ERROR CORRECTION IN EXTENDED ORTHOGONAL LATIN SQUARE CODES USING SYNDROME FAULT DETECTION AND MAJORITY LOGIC DECODING

Dr.Nikhil Raj [1], Dr. Om prakash[2], Dr M. Thamarai[3], Dr. Vaibhav A Meshram[4], Dr.K. Srinivasulu[5]
[1,2,3,4,5]Professor, Dept. of ECE,MRCE, Hyderabad

Abstract:

Error correction codes (ECCs) are commonly used to Protect memories from errors. As multi-bit errors become more frequent, single error correction codes are not enough and more advanced ECCs are needed. The use of advanced ECCs in memories is, however, limited by their decoding complexity. In this context, one-step majority logic decodable (OS-MLD) codes are an interesting option as the decoding is simple and can be implemented with low delay. Orthogonal Latin squares (OLS) codes are OS-MLD and have been recently considered to protect caches and memories. The main advantage of OLS codes is that they provide a wide range of choices for the block size and the error correction capabilities. We can also extend these codes to accommodate more number of data bits thus reducing the overhead. But most of the time all the words in the memory are not error prone, but still we try to decode them and waste clock cycles on it. In this brief, a method is presented to detect whether an error is present in the code word and if present then only the correction is done using majority logic decoding.

Keywords—Error correction codes (ECCs), Extended Orthogonal Latin squares, Syndrome fault detection (SFD), majority logic decoding, and memory.

I. INTRODUCTION

To mitigate errors, error correction codes (ECCs) are commonly used in memories [1]. Because of their simplicity, single error correction codes that can correct one bit per word are traditionally used [2]. Other codes that can also correct double adjacent errors [3] or double errors in general have also been studied [4]. Codes that can correct more errors have a larger impact on delay and power that can limit their applicability to memory designs [5]. One alternative to minimize those impacts is to use codes that are one-step majority logic decodable (OS- MLD). OS-MLD codes can be decoded with low latency and are, therefore, attractive to protect memories [6]. Several types of OS-MLD codes have been proposed for memory protection. One example is a type of Euclidean geometry (EG) codes studied in [7] and [8].

EG codes provide a limited number of block sizes and error correction capabilities. For example, for double error correction (DEC), only very small data block sizes (smaller than 16 bits) can be implemented. In addition, the error correction capability for a block size is fixed and cannot be adapted to the error rate. Another type of code that is OSMLD is orthogonal Latin squares (OLS) code [11]. OLS codes can be implemented for a wide range of block sizes and error correction capabilities. This flexibility and the simple and fast decoding are the main advantages of OLS codes. However, OLS codes typically require more parity bits than other codes to correct the same number of errors. In some applications, this disadvantage is offset by their modularity and the simple and low delay decoding implementation (as OLS codes are OS-MLD). For example, OLS codes have been recently considered to protect memories [12], caches [13], and interconnections [14].

The rest of this brief is organized as follows. Section II provides an overview of OLS and

Extended OLS codes summarizing some of their properties that are used in the rest of this paper. Then, the proposed method for error detection and correction is presented in Section III. Section IV speaks of the results. Finally, the conclusions are presented in Section V.

II. OLS and Extended OLS Codes

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 [15].Two Latin squares are said to be orthogonal if when they are superimposed every ordered pair of elements appears only once. OLS codes are derived from OLS [11]. The block sizes for OLS codes are k = m2 data bits and 2 tm parity bits, where t is the number of errors that the code can correct and m is an integer. For a given pair of values of t and m, the corresponding OLS code exists only if there are at least 2t OLS of size m.

The extended codes have the same number of parity bits as the original OLS codes but a larger number of data bits. 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.

$$
\begin{array}{c}
\left[\begin{array}{cccccccccccccccc|cccc|cccccccccccccccc}
1&1&1&1&0&0&0&0&0&0&0&0&0&0&0&0&1&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&1&1&1&1&0&0&0&0&0&0&0&0&1&0&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&1&1&1&1&0&0&0&0&1&0&0&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&1&1&1&1&1&0&0&0&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0&0\\
1&0&0&0&1&0&0&0&1&0&0&0&1&0&0&0&0&1&0&0&0&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0\\
0&1&0&0&0&1&0&0&0&1&0&0&0&1&0&0&0&1&0&0&0&0&0&0&0&1&0&0&0&0&0&0&0&0&0&0\\
0&0&1&0&0&0&1&0&0&0&1&0&0&0&1&0&0&1&0&0&0&0&0&0&0&0&1&0&0&0&0&0&0&0&0&0\\
0&0&0&1&0&0&0&1&0&0&0&1&0&0&0&1&0&1&0&0&0&0&0&0&0&0&0&1&0&0&0&0&0&0&0&0\\
1&0&0&0&0&1&0&0&0&0&1&0&0&0&0&1&0&0&1&0&0&0&0&0&0&0&0&0&1&0&0&0&0&0&0&0\\
0&1&0&0&1&0&0&0&0&0&0&1&0&0&1&0&0&0&1&0&0&0&0&0&0&0&0&0&0&1&0&0&0&0&0&0\\
0&0&1&0&0&0&0&1&1&0&0&0&0&1&0&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0&1&0&0&0&0&0\\
0&0&0&1&0&0&1&0&0&1&0&0&1&0&0&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0&0&1&0&0&0&0\\
1&0&0&0&0&0&1&0&0&0&0&1&0&1&0&0&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0&0&1&0&0&0\\
0&1&0&0&0&0&0&1&0&0&1&0&1&0&0&0&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&0\\
0&0&1&0&1&0&0&0&0&1&0&0&0&0&0&1&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0\\
0&0&0&1&0&1&0&0&1&0&0&0&0&0&1&0&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1
\end{array}\right]\\
\text{Added columns}
\end{array}
$$

Fig. 1. H matrix for the extended OLS codes with k = 20 and t = 2.

In the above figure, if we remove the extra added columns we get the Parity check matrix H for the OLS codes with codes k
= 16 and t = 2.

From the figure, we can infer that every column in the matrix is having exactly 4 one's. Thus 4 parity checksums can be formed which can be majority decoded. Not all orthogonal Latin squares can be used as OLS-MLD codes; they need to satisfy 2 main conditions:

1) Each data bit participates in exactly 2t parity check bits;
2) Each other data bit participates in at most one of those parity check bits.

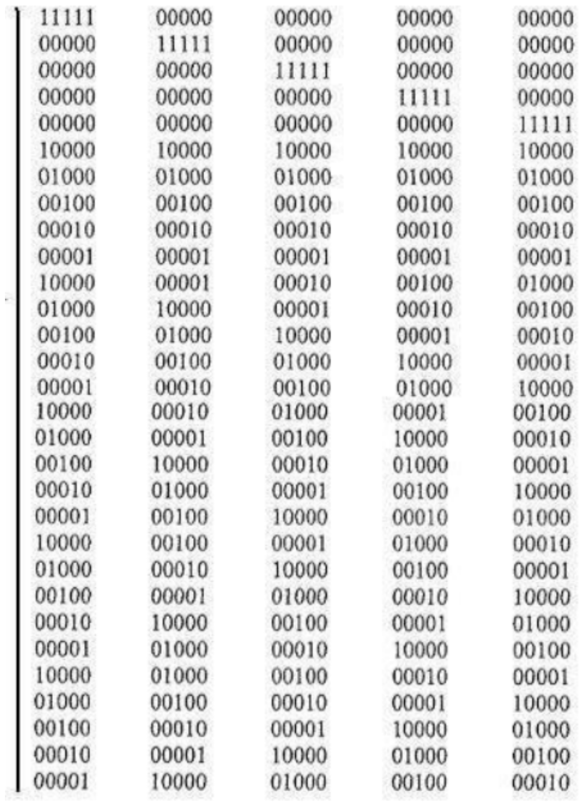
For an arbitrary value of k = m2, the H matrix for a DEC OLS code is constructed as follows:

$$
H = \begin{bmatrix} M_1 & \\ M_2 & \\ M_3 & I_{4m} \\ M_4 & \end{bmatrix} \qquad (1)
$$

where I4m is the identity matrix of size 4m and M1, M2, M3, and M 4 are the matrices of size m × m2 derived from OLS of size m × m. In a general case, for an OLS code that can correct t errors, the parity check matrix is constructed using 2t Mi matrices.

When m is small finding, such combinations is easy. For example, in the case considered in Fig. 1, there is only one possible combination per group. 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. 1 in each of the groups. Therefore, in this case, 4 × 20 = 80 data bits can be added in the extended code.

An example for parity check matrix for triple bit error correction can be shown as follows:



11111	00000	00000	00000	00000
00000	11111	00000	00000	00000
00000	00000	11111	00000	00000
00000	00000	00000	11111	00000
00000	00000	00000	00000	11111
10000	10000	10000	10000	10000
01000	01000	01000	01000	01000
00100	00100	00100	00100	00100
00010	00010	00010	00010	00010
00001	00001	00001	00001	00001
10000	00001	00010	00100	01000
01000	10000	00001	00010	00100
00100	01000	10000	00001	00010
00010	00100	01000	10000	00001
00001	00010	00100	01000	10000
10000	00010	01000	00001	00100
01000	00001	00100	10000	00010
00100	10000	00010	01000	00001
00010	01000	00001	00100	10000
00001	00100	10000	00010	01000
10000	00100	00001	01000	00010
01000	00010	10000	00100	00001
00100	00001	01000	00010	10000
00010	10000	00100	00001	01000
00001	01000	00010	10000	00100
10000	01000	00100	00010	00001
01000	00100	00010	00001	10000
00100	00010	00001	10000	01000
00010	00001	10000	01000	00100
00001	10000	01000	00100	00010

Fig. 2. Parity check matrix H for the OLS codes with k = 25 and t = 3.

The above parity check matrix consists of 25 data bits and 30 check bits. This matrix is having 6 one's in each column thus, can correct up to 3 bit errors. This is a non symmetric code which means that the number of data bits is not equal to the number of check bits. It is a trivial task to extend these types of OLS codes.

The overhead on the code word is very much higher in case of the TEC codes when compared to the DEC codes. As discussed before, the study of methods to find the optimum set of combinations that can be used to extend the code in a general case is a complex mathematical problem that is left for future work.

III. PROPOSED METHOD

Normally, all the words in the memory are not error prone. Just a few words say 1 out of every 20 words is having an error. But using a simple correction scheme we will be wasting the processor clock cycles to correct a valid code word. In this paper a method is presented to first detect the presence of an error in the code word using the syndrome fault detection. Later if any error exists then it is sent to the correction unit for further processing. After getting the output of the correction unit, once again the syndrome fault detection is performed to check for errors. If we get an error at this stage then we simply ask the transmitter to re transmit the word again as it will not able to correct those errors.

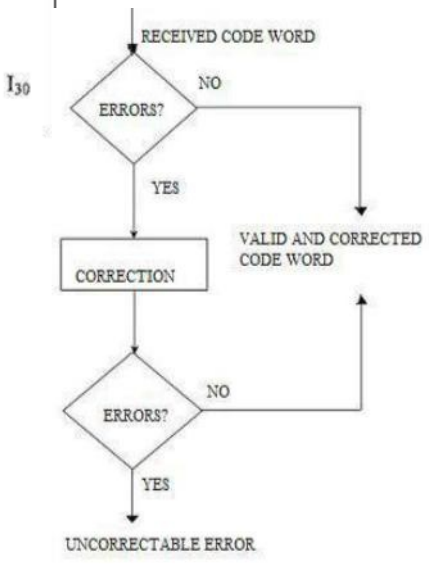


Fig. 3. Graphical representation of the proposed method.

The main idea of the proposed method is to speed-up the detection process where ever possible and also to save the power that might be wasted for unnecessary correction cycles.

A. SYNDROME FAULT DETECTION

The syndrome of a particular code can be formed by multiplying the received code with the transpose of the parity check matrix. The syndrome is independent of the transmitted code but is dependent on the error pattern that might have affected the code in the channel or in the memory. After getting the syndrome vector, check if all the bits in the vector are 0's. If all are 0's then there is no error present but if at least one bit is 1 then the data needs to be corrected.

B. MAJORITY LOGIC DECODING

In this method, the parity check sum of each bit in the received code vector is computed and they are fed to a majority logic gate that will decide whether the bit is in error or not.

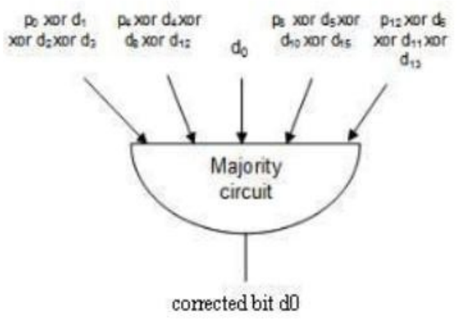


Fig. 4. Illustration of OS-MLD decoding for OLS codes.

The above figure shows the OS-MLD implementation for 0th bit of (32, 16) OLS code that has been shown in Fig.1

IV. RESULTS

Three different implementations have been simulated and dumped on to Virtex -4 FPGA kit and the simulated outputs are verified using the ChipScope tool. Fig.5 represents the simulation results for (32,16) code which can be used to correct up to 2 bit errors and also will be having an message length of 16 bits and check length of 16 bits. Fig.6 represents the simulation results for (55,25) code which can be used to correct up to 3 bit errors and will have an message length of

25. Fig.7 is the result of extended OLS codes having 16 check bits and 20 data bits. The main important thing that we need to consider is the hamming distance and valid code words of a particular code. 88995 and 88a53 are two messages that are encoded as 88995cf3a and 88a53cf3a respectively. Now say, if an error pattern of 003c0000 occurred in 88995cf3a, then that is transformed into 88a53cf3a, another valid word by the corrector. Thus at this instant if we check for an error we will feel the data is error free, but the originality is different. That’s what we can infer from the simulation results, this is one case. There is another case of silent error corruption before correction only we get a valid code word which is not the original valid code word. This is main disadvantage of many of the decoding techniques that we use, as we cannot differentiate between the errors that change one valid code word into another.

By using the proposed method we can save the processor clock cycles which we might be wasting on unnecessary correction of code words and also find out if the number of errors that occurred are more than the number of errors that a particular code can correct and ask for a retransmission of the same if more errors occur. The only exceptional case is that of the valid code words which is discussed earlier. This method can be used for both forward error correcting and as well as ARQ schemes.

It can be observed that the data block sizes of the extended codes are not a power of two. As in many memory applications, the data block size is a power of two; this may limit the use of the extended OLS codes. However, in some specific applications, word sizes that are not a power of two are used and in those cases, the proposed codes can be useful. The codes can also be used when the memory is extended to incorporate flags or tags as is the case in caches. For example, in a cache with a 256-bit data line, the extended DEC code

with m =16 can be used to support up to 80-tag bits. This will

be more than enough for a 64-bit processor. Another potential application is the protection of content addressable memories (CAMs) and their associated data.

V. CONCLUSION

In this brief, a method to detect errors in extended OLS codes and correct them has been proposed. The extended codes have the same number of parity bits as the original OLS codes but a larger number of data bits. Therefore, the relative overhead is smaller. The extended OLS codes are already hardware redundant in nature and extra detection circuitry to the above makes it more redundant. But looking on the brighter side we have faster decoding of the code word and these codes are simple to implement to a certain level. New methods can be developed in near future to extend the non-symmetric OLS codes. And the derived codes can be decoded using OS- MLD as the original OLS codes. The decoding area and delay area 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. The proposed method can be applied to any OLS code but in some cases, obtaining the combinations to extend the code is difficult. This can be formalized as a mathematical problem that involves the design of OS -MLD codes with smaller data block sizes. The study of this complex problem is left for future work. Most of the codes derived in this brief are double ECCs and few triple ECCs. The use of the method for codes that can correct more than three errors will be also addressed in future work. In any case, as discussed in this brief, the proposed method is expected to provide better benefits for double ECCs

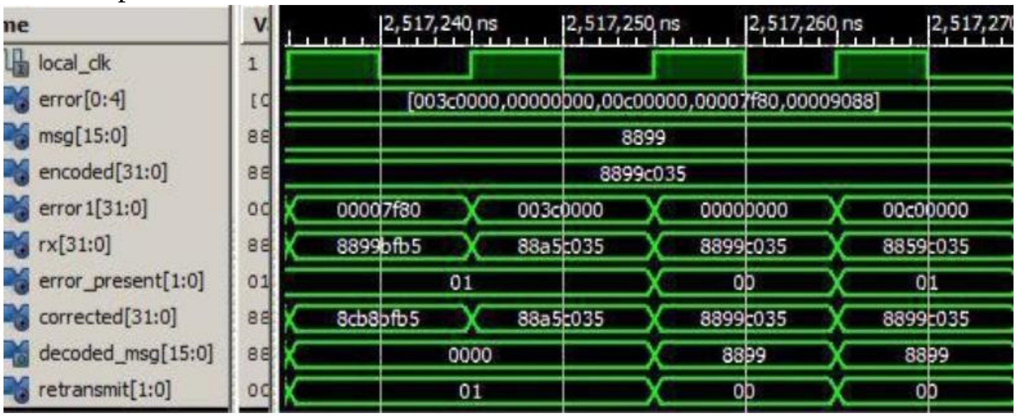


Fig. 5. Simulation results for the OLS codes with k = 16 and t = 2

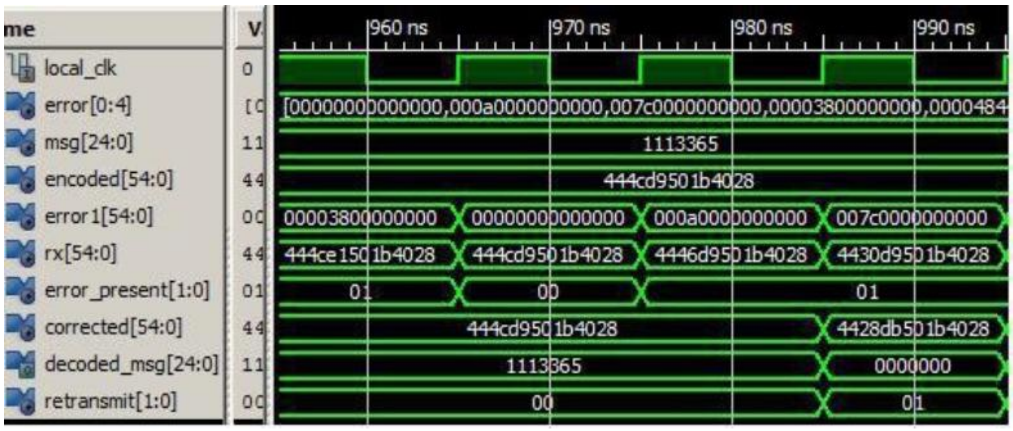


Fig. 6. Simulation results for the OLS codes with k = 25 and t = 3.

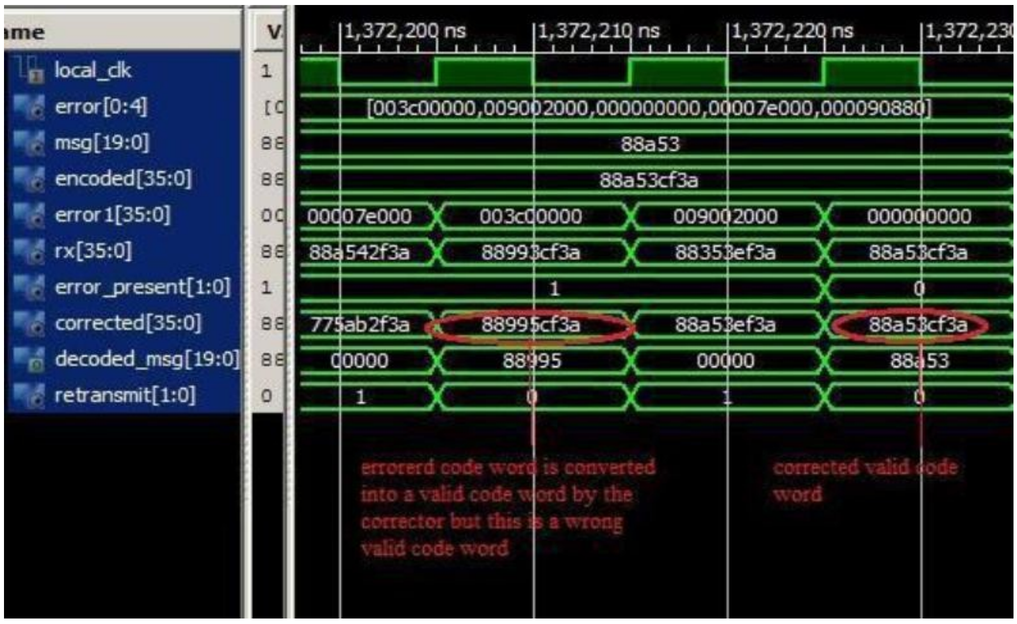


Fig. 7. Extended OLS codes with k = 20, t = 3 and message = "88a53"

References

[1] Pedro Reviriego, Salvatore Pontarelli, Alfonso Sánchez- Macián, and Juan Antonio

Maestro, “A Method to Extend Orthogonal Latin Square Codes,” IEEE Transactions on very large scale integration (VLSI) systems, vol. 22, NO. 7, July 2014

[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] M. Y. Hsiao, ―A class of optimal minimum odd-weight column SEC- DED codes,‖ IBM J. Res. Develop., vol. 14, no. 4, pp. 395–301, Jul. 1970.

[4] A. Dutta and N. A. Touba, ―Multiple bit upset tolerant memory using a selective cycle avoidance based SEC-DED-DAEC code,‖ in Proc. 25th IEEE VLSI Test Symp., May 2007, pp. 349–354.

[5] R. Naseer and J. Draper, ―DEC ECC design to improve memory reliability in sub-100 nm technologies,‖ in Proc. IEEE ICECS, Sep. 2008, pp. 586–589.

[6] G. C. Cardarilli, M. Ottavi, S. Pontarelli, M. Re, and A. Salsano,

―Fault tolerant solid state mass memory for space applications,‖ IEEE Trans. Aerosp. Electron. Syst., vol. 41, no. 4, pp. 1353–1372, Oct. 2005.

[7] S. Liu, P. Reviriego, and J. A. Maestro, ―Efficient majority logic fault detection with difference-set codes for memory applications,‖ IEEE Trans. Very Large Scale Integr. (VLSI) Syst., vol. 20, no. 1, pp. 148– 156, Jan. 2012.

[8] P. Reviriego, M. Flanagan, S. Liu, and J. A. Maestro, ―Multiple cell upset correction in memories using difference set codes,‖ IEEE Trans. Circuits Syst. I, Regular Papers, vol. 59, no. 11, pp. 2592–2599, Nov. 2012.

[9] M. Y. Hsiao, D. C. Bossen, and R. T. Chien, ―Orthogonal Latin square codes,‖ IBM J. Res. Develop., vol. 14, no. 4, pp. 390–394, Jul. 1970.

[10]R. Datta and N. A. Touba, ―Generating burst-error correcting codes from orthogonal Latin square codes—A graph theoretic approach,‖ in Proc. IEEE Int. Symp. DFT, 2011, pp. 367–373.

[11]A. R. Alameldeen, Z. Chishti, C. Wilkerson, W. Wu, and S.-L. Lu,

―Adaptive cache design to enable reliable low-voltage operation,‖

IEEE Trans. Comput., vol. 60, no. 1, pp. 50–63, Jan. 2011.

[12]S. E. Lee, Y. S. Yang, G. S. Choi, W. Wu, and R. Iyer, ―Low-power, resilient interconnection with orthogonal Latin squares,‖ IEEE Design Test Comput., vol. 28, no. 2, pp. 30–39, Mar./Apr. 2011.

[13]J. Dénes and A. D. Keedwell, Latin Squares and Their Applications. San Francisco, CA, USA: Academic, 1974.

[14]C. McNairy and D. Soltis, ―Itanium 2 processor microarchitecture,‖

IEEE Micro, vol. 23 no. 2, pp. 44–55, Mar./Apr. 2003.

[15]J. E. Stine, I. Castellanos, M. Wood, J. Henson, F. Love, W. R. Davis,

P. D. Franzon, M. Bucher, S. Basavarajaiah, J. Oh, and R. Jenkal,

―FreePDK: An open-source variation-aware design kit,‖ in Proc. IEEE Int.

Conf. MSE, Jun. 2007, pp. 1

[16]H. Jaber, F. Monteiro, S. J. Piestrak, andA. Dandache, “Design of parallel fault-secure encoders for systematic cyclic block trans- mission codes,” Microelectron. J., vol. 40, no. 12, pp. 1686–1697, Dec.2009.

[17]S. J.Piestrak,A. Dandache,and F. Monteiro, “Designing fault-secure parallelencoders forsystematic linearerror correctingcodes,” IEEE

Trans.Reliab.,vol.52,no.4,pp.492–500,Apr.20 03. [18]J.A.Maestro,P.Reviriego,C.Argyrides,

andD.K.Pradhan,“Fault tolerant singleerror correction

encoders,” J. Electron.Test., TheoryAppl.,vol.27,no.2,pp.215–218,Apr.2011.