DESIGN OF EFFICIENT MULTI BITS ERROR CORRECTION CODES USING OLS CODES

TALAGALLA TULASI RAM[1], M.VIJAYA RAMARAJU[2], A.BHARGAV[3]

[1] M.TECH (COMMUNICATION SYSTEMS) IN SAGI RAMAKRISHNAM RAJU ENGINEERING COLLEGE, BHIMAVARAM, ANDHRAPRADESH, INDIA.

[2] ASSOCIATE PROFESSOR IN ECE IN SAGI RAMAKRISHNAM RAJU ENGINEERING COLLEGE, BHIMAVARAM, ANDHRAPRADESH, INDIA.

[3] ASSISTANT PROFESSOR IN ECE IN SAGI RAMAKRISHNAM RAJU ENGINEERING COLLEGE, BHIMAVARAM, ANDHRAPRADESH, INDIA.

Abstract : The use of error-correction codes (ECCs) with advanced correction capability is a common system-level strategy to harden the memory against multiple bit upsets (MBUs).Therefore, the construction of ECCs with advanced error correction and low redundancy has become an important problem, especially for adjacent ECCs. Existing codes for mitigating MBUs mainly focus on the correction of up to 3-bit burst errors. As the technology scales and cell interval distance decrease, the number of affected bits can easily extend to more than 3 bit. The previous methods are therefore not enough to satisfy the reliability requirement of the applications in harsh environments. In this paper, a technique to extend 3-bit burst error-correction (BEC) codes with quadruple adjacent error correction (QAEC) is pre-scented. First, the design rules are specified and then a searching algorithm is developed to find the codes that comply with those rules. The Hmatrices of the 3-bit BEC with QAEC obtained are presented. They do not require additional parity check bits compared with a 3-bit BEC code. By applying the new algorithm to previous 3-bit BEC codes, the performance of 3-bit BEC is also remarkably improved. The encoding and decoding procedure of the proposed codes is illustrated with an example. The results show that our codes have moderate total area and delay overhead to achieve the correction ability extension.

Keywords: Error correction codes (ECCs), Orthogonal Latin square (OLS), One step majority logic decodable (OS-MLD), Memory, Parity.

INTRODUCTION

Reliability is a major concern in advanced electronic circuits. To ensure errors do not affect the circuit functionality a number of mitigation techniques can be used. Among them, Error Correction Codes (ECC) are used to protect memories and registers in electronic circuits. The general idea for achieving error correction is to add some redundancy which means to add some extra data to a message, which receiver can use to check uniformity of the delivered message, and to pick up data determined to be corrupt. Error correction scheme may be systematic or it may be non-systematic. In the system of the module non-

systematic code, an encoded is achieved by transformation of the message which has least possibility of number of bits present in the message which is being converted. Another classification is the type of systematic module unique data is sent by the transmitter which is attached by a fixed number of parity data like check bits that obtained from the data bits. The receiver applies the same algorithm when only detection of the error is required to the received data bits which is then compared with its output with the receive check bits if the values does not match, there we conclude that an error has occurred at some point in the process of transmission. Error correcting codes are regularly used in lower-layer communication, as well as for reliable storage in media such as CDs, DVDs, hard disks and RAM. Provision against soft errors that apparent they as the bit-flips in memory is the main motto of error detection and correction. Several techniques are used present to mitigate upsets in memories. For example, the Bose – Chaudhuri– Hocquenghem codes, Reed–Solomon codes, punctured difference set codes, and matrix codes has been used to contact with MCUs in memories. But the above codes mentioned requires more area, power, and delay overheads since the encoding and decoding circuits are more complex in these complicated codes. Reed-Muller code is another protection code that is able to detect and correct additional error besides a Hamming code. But the major drawback of this protection code is the more area it requires and the power penalties. Hamming Codes are mostly used to correct Single Error Upsets (SEU's) in memory due to their ability to correct single errors through reduced area and performance overhead. Although it is brilliant for correction of single errors in a data word, but they cannot correct two bit errors caused by single event upset. An extension of the basic SEC-DED Hamming Code has been proposed to form a special class of codes known as Hsiao Codes to increase the speed, cost and reliability of the decoding logic. One more class of SEC-DED codes known as Single-error-correcting, double error-detecting, Single-byte-error-detecting SEC-DED-SBD codes be proposed to detect any number of errors disturbing a single byte. These codes are additional suitable than the conventional SEC-DED codes for protecting the byteorganized memories. Though they operate through lesser overhead and are good for multiple error detection, they cannot correct multiple errors. There are additional codes such as the single-byte-error-correcting, double-byte-error detecting (SBCDBD) codes, double-error-correcting, triple error-detecting (DEC-TED) codes that can correct multiple errors. The Single-error-correcting, Double-error-detecting and Double-adjacent-error-correcting (SEC-DED-DAEC) code provides a low cost ECC methodology to correct adjacent errors as proposed. The only drawback through this code is the possibility of miss-correction for a small subset of many errors.

LITERATURE SURVEY

Most prior work in memory ECC has focused on low failure rates present at normal operating voltages, and has not focused on the problem of persistent failures in cache memory operating at ultralow voltage where defect rates are very high.

Error correction codes (ECCs) have been used to protect memories for many years. There are wide ranges of codes that used or proposed for the applications in the memory. The codes that can correct one bit per word called the Single error correction are commonly used known as SEC. More sophisticated studies are carried on the codes that correct the two adjacent errors or the two errors. Further the use of more complex codes that corrects more errors is limited by their impact on delay and power, which limits their applicability to the design of memory. To surmount the issues, the use of codes that are one step majority logic decodable (OS -MLD) has been proposed recently. OS-MLD codes can be decoded with low latency and so they are used for the protection of memories. Another type of code that is OS-MLD is orthogonal Latin squares called the OLS codes. 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. The post-manufacturing customization approach proposed in this paper can be used to reduce the number of check bits and hence the amount of redundancy required in the memory while still providing the desired level of reliability. Note that the proposed approach does not reduce the hardware requirements for the OLS ECC as the whole code needs to be implemented on-chip since the location of the defects is not known until post-manufacturing test is performed.

PROPOSED METHOD

The concept of Latin squares and their applications are well known. A Latin square of size m is an m * m matrix that has permutations of the digits 0, 1,… and m-1 in both its rows and columns . There can be more than one Latin square for each value of m. In that case, two latin squares are said to be orthogonal if every ordered pair of elements appears only once when they are superimposed. Orthogonal Latin Squares (OLS) codes are derived from Orthogonal Latin squares. These codes have k=m2 data bits and 2tm check bits where ‘t’ is the number of errors that the codes can correct.OLS codes can be decoded using OS-MLD. OS-MLD is a simple procedure in which each bit is decoded by simply taking the majority value of the set of the recomputed parity check equations, in which it participates. This is shown in Fig. 1 for a given data bit di . The reasoning behind OS-MLD is that when an error occurs in bit di, the recomputed parity checks in which it participates will take a value of one. Therefore, a majority of ones in those recomputed checks is an indication that the bit is in error and therefore it needs to be corrected. However, it may also occur that errors in other bits different from di provoke a majority of ones that would cause miss-correction. For a few codes, their properties ensure that this miss-correction cannot occur, and therefore OS-MLD can be used. For a Double Error Correction (DEC) code t=2 and therefore 4m check bits are used. This means that to obtain a code that can correct t+1 errors, simply 2m check bits are added to the code that can correct

t errors. The modular property enables the selection of the error correction capability for a given word size. As mentioned in the introduction, OLS codes can be decoded using One Step Majority Logic Decoding (OS-MLD) as each data bit participates in exactly 2t check bits and each other bit participates in at most one of those check bits. This enables a simple correction when the number of bits in error is 't' or less. The 2t check bits are recomputed and a majority vote is taken, if a value of one is obtained, that bit is error and must be corrected. Otherwise it is correct. As long as number of errors is t or less ensures the error correction as the remaining 't-1' errors can, in the worst case it effect t-1 check bits so that still a majority of t+1 triggers the correction of an erroneous bit. For an OLS code that can correct t errors using OS-MLD, t+1 errors can cause miss-corrections. This occurs for example if the errors affect t+1 parity bits in which bit di participates as this bit will be misscorrected. The same occurs when the number of errors is larger than t+1. Each of the 2t check bits in which a data bit participates is taken from a group of m parity bits. Those groups are bits 1 to m, m+1 to 2m, 2m+1 to 3m and 3m+1 to 4m.

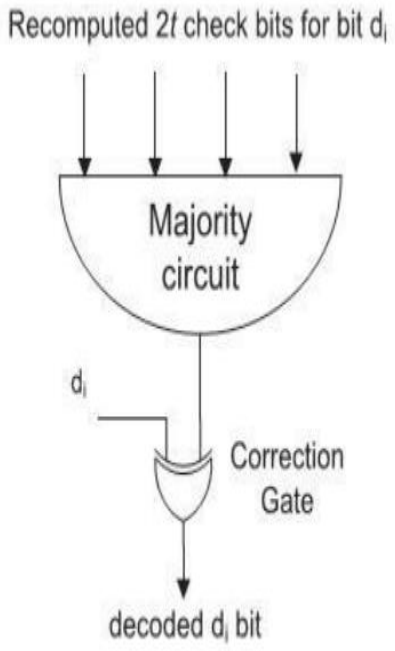


Fig-1: Illustration of OS-MLD decoding for OLS codes

The parity matrix for OLS codes is built from their properties. The matrix is capable of correcting two errors. By the fact that in direction of the modular structure it might be able to correct many errors. The OLS codes have check bits of number "2tm" in which "t" stands for number of errors that code can corrects.

111100000000000	1000000000000000
000011110000000	0100000000000000
000000011110000	0010000000000000
000000000001111	0001000000000000
100010001000100	0000100000000000
010001000100010	0000010000000000
001000100010001	0000001000000000
100001000100001	0000000100000000
010010000010010	0000000010000000
001000110000100	0000000001000000
000100100100100	0000000000100000
100000100001010	0000000000010000
010000100101000	0000000000001000
001010000100001	0000000000000100
000101001000010	0000000000000010
	0000000000000001

Fig-2: Parity check matrix for OLS code with k = 16 and t = 2.

The implementing method is based on the observation that by construction, the groups formed by the m parity bits in each Mi matrix have at most a one in every column of parity matrix. For

the example in Fig. 2, those groups correspond to bits (or rows) 1– 4, 5–8, 9–12 and 13–16. Therefore, any combination of four bits from one of those groups will at most share a one with the existing columns in parity matrix. For example, the combination formed by bits 1, 2, 3, and 4 shares only bit 1 with columns 1, 2, 3, and 4. This is the condition needed to enable OS-MLD. Therefore, combinations of four bits taken all from one of those groups can be used to add data bit columns to the parity matrix. For the code with k=16 and t=2 shown in Fig. 2, we have m=4. Hence, one combination can be formed in each group by setting all the positions in the group to one. This is shown in Fig. 3, where the columns added are highlighted. In this case, the data bit block is extended from k= 16 to k=20 bits.

$$\left[\begin{array}{c|c|c}
1111000000000000 & 1000 & 1000000000000000 \\
0000111100000000 & 1000 & 0100000000000000 \\
0000000011110000 & 1000 & 0010000000000000 \\
0000000000001111 & 1000 & 0001000000000000 \\
1000100010001000 & 0100 & 0000100000000000 \\
0100010001000100 & 0100 & 0000010000000000 \\
0010001000100010 & 0100 & 0000001000000000 \\
0001000100010001 & 0100 & 0000000100000000 \\
1000100001000100 & 0010 & 0000000010000000 \\
0100100000010010 & 0010 & 0000000001000000 \\
0010000110000100 & 0010 & 0000000000100000 \\
0001001001001000 & 0010 & 0000000000010000 \\
1000010000010100 & 0001 & 0000000000001000 \\
0100000100101000 & 0001 & 0000000000000100 \\
0010100001000001 & 0001 & 0000000000000010 \\
0001010010000010 & 0001 & 0000000000000001
\end{array}\right]$$

Fig- 3: Parity check matrix for the extended OLS code with k= 20 and t= 2.

The implementing method first divides the parity check bits in groups of m bits given by the Mi matrices. Then, the second step is for each group to find the combinations of 2t bits such that any pair of them share at most one bit. This second step can be seen as that of constructing an OS-MLD code with m parity check bits. Obviously, to keep the OS-MLD property for the extended code, the combinations formed for each group have to share at most one bit with the combinations formed in the other $2t - 1$ groups. When m is small, such combinations are found easily. When m is larger, several combinations can be formed in each group. This occurs, for example, when m = 8. In this case, the OLS code has a data block size k = 64. With eight positions in each group, now two combinations of four of them that share at most one position can be formed. This means that the extended code will have eight (4×2) additional data bits. As the size of the OLS code becomes larger, the number of combinations in a group also grows. For the case m = 16 and k= 256, each group has 16 elements. Interestingly enough, there are 20 combinations of four elements that share at most one element. In fact, those combinations are obtained using the extended OLS code shown in Fig. 3 in each of the groups. Therefore, in this case, $4 \times 20 = 80$ data bits can be added in the extended code. The parameters of the extended codes are shown in Table I, where $n - k = 2tm$ is the number of parity bits. The data block size for the original OLS codes (kOLS) is also shown for reference.

The method can be applied to the general case of an OLS code with $k = m2$ that can correct t errors. Such a code has 2tm parity bits that as before are divided in groups of m bits. The code can be extended by selecting combinations of 2t parity bits taken from each of the groups. These combinations can be added to the code as long as

any pair of the new combinations share at most one bit. When m is small, a set of such combinations with maximum size can be easily found. However, as m grows, finding such a set is far from trivial (as mentioned before, solving that problem is equivalent to designing an OS-MLD code with m parity bits that can correct t errors). An upper bound on the number of possible combinations can be derived by observing that any pair of bits can appear only in one combination. Because each combination has 2t bits, there are (2t 2) pairs in each combination. The number of possible pairs in each group of m bits is m2 . Therefore, the number of combinations NG in a group of m bits has to be such that

$$\binom{m}{2} \geq \binom{2t}{2} * N_G$$

which can be simplified as $\frac{m^2-m}{4t^2-2t} \geq N_G$

PROPOSED METHOD

PROPOSED CODES

In terms of computing time, it is possible for QAEC codes with 16 data bits to have access to all the solutions, but it is impossible for QAEC codes with 32 and 64 data bits. Therefore, in this paper, the best solutions are presented for QAEC codes with 16 data bits and the best solutions found in a reasonable time (one week) using the proposed searching algorithm are presented for QAEC codes with 32 and 64 data bits. In terms of the two optimization criteria mentioned in Section III, for codes (23, 16), the two best parity check matrices are shown in Figs. 6 and 7 with both criteria best optimized. The parity check matrix for (40, 32) optimized to reduce the total number of one's is shown in Fig. 8 and the parity check matrices for (40, 32) optimized to reduce the maximum number of ones in a row is shown in Fig. 9. For codes (73, 64), the solutions obtained with both criteria better optimized are the same. The matrix is shown in Fig. 10.

VI. PROCEDURE OF ENCODING AND DECODING FOR QAEC CODES

In this section, we elaborate on the encoding and decoding procedure of the proposed 3-bit BEC-QAEC codes. The fundamental theory of encoding and decoding were discussed in Section II. Here, an example for 16 data bits is illustrated in Fig. 11 with the H matrix used in Fig. 6. Based on the structure of the parity check matrix, the check bits are calculated by the corresponding data bits. The new encoded codeword, the combination of check bits and data bits is stored in the memory. When the particles hit the memory resulting in MBUs, the contents of affected memory cells are flipped. Here, to elaborate on the correction ability of QAEC codes, quadruple adjacent bits are flipped on D2, D3, D4, and D5. In the decoding process, the syndrome is calculated using the stored check bits and data bits and the structure of the parity check matrix. Through the corresponding relationship between the syndrome and the XOR result of the columns mentioned in Section II, the flipped bits can be located. With the flipped bits inverted, the errors from the storage stage in the memory are effectively corrected. This is the whole procedure of encoding and decoding for the proposed QAEC codes.

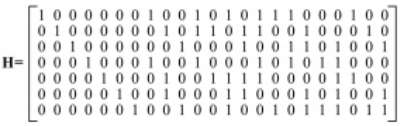


Fig. 6. Best parity check matrix for the (23, 16) 3-bit burst with QAE optimized to reduce the total number of ones and the maximum number ones in a row.

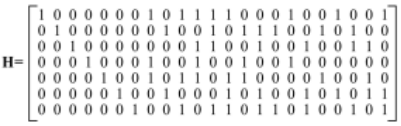
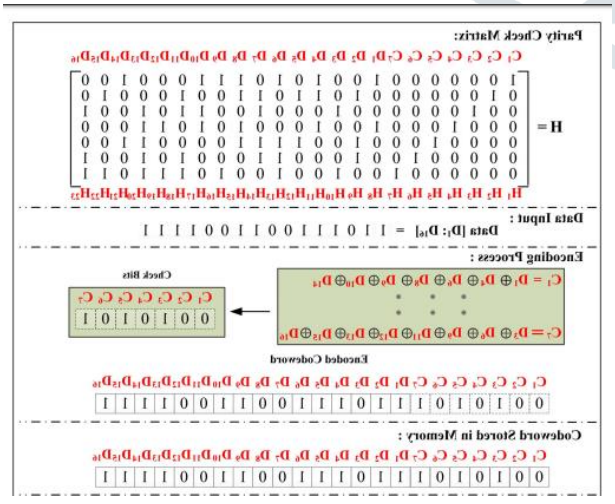


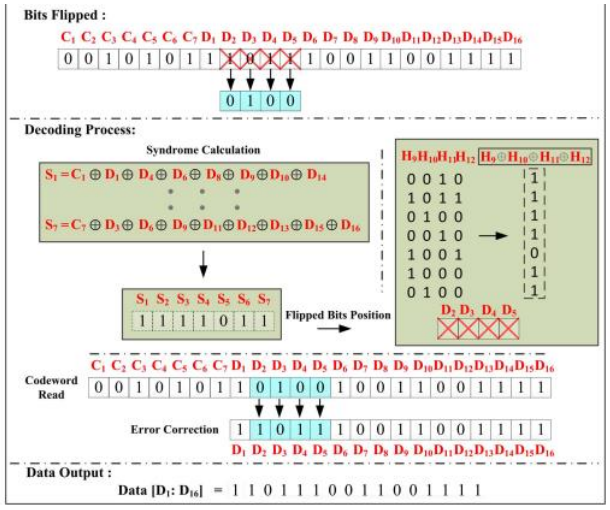
Fig. 7. Best parity check matrix for the (23, 16) 3-bit burst with QAE optimized to reduce the total number of ones and the maximum number ones in a row.

Fig. 8. Parity check matrix for the (40, 32) 3-bit burst with QAEC optimized to reduce the total number of ones.

Fig. 9. Parity check matrix for the (40, 32) 3-bit burst with QAEC optimized to reduce the maximum number of ones in a row.

Fig. 10. Parity check matrix for the (73, 64) 3-bit burst with QAEC optimized to reduce the total number of ones and the maximum number of ones in a row.





RESULTS

The Xilinx ISE software used to perform compilation and synthesis. A test bench is created to execute the simulation. The decoder is stimulated using the Xilinx ISE software. The simulation results for the decoder using OLS code with k=16, t=2, r_data=32 bits (16 data bits and 16 parity bits) and the decoder using extended OLS code with k=20, t=2, r_data=36 (20 data bits and 16 parity bits) is shown as follows, where k- data bits, t- number of errors that extended OLS code can correct, r_data - input of decoder and data_op1 - output. Simulation data: d= 0010_1000_0001_1000. The error is introduced in d0, d1 bits. The corrected simulation output seen in fig 4 with k = 16 bits and t = 2 errors.

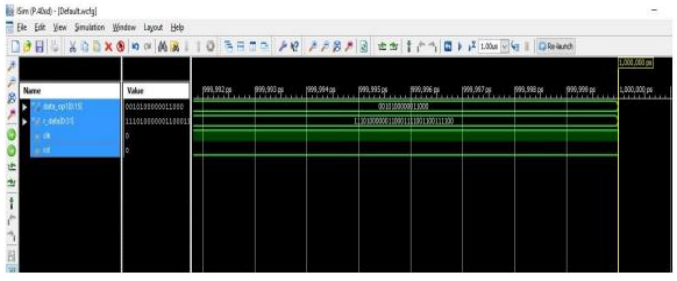


Fig- 4: For k = 16 bits OLS codes.

For d = 0010_1000_0001_1000_0100. The error is introduced in d4 ,d5 bits. The simulation output

with corrected output seen in fig 5 with k= 20 bits and t = 2 errors.

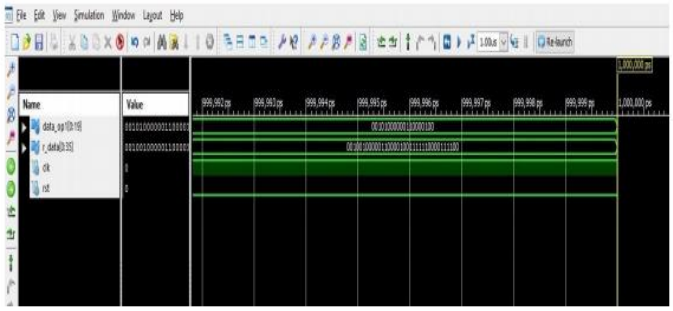


Fig- 5: For k= 20 bits extended OLS codes.

CONCLUSION

In this brief, error correction technique for extended OLS codes is implemented. The extended codes have the same number of parity bits as the original OLS codes but a larger number of data bits. The data bits in extended OLS codes are 20 bits whereas parity bits are 16 bits. The number of errors extended OLS codes can correct is 2 adjacent errors. Therefore, the relative overhead is smaller. The derived codes can be decoded using OS-MLD as the original OLS codes. The decoding area and delay are also similar. Therefore, the new codes can be an interesting option to reduce the number of parity bits required to implement multiple bit error correction in memories or caches.

REFERENCES

[1] P. P. Vaidyanathan, Multirate Systems and Filter Banks, Englewood Cliffs, N.J., USA: Prentice Hall, 1993.

[2] A. Sibille, C. Oestges and A. Zanella, MIMO: From Theory to Implementation, New York, NY, USA: Academic, 2010.

[3] N. Kanekawa, E. H. Ibe, T. Suga and Y. Uematsu, Dependability in Electronic Systems: Mitigation of Hardware Failures, Soft Errors, and ElectroMagnetic Disturbances, New York, NY, USA: Springer Verlag, 2010.

[4] M. Nicolaidis, "Design for soft error mitigation," IEEE Trans. Device Mater. Rel., vol. 5, no. 3, pp. 405–418, Sep. 2005.

[5] 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.

[6] A. Reddy and P. Banarjee "Algorithm-based fault detection for signal processing applications," IEEE Trans. Comput., vol. 39, no. 10, pp. 1304–1308, Oct. 1990.

[7] S. Pontarelli, G. C. Cardarilli, M. Re, and A. Salsano, "Totally fault tolerant RNS based FIR filters," in Proc. IEEE IOLTS, 2008, pp. 192–194.

[8] Z. Gao, W. Yang, X. Chen, M. Zhao and J. Wang, "Fault missing rate analysis of the arithmetic residue codes based fault-tolerant FIR filter design," in Proc. IEEE IOLTS, 2012, pp. 130–133.

[9] B. Shim and N. Shanbhag, "Energy-efficient soft error-tolerant digital signal processing," IEEE Trans. Very Large Scale Integr. Syst., vol. 14, no. 4, pp. 336–348, Apr. 2006.

[10] Y.-H. Huang, "High-efficiency soft-error-tolerant digital signal processing using fine-grain subword-detection processing," IEEE Trans. Very Large Scale Integr. Syst., vol. 18, no 2, pp. 291–304, Feb. 2010.