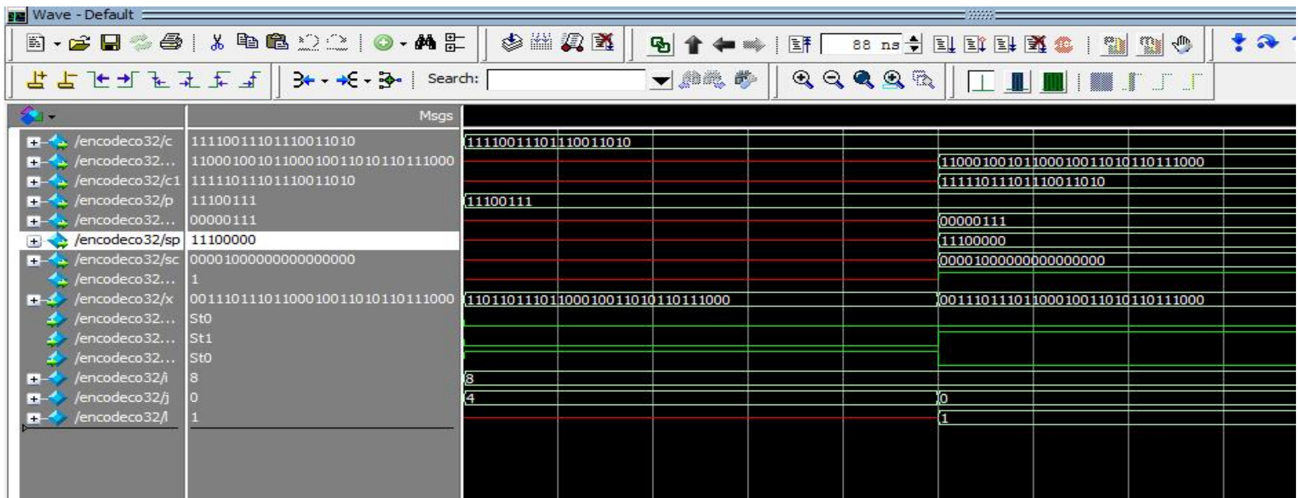
   $(P'_0 - P'_7)$ = "0000 0111"
5. $Generate\ syndrome\ bits\ of\ parity\ bits:$

   $(SP_0 - SP_7)$ = "1110 0000"
6. $Correct\ every\ saved\ bit\ if\ it\ is\ erroneous\ using$ (7)
7. $M_{0correct} = (M0err \oplus 0) \oplus (DED0 * SP0)$
8. $M_{0correct} = (0 \oplus 0) \oplus (1 * 1)$
9. $M_{0correct} = (0) \oplus (1) = 1\ Initial\ Value$
10. $M_{1correct} = (M_{1err} \oplus 0) \oplus (DED_0 * SP1)$
11. $M_{1correct} = (0 \oplus 0) \oplus (1 * 1)$
12. $M_{1correct} = (0) \oplus (1) = 1\ Initial\ Value$
13. $M_{2correct} = (M_{2err} \oplus 0) \oplus (DED_0 * SP_1)$
14. $M_{2correct} = (1 \oplus 0) \oplus (1 * 1)$
15. $M_{2correct} = (1) \oplus (1) = 1\ Initial\ Value$
16. $Output\ the\ corrected\ word\ =$

    "**110** 11011101100010011010110111000"

The read and write procedure for the memory with error correcting technique can be explained as follows. Word in the modules is segmented into multiple bit segments. Then each n bit segment is encoded to k bit segment of $(k - n)$ check bits. Algorithms 2 and 3 show the procedure for reading/writing words from a memory location or to a memory location, respectively.

Fig 3: Simulated output of fault simulation contains data bits



Algorithm 2: MEMORY READ

1. Read the word which contains the desired bits.
2. Correct for any errors.
3. Route the desired bits on the tree to the root node

Algorithm 3: MEMORY WRITE

1. Read the word which includes the desired bit.
2. Check for errors and correct them (if any)
3. Compare the value of the bit to be written against the value stored in the memory.
4. if bits are different then
5. Re-compute the check bits based on this new value.
6. Write back the data and the newly computed check bits
7. Else
8. Write back the data and the newly computed check bits
9. end if

Hence based on the algorithm the error deduction/correction is carried out

4. Simulation Results

The entire coding is done in verilog HDL and simulated. Fault injection is one of the key methods to estimate the error detection/correction capabilities of the circuits which utilize error detection and correction codes. Using a fault injection method, the coverage of the proposed technique was estimated. A thousand of faults were thrown and results were

analysed. Compared to the previous methods the proposed method was able to deduct and correct up to eight errors in a row with a condition that no errors occur in the same column. Since this technique can correct only one error per row. Figure 3 shows the simulated fault injection method with three faults in data bits 0, 1 and 2 positions. And the result shows a successful error correction by using this method. Figure 4 shows the deduction and the correction coverage per code word of 8bits and found that the proposed technique Proves to be a more efficient method for multi-bit correction methods.

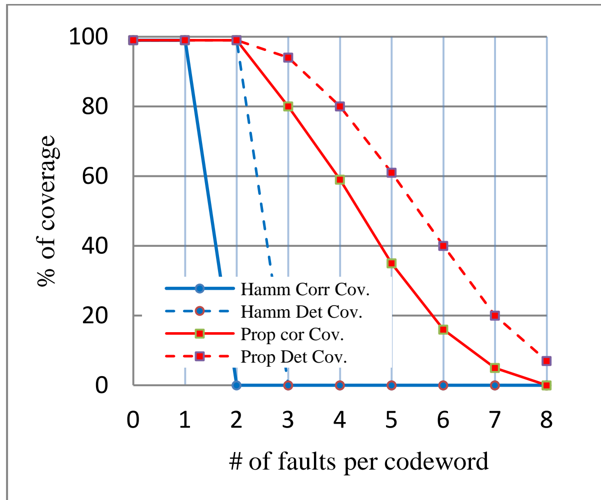


Fig 4: Fault coverage

5. Conclusion

Here a high level error detection and correction method is introduced. The proposed protection code combines Hamming code and Parity code, so that multiple errors can be detected and corrected. The fault-injection based experimental results show that the proposed method provides better Detection and correction coverage than the Hamming codes. A 32 bit encoder and decoder are designed and simulated. By using the fault simulation method faults are forced for multiple bits by using forcing value in modelsim and results are verified. The code is able to deduct SEU/MBU and correct the errors.

6. References

[1] Hareland, S., Maiz, J., Alavi, Mistry, K., Walsta, S., Changhong Dai, "Impact of CMOS process scaling and SOI on the soft error rates of logic processes", VLSI Technology, 2001. Digest of Technical Papers. 2001 Symposium on, 73 - 74

[2] Cardarilli, G., Leandri, A, Marinucci, P., M. Ottavi, Pontarelli, S., M. Re, and Salsano, A., "Design of a fault tolerant solid state mass memory," IEEE Transactions on Reliability., vol. 52, no. 4, pp. 476–491, Dec. 2003.

[3] Ferreyra, P.A., Marques, C.A., Ferreyra, R.T., Gaspar, J.P., "Failure map functions and accelerated mean time to failure tests: New approaches for improving the reliability estimation in systems exposed to single event upsets," *IEEE Trans. Nucl. Sci.*, vol. 52, no. 1, pp. 494–500, Jan. 2005.

[4] Hazucha, P., Svensson, C. "Impact of CMOS technology scaling on the atmospheric neutron soft error rate," *IEEE Trans. Nucl. Sci.*, vol. 47, no. 6, pp. 2586–2594, Dec. 2000.

[5] Karlsson, J., Liden, P., Dahlgren, P., Johansson, R., Gunneflo, U. "Using heavy-ion radiation to validate fault-handling mechanisms," *IEEE Trans. Microelectron.*, vol. 14, pp. 8–23, 1994.

[6] Reed, R.A., Carts, M.A., Marshall, P.W., Marshall, C.J., Musseau, O., McNulty, P.J., Roth, D.R., Buchner, S., Melinger, J., Corbiere, T. "Heavy ion and proton-induced single event multiple upset," *IEEE Trans. Nucl. Sci.*, vol. 44, no. 6, pp. 2224–2229, Dec. 1997.

[7] Seifert, N., Moyer, D., Leland, N., Hokinson, R. "Historical trend in alpha-particle induced soft error rates of the Alpha microprocessor," in *Proc. 39th Annu. IEEE Int. Reliab. Phys. Symp.*, 2001, pp. 259–265.

[8] Satoh, S., Tosaka, Y., Wender, S.A. "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, 2000.

[9] Dutta, A., Touba, N.A. "Multiple bit upset tolerant memory using a selective cycle avoidance based SEC-DED-DAEC code." in *Proc. IEEE VLSI Test Symp. (VTS)*, 2007, pp. 349–354.

[10] Nicolaidis, M., Vargas, F., Courtois, B. "Design of built-in current sensors for concurrent checking in radiation environments," *IEEE Trans. Nucl. Sci.*, vol. 40, no. 6, pp. 1584–1590, Dec. 1993.

[11] Lo, J,. "Analysis of a BICS-only concurrent error detection method," *IEEE Trans. Computers*, vol. 51, no. 3, pp. 241–253, 2002.

[12] Lu, S. K., "Efficient built-in redundancy analysis for embedded memories with 2-D redundancy," *IEEE Trans. Very Large Scale Integr. (VLSI) Systems*, vol. 14, no. 1, pp. 34–42, Jan. 2006.

[13] Shyue-Kung Lu, Shih-Chang Huang, "Built-in self-test and repair (BISTR) techniques for embedded RAMs," in *Proc. Int.Workshop, Memory Technol. Des. Test.*, Aug. 2004, pp. 60–64.

[14] Argyrides, C., Al-Yamani, A., Lisboa, C., Carro, L., Pradhan, D. "Increasing memory yield in future technologies through innovative design," in *Proc. 8th Int. Symp. Quality Electron. Des. (ISQED)*, Mar. 2009, pp. 622–626.

[15] Elmer, B., Tchon, W., Denboer, A., Kohyama, S., Hirabayashi, K., Nojima, I. "Fault tolerant 92160 bit multiphase ccd memory," in *IEEE Int. Conf. Solid-State Circuits. Dig. Techn. Papers*, Feb. 1977, 116–117.