A Method to Extend Orthogonal Latin Square Codes

Pedro Reviriego, Salvatore Pontarelli, Alfonso Sánchez-Macián, and Juan Antonio Maestro

***Abstract*— Error correction codes (ECCs) are commonly used to protect memories from errors. As multibit 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. In this brief, a method to extend OLS codes is presented. The proposed method enables the extension of the data block size that can be protected with a given number of parity bits thus reducing the overhead. The extended codes are also OS-MLD and have a similar decoding complexity to that of the original OLS codes. The proposed codes have been implemented to evaluate the circuit area and delay needed for different block sizes.**

***Index Terms*— Error correction codes (ECCs), Latin squares, majority logic decoding, 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]. The use of difference set (DS) codes has also been recently analyzed in [9] and [10]. Both EG and DS 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 OS-MLD is orthogonal Latin squares (OLS) code [11]. OLS codes can be implemented for a wide range of block sizes and error correction capabilities. The error correction capability for a given data block size can also be easily adapted by simply adding or removing some parity bits. 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].

Manuscript received January 4, 2013; revised April 26, 2013; accepted July 25, 2013. Date of publication August 7, 2013; date of current version June 23, 2014. This work was supported in part by the Spanish Ministry of Science and Education under Grant AYA2009-13300-C03, and collaboration in the framework of COST ICT Action 1103 "Manufacturable and Dependable Multicore Architectures at Nanoscale."

P. Reviriego, A. Sánchez-Macián, and J. A. Maestro are with the Departamento de Ingeniería Industrial, Universidad Antonio de Nebrija, Madrid 28040, Spain (e-mail: previrie@nebrija.es; asanchep@nebrija.es; jmaestro@nebrija.es).

S. Pontarelli is with the Department of Electronic Engineering, University of Rome "Tor Vergata," Rome 00133, Italy (e-mail: pontarelli@ing.uniroma2.it).

Digital Object Identifier 10.1109/TVLSI.2013.2275036

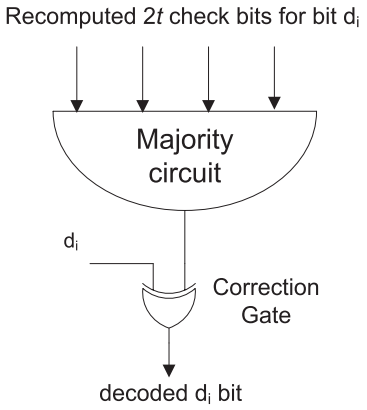

Fig. 1. Illustration of OS-MLD decoding for OLS codes.

In this brief, a method to extend OLS codes such that larger data blocks can be protected with the same number of parity bits is presented. This makes the extended codes more cost effective. The modification does not affect the decoding that can be performed using OS-MLD. The decoders for different block sizes have been implemented in Hardware Description Language (HDL) and synthesized to estimate the required area and delay. The results show that the complexity and delay are similar to that of traditional OLS codes.

The rest of this brief is organized as follows. Section II provides an overview of OLS codes summarizing some of their properties that are used in the rest of this paper. Then, the proposed method to extend OLS codes is presented in Section III, detailing the parameters of the extended codes for different block sizes. Section IV evaluates the complexity of the decoders for the extended codes in terms of circuit area and delay. Finally, the conclusions are presented in Section V.

II. OLS Codes

A Latin square of size m is an $m \times m$ 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 $k = m^2$ 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.

As mentioned in Section I, 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 [6]. This is shown in Fig. 1 for a given data bit d_i. The reasoning behind OS-MLD is that when an error occurs in bit d_i, 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

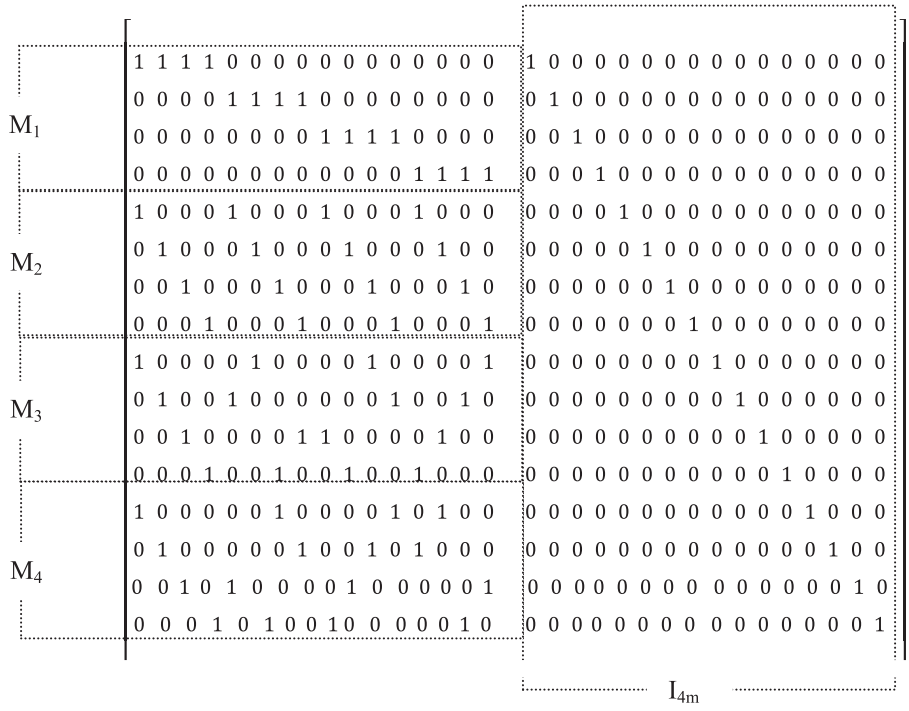

Fig. 2. Parity check matrix H for the OLS code with $k = 16$ and $t = 2$.

the bit is in error and therefore needs to be corrected. However, it may also occur that errors in other bits different from d_i provoke a majority of ones that would cause miscorrection. For a few codes, their properties ensure that this miscorrection cannot occur, and therefore OS-MLD can be used. This is the case for some EG codes, DS codes, and for OLS codes as mentioned in Section I.

In particular, OLS are by construction such that:

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.

This enables the use of OS-MLD as for a number of errors t or smaller, when one error affects a given bit, the remaining $t - 1$ errors can, in the worst case affect $t - 1$ check bits. Therefore, still a majority of $t + 1$ triggers the correction of an erroneous bit. In a similar manner, when a given bit is correct, t errors on other bits will not cause misscorrection as a majority of $t + 1$ is needed. As shown in Fig. 1, the use of OS-MLD enables a simple and fast decoding that is interesting for high-speed memories.

As mentioned before, the parity check matrix H for OLS codes is constructed from the OLS. As an example, the matrix for a code with $k = m^2 = 16$ data bits and $2\ tm = 16$ parity bits that can correct double errors is shown in Fig. 2. As explained before, the decoding starts by recomputing the parity checks. The result is the syndrome and a value of one in a given parity check (rows of H) means that an error has been detected in that parity check. A given data bit participates in the parity checks that have a one in the column that corresponds to that bit. For example, the first column that corresponds to the first data bit has ones in positions 1, 5, 9, and 13. For OLS codes, as described before, the decoding is done by taking a majority vote of the syndrome bits in which the bit participates (1, 5, 9, and 13 in our example). If the majority is one, then the data bit is in error and is corrected by inverting the bit. In the example of Fig. 2, all the data bits (first 16 columns) participate in exactly four parity bits ($2t$) and each pair of columns share at most one position with a value of one.

For an arbitrary value of $k = m^2$, the H matrix for a DEC OLS code is constructed as follows:

$$H = \begin{bmatrix} M_1 \\ M_2 \\ M_3 \\ M_4 \end{bmatrix} I_{4m} \tag{1}$$

where I_{4m} is the identity matrix of size $4m$ and M_1, M_2, M_3, and M_4 are the matrices of size $m \times m^2$ derived from OLS of size $m \times m$. Those matrices are illustrated in the example shown in Fig. 2. In a general case, for an OLS code that can correct t errors, the parity check matrix is constructed using $2t$ M_i matrices. The M_i matrices have only a one in each of its columns. Therefore, the first $k = m^2$ columns in H have a weight of $2t$. Additionally, as the Latin squares used to derive the M_i matrices are orthogonal, any pair of columns has at most a position with a one in common. This, as discussed before, enables the use of OS-MLD for decoding.

III. Proposed Method to Extend OLS Codes

The proposed method is based on the observation that by construction, the groups formed by the m parity bits in each M_i matrix have at most a one in every column of H. For the example in Fig. 2, those groups correspond to bits (or rows) 1–4 (M_1), 5–8 (M_2), 9–12 (M_3), and 13–16 (M_4). Therefore, any combination of four bits from one of those groups will at most share a one with the existing columns in H. 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 H 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.

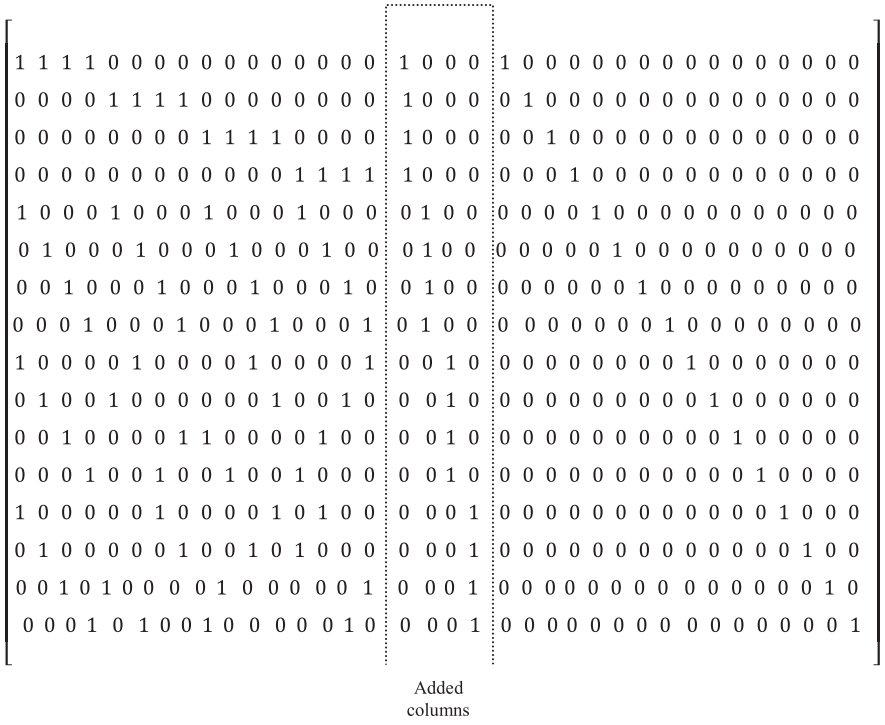

Fig. 3. Parity check matrix H for the extended OLS code with $k = 20$ and $t = 2$.

The proposed method first divides the parity check bits in groups of m bits given by the M_i 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. This is not an issue as by construction, different groups do not share any bit.

When m is small finding, such combinations is easy. For example, in the case considered in Fig. 3, 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. 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 $n - k = 2\ tm$ is the number of parity bits. The data block size for the original OLS codes (k_{OLS}) is also shown for reference.

The method can be applied to the general case of an OLS code with $k = m^2$ 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 $\binom{2t}{2}$ pairs in each combination. The number of possible pairs in each group of m bits is $\binom{m}{2}$. Therefore, the number of combinations N_G in a group of m bits has to be such that

$$\binom{m}{2} \geq \binom{2t}{2} \times N_G \tag{2}$$

which can be simplified as

$$\frac{m^2 - m}{4t^2 - 2t} \geq N_G. \tag{3}$$

For the examples in Table I, the above equality is met for $m = 4$ and $m = 16$ but not for $m = 8$. This shows that the upper bound will be achieved only in some cases. Because there are $2t$ groups, the total number of combinations (and therefore of data bits that can be added to the code) N_C can be bounded by

$$\frac{m^2 - m}{2t - 1} \geq N_C \tag{4}$$

which shows that the number of combinations, and therefore the benefits of the code extension decreases with t. That is, why the examples presented so far in this brief are for double ECCs. In Table II, the parameters of the extended codes when $t = 3$ are summarized. In this case, there is no possibility to extend a code with $m = 4$ as combinations of $2t = 6$ bits are needed. When $m = 8$, one combination can be added to each group. Finally, for $m = 16$, three combinations are possible. In any case, it can be observed that, as expected, for these triple error correction (TEC) codes, the proposed method provides smaller benefits than for DEC codes.

TABLE I
PARAMETERS OF THE DEC OLS EXTENDED CODES

k_{OLS}	$k_{Extended}$	n-k	m
16	20	16	4
64	72	32	8
256	336	64	16

TABLE II
PARAMETERS OF THE TEC OLS EXTENDED CODES

k_{OLS}	$k_{Extended}$	n-k	m
64	70	48	8
256	274	96	16

TABLE III
AREA ESTIMATES (in μm^2)

OLS codes			Extended codes		
k	Encoder	Decoder	k	Encoder	Decoder
16	299	921	20	382	1097
64	1198	3928	72	1458	4220
256	4613	10721	336	6277	14354

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.

One particular case for which a simple solution can be found is when $m = 2t \times l$. In this case, an OLS code with a smaller data block size (l^2) can be used in each group. One example for $t = 2$ is when $m = 16$ ($k = 256$) for which the OLS code with block size $k = 4^2$ can be used in each group as explained before. Similarly, for $t = 2$, when k =1024 ($m = 32$) the OLS code of size $k = 8^2$ can be used in each group.

From Tables I and II, 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 [16]. 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. As an example, consider a CAM used for traffic classification with 64-bit entries and an associated type of service field of 8 bits. In this case, the extended DEC code with $m = 8$ can be used to protect both the CAM entry and the type of service field. These examples show how the extended codes can be used to protect both the data stored in the memory and the associated flags or tags. Finally, the code with smallest block size (20 data bits) can also be useful to protect groups of flip-flops in digital circuits.

IV. EVALUATION

For the parameters shown in Table I, the proposed DEC extended codes have been implemented in MATLAB. Then, their error correction capabilities have been verified using random errors. Because the codes are double ECCs, one or two bit errors have been inserted randomly. One million random combinations were tested and the errors were corrected in all the cases.

TABLE IV
DELAY ESTIMATES (in nanoseconds)

OLS codes			Extended codes		
k	Encoder	Decoder	k	Encoder	Decoder
16	0.17	0.36	20	0.20	0.38
64	0.24	0.43	72	0.25	0.44
256	0.30	0.49	336	0.33	0.54

TABLE V
PER BIT AREA ESTIMATES (in μm^2)

OLS codes			Extended codes		
k	Encoder	Decoder	k	Encoder	Decoder
16	18.69	57.56	20	19.10	54.85
64	18.72	61.38	72	20.25	58.61
256	18.01	41.88	336	18.68	42.72

The encoders and decoders have also been implemented in HDL and synthesized for the 45-nm Oklahoma State University (OSU) FreePDK standard cell library [17] using Synopsys design compiler. The synthesizer was configured to optimize the delay. Therefore, the results provide the lowest delay that can be achieved. The reported circuit area could be reduced at the expense of increasing the delay. The area and delay results are presented in Tables III and IV. They are also compared with those of the original OLS codes. It can be observed that the area is larger than for the OLS codes. When the area is, however, divided by the number of data bits k, the area is roughly the same. This is shown in Table V, where the area/data bit is shown.

The delay for the proposed codes is slightly higher than for the original OLS codes. This increase is also due to the increase in data block size. Larger blocks mean that each parity check has more bits and therefore more delay. In particular for $m = 4$, the number of bits that participate in the extended code is five compared with four in the original OLS code. For $m = 8$, the number increases from eight to nine and finally for $m = 16$, the number grows from 16 in the original OLS code to 21 in the extended codes. Those increases are moderate and inline with the results observed in the synthesized circuits.

In summary, the results confirm that the area and delay of the proposed codes are similar to that of the original OLS codes. The increases observed are mostly due to the larger block size of the extended codes.

V. CONCLUSION

In this brief, a method to extend OLS codes has been proposed. The method has been used to derive extended double ECCs for different block sizes. 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.

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. The use of the method for codes that can correct more than two 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.

REFERENCES

[1] 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.

[2] 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.

[3] 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.

[4] 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.

[5] 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.

[6] S. Lin and D. J. Costello, *Error Control Coding*, 2nd ed. Englewood Cliffs, NJ, USA: Prentice-Hall, 2004.

[7] S. Ghosh and P. D. Lincoln, "Dynamic low-density parity check codes for fault-tolerant nano-scale memory," in *Proc. Found. Nanosci. FNAN*, 2007.

[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] 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.

[10] 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.

[11] 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.

[12] 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.

[13] 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.

[14] 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.

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

[16] C. McNairy and D. Soltis, "Itanium 2 processor microarchitecture," *IEEE Micro*, vol. 23 no. 2, pp. 44–55, Mar./Apr. 2003.

[17] 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. 173–174.

Improved Analytical Delay Models for RC-Coupled Interconnects

Feng Shi, Xuebin Wu, and Zhiyuan Yan

***Abstract*— As process technologies scale into deep submicrometer region, crosstalk delay is becoming increasingly severe, especially for global on-chip buses. To cope with this problem, accurate delay models of coupled interconnects are needed. In particular, delay models based on analytical approaches are desirable, because they are not only largely transparent to technology, but also explicitly establish the connections between delays of coupled interconnects and transition patterns, thereby enabling crosstalk alleviating techniques such as crosstalk avoidance codes. Unfortunately, existing analytical delay models, such as the widely cited model in [1], have limited accuracy and do not account for loading capacitance. In this brief, we propose analytical delay models for coupled interconnects that address these disadvantages.**

***Index Terms*— Bus, crosstalk, delay, interconnect.**

I. INTRODUCTION

Crosstalk caused by coupling capacitance between adjacent wires leads to additional delay to multiwire buses. As the process technologies scale into deep submicrometer region, coupling capacitance between adjacent wires and hence crosstalk delays increase greatly. According to the International Technology Roadmap of Semiconductors [2], gate delay **decreases** with scaling, while global wire delay **increases**. Hence, the crosstalk delay problem is becoming increasingly severe in VLSI designs, especially for global on-chip buses, and will become the performance bottleneck in many high-performance VLSI designs.

This brief focuses on analytical delay models applicable to general *RC*-coupled interconnects. Although various delay models of interconnects have been proposed in the literature [1], [3]–[7], few are comparable with our work in this brief. Some delay models [3], [7] do not consider crosstalk from adjacent wires. In addition, most previously proposed delay models are based on numerical approaches [3], [5]–[7]. They can achieve high accuracy, but they have several drawbacks, such as bulky lookup tables, dependence on technology, poor portability, and high complexity. A widely cited analytical delay model proposed in [1] and [4], which uses the similar methodology to that in [8], appears to be the most comparable previous delay model to our work in this brief.

Based on the model in [1] and [4], the delay of the kth wire ($k \in \{1, 2, \cdots, m\}$) of an m-bit bus is given by

$$T_k = \begin{cases} \tau_0[(1+\lambda)\Delta_1^2 - \lambda\Delta_1\Delta_2], & k = 1 \\ \tau_0[(1+2\lambda)\Delta_k^2 - \lambda\Delta_k(\Delta_{k-1} + \Delta_{k+1})], & k \neq 1, m \\ \tau_0[(1+\lambda)\Delta_m^2 - \lambda\Delta_m\Delta_{m-1}], & k = m \end{cases} \quad (1)$$

where λ is the ratio of the coupling capacitance between adjacent wires and the ground capacitance of each wire, τ_0 is the intrinsic delay of a transition on a single wire, and Δ_k is 1 for $0 \rightarrow 1$ transition, –1 for $1 \rightarrow 0$ transition, or 0 for no transition on the kth wire. We observe that in this model, the delay of the kth wire

Manuscript received March 20, 2012; revised January 29, 2013 and April 8, 2013; accepted July 25, 2013. Date of publication August 22, 2013; date of current version June 23, 2014.

F. Shi and Z. Yan are with the Department of Electrical and Computer Engineering, Lehigh University, Bethlehem, PA 18015 USA (e-mail: fes209@lehigh.edu; yan@lehigh.edu).

X. Wu is with LSI Corporation, Milpitas, CA 95035 USA (e-mail: xuebin.wu@lsi.com).

Digital Object Identifier 10.1109/TVLSI.2013.2275071