Error Correcting Code Analysis for Cache Memory High Reliability and Performance*

Daniele Rossi
DEIS – U. of Bologna (Italy)
d.rossi@unibo.it

Nicola Timoncini
DEIS – U. of Bologna (Italy)
n.timoncini@studio.unibo.it

Michael Spica
Cypress Semiconductor (USA)
msr@cypress.com

Cecilia Metra
DEIS – U. of Bologna (Italy)
cecilia.metra@unibo.it

***Abstract*— In this paper we address the issue of improving ECC correction ability beyond that provided by the standard SEC/DED Hsiao code. We analyze the impact of the standard SEC/DED Hsiao ECC and for several double error correcting (DEC) codes on area overhead and cache memory access time for different codeword sizes and code-segment sizes, as well as their correction ability as a function of codeword/code-segment sizes. We show the different trade-offs that can be achieved in terms of impact on area overhead, performance and correction ability, thus giving insight to designers for the selection of the optimal ECC and codeword organization/code-segment size for a given application.**

I. INTRODUCTION

The continuous scaling of microelectronic technology enables to keep on increasing system complexity, that will soon exceed billions of transistors integrated on the same die. However, since this growth comes together with the reduction in power supply and, consequently, noise margins, the vulnerability of electronic systems to radiation induced errors (namely soft errors, or Single Event Upsets (SEUs)) is expected to increase considerably as well. Particularly, SEUs affecting caches are considered a major threat to the reliability of next generation microprocessors. This due to the increased likelihood of SEUs with technology scaling, as well as to the continuous increase in cache size to improve microprocessor performance.

Several error detecting codes (EDCs) and error correcting codes (ECCs) have been proposed so far to improve cache reliability. They range from the simple parity check code (used, for instance, to protect the L1 caches of the Itanium [1], and the L1 and L2 caches of the Power4 [2] microprocessors), to the more complex Single Error Correcting/Double Error Detecting (SEC/DED) ECC [3] (used to protect the L2 and L3 caches in the Itanium microprocessor [1].

However, in the perspective of the continuous scaling of microelectronic technology, and the increase in cache size, the problem to extend the correction ability of the cache ECCs is arising. In fact, with increasingly higher density memory arrays and more scaled down technologies, the probability of SEUs will increase, and that of multiple bit upsets (MBUs) will no longer be negligible [4]. Currently, MBUs are tackled by memory interleaving [5, 6], that is by logically mapping physically adjacent memory cells into different memory logical words. This way, a clustered error affecting two or more adjacent cells manifests itself as a single error affecting two or more different memory words, thus being successfully corrected by a SEC/DED code. However, interleaving generally requires a more complex (thus more expensive) decoding circuitry, and can not guarantee error correction in case of two errors affecting the same memory word. Moreover, it has been proven that a single MBU, or an SEU followed by an MBU (or vice-versa), or a combination of MBUs may lead to double bit errors in the same logical word [7, 8].

Therefore, extending the correction ability of the ECCs to be employed to protect caches of next generation microprocessors is still an open issue. The problem is that, while powerful ECCs do exist, they usually imply high area overhead and non negligible impact on performance, mainly due to the higher number of check bits to be stored, and to the more complex encoding/decoding procedure, respectively, compared to standard SEC/DED codes. Of course, the optimal solution depends on the considered memory architecture (high level caches may tolerate higher area and performance penalties than low level caches), and the considered application (high reliability applications may favor error protection towards cost reduction, while the opposite applies for general purpose applications).

On principle, a simple way to increase the number of errors within a memory array that can be corrected by an ECC, would be to reduce the codeword length. In fact, this way the number of codewords stored in a memory array of a given capacity increases and, for a given ECC detection/correction ability, the number of correctable errors increases as well. Similarly, a memory word could be segmented [9], thus considering each segment as a different (shorter) codeword to be encoded/decoded in parallel.

However, shorter codewords/segments imply a higher number of check bits to be stored, thus a reduction of the useful memory, that is the memory portion storing data bits. For shorter codewords, such a drawback may be partially counterbalanced by the smaller area required by the ECC

*Partially supported by Intel Corporation and by the Italian Ministry of University and Research (MIUR) under PRIN Project n. 2008K4P7X9

978-3-9810801-7-9/DATE11/©2011 EDAA

encoding/decoding circuitry. Instead, this is not the case for code segmentation, since all segments are encoded/decoded in parallel, thus mandating an encoding/decoding circuit for each segment, with consequent increase in area. Such an increase in area is not linear with the number of segments. For instance, doubling the number of segments does not double the area overhead due to the ECC circuitry.

It clearly appears from the above considerations that evaluating the impact on area overhead and performance of different ECCs with different codeword organization (different codeword size, code segmentation, etc.) is not a straightforward task. Similarly, for the impact on performance. Based on these considerations, in this paper we address the issue of improving ECC correction ability beyond that provided by the standard SEC/DED Hsiao code, and provide a systematic evaluation of the different cost-reliability tradeoffs that can be achieved by different ECCs, with different codeword organization.

To perform our evaluation, we have developed an *ad hoc* software tool allows the user to analyze and compare area overhead, delay and correction ability of several ECCs, each with different codeword length and/or code-segment size. Our tool is scalable in codeword length and code-segment size, as well as considered ECC.

We will present the results obtained for the standard SEC/DED Hsiao ECC and for several double error correcting (DEC) codes. We will show the different trade-offs that can be achieved in terms of impact on area overhead, performance and correction ability, thus giving insight to designers for the selection of the optimal combination of ECC and codeword organization/code-segment size, for a target cost-correction ability tradeoff.

This paper is organized as follows. In Section 2, we describe our developed software tool for ECC evaluation and the considered metrics. In Section 3, we first show its application to the standard SEC/DED Hsiao ECC and to several DEC codes, and compare the achieved results. Finally, in Section 4, we give some conclusive remarks.

II. Developed Software Tool and Metrics for ECC Evaluation

In order to analyze and compare different ECCs, we have developed an *ad hoc* software tool that, for a given (logical) codeword length, is able to: i) generate all codewords of a chosen ECC; ii) generate the ECC encoding/decoding circuitries, described in VHDL language, given its matricial or polynomial description; iii) evaluate and compare different ECCs, considering the required area overhead, delay and correction ability, according to the metrics defined later in this section.

The platform that has been considered for such a software tool is Visual Studio 2005 (working on Microsoft Windows Operating System). As for the ECC generation and evaluation, the software has been implemented in the C++ standard language. As for the graphic interface, the software has been implemented in Visual C++.NET.

Let us now introduce the metrics that we employed for our ECC analysis. They are based on the evaluation of the area overhead and delay due to the ECC encoding/decoding circuitry in terms of equivalent gates, assuming a 2-input NAND gate as the equivalent gate unit. Table 1 reports the derived area and delay figures of various 2-input gates considered for our evaluation, assuming their static implementation by a standard CMOS technology.

Table 1 Area and delay figures expressed in terms of equivalent gates.

Gate	Area (*A*)	Delay (*D*)
Not	0.5	0.5
Nand	1	1
Nor	1	1
And	1.5	1.5
Or	1.5	1.5
Xor	2	1.75
Xnor	2	1.75
Memory cell	2	na

The area A and delay D of an n-input gate g (denoted by $X_g(n)$ with X = A, D) has been expressed as a function of that of a 2-input gate g as follows:

$$X_g(n) = \frac{n}{2} X_g(2), \text{ with } X=A, D. \quad (2)$$

The metrics that we have employed for our ECC analysis are defined in the remainder of this section.

1) The ***Area Overhead*** (*AO*) due to the implementation of the ECC is given by:

AO = (*Area ECC) / (Area Data)*,

where *Area ECC* is the area occupied by the ECC circuitry (A_{cir}) plus the area of the portion of the memory array employed to store the check bits (A_{check}), while *Area Data* (A_{data}) is the portion of the memory array required to store the (useful) data bits.

The area overhead can therefore be written as:

$$AO = \frac{A_{check} + A_{cir}}{A_{data}}. \quad (3)$$

In order to evaluate the area of the portion of the memory array devoted to store data and check bits, let us first denote the total memory array capacity by *MAC*. If *CL* indicates the codeword length, the number of codewords N_W that can be stored in the memory is given by: $N_W = \lfloor MAC/CL \rfloor$. Denoting by a_{cell} the area of a memory cell, the area of the whole memory array can be written as $A_{mem}=MAC\ a_{cell}$,

while the effective area of the memory array, that is the area actually utilized to store codewords, is:

$$A_{mem\text{-}eff} = N_W \, CL \, a_{cell}. \quad (4)$$

Finally, the area of the memory array portion that stores check bits is given by:

$$A_{check} = m \, N_W \, a_{cell} = A_{mem\text{-}eff} \rho \, a_{cell}, \quad (5)$$

where m denotes the number of check bits per codeword, and ρ the ECC redundancy.

2) The ***Delay*** introduced by the ECC encoding/decoding circuitry has been evaluated considering the delay of its longest path that is activated during a read/write operation, that is during the decoding/encoding procedure.

In this regard, it should be noted that the decoding procedure, and in particular the phase devoted to error localization, is the more time consuming. Therefore, the delay of the ECC has been evaluated considering only the decoding circuit. In particular, the tool identifies the longest path of the ECC decoding circuit, and evaluates its delay in terms of number of gate delay levels, according to Eq. (2).

3) The ***Correction ability*** (*CA*) of an ECC has been defined as the total number of detectable/correctable errors within a memory codeword.

In order to compute the *CA*, we have assumed that errors are uniformly distributed throughout the memory codewords, as it may likely be the case for SEUs.

Actually, as introduced in Sect. 1, in dense semiconductor memories, errors can involve a cluster of cells, as it is the case for MBUs [4, 7]. In order to allow standard ECCs (like, for instance, the Hsiao code) to cope with physically clustered errors, interleaving is usually adopted [5, 6, 8, 10]. This way, memory cells that are physically adjacent are logically mapped to different codewords. Clustered errors are therefore disaggregated into single errors, thus being possibly corrected by a SEC (or SEC/DED) code. However, the application of a SEC code with interleaving does not guarantee error correction in case of two errors affecting the same logical word which, as discussed in the Introduction, has been proven to be a possible event [7, 8].

The correction ability (*CA*) has been defined as reported in Eq. (6), that is as the weighted mean (with weights α_d and α_c) of the probability to detect up to j_d errors (p_d), and correct up to j_c errors (p_c), considering the probability $p_e(i)$ to have i errors (for $i = 1 \,..\, j_{d,c}$) affecting the same codeword.

$$CA = \frac{\alpha_d \sum_{i=1}^{j_d} p_d(i) \, p_e(i) + \alpha_c \sum_{i=1}^{j_c} p_c(i) \, p_e(i)}{\alpha_d \sum_{i=1}^{j_d} p_e(i) + \alpha_c \sum_{i=1}^{jc} p_e(i)}. \quad (6)$$

The weights α_d and α_c, and the numbers j_d and j_c can be configured by the user. The coefficients α_d and α_c allow to weight differently the probability to correct errors (p_c), or to detect them only (p_d). As for p_d and p_c, they have to fulfill the following conditions: i) $0 \le p_c(j) \le p_d(j) \le 1$, $\forall j$; ii) $p_d(0) = p_c(0) = 1$; iii) $p_d(1) = p_c(1) = 1$.

Particularly, condition i) derives from the fact that, by definition, probability values range from 0 to 1. Furthermore, the detection probability is always higher than the correction probability, since to correct an error we first have to detect it. As an example, considering a SEC/DED code, it is:

$$p_d(i) = \begin{cases} 1, i \le 2 \\ 0, i > 2 \end{cases} \; ; \; p_c(i) = \begin{cases} 1, i \le 1 \\ 0, i > 1 \end{cases} \quad (7)$$

Condition ii) derives from the fact that, if no error has occurred (so that $i = 0$), the ECC circuitry must indicate that the codeword is error free, and must give the correct word to the output.

Analogously, property iii) derives from the fact that our considered ECCs, which are at least single error correcting, are able to handle properly a single error (i.e., they provide the proper indication).

As for the error probability p_e, it has been expressed it as the number of expected errors (*ne*) per kB of memory. As an example, we have assumed *ne* = 1). However, the value of *ne* is an input to our tool, to be specified by the user.

III. ECC Analysis and Comparison

A. SEC/DED Code Analysis

We have considered the SEC/DED Hsiao code [3] as the reference code. In fact, thanks to its limited impact on cache area and performance, it is the *de facto* standard. This is due to its requiring a minimum number of ones in the parity check matrix, thus allowing the implementation of the ECC encoding/decoding circuitry by the minimum number of XOR gates. The constraints to compose the parity check matrix H are the following: i) all columns are different from 0; ii) each column is different from the others (thus allowing to identify and correct all single errors [3]); iii) each column has an odd number of ones (thus allowing to detect, but not correct, all double errors [3]).

As shown later, its area overhead varies with the size of the memory array. Therefore, as an example, for our analysis we have considered two different memory sizes: 32 kB and 6MB, which correspond to the size of the L1 cache (16 kB data + 16 kB instruction) and the L3 cache of the Intel Itanium 2 9010, respectively.

1) Codeword Size Analysis

We have considered different codeword sizes, with data bits ranging from 8 to 2048. As expected, as the code efficiency (that is the ratio between the number of data bits in a codeword and the whole codeword length) increases, also the area overhead and delay due to the ECC encoding/decoding circuitry increase.

Tab. 2 reports the values generated by our tool, for delay, area overhead and correction ability, when the codeword size is changed. As for delay, the values are in terms of gate delay levels computed as in Eq. (2); as for area overhead, the values are computed according to Eq. (3), and are expressed as a percentage; finally, correction ability has been calculated as shown in Eq. (6), and its value has been reported as a percentage.

We can notice that the ECC encoding/decoding circuitry delay slightly increases with the codeword length (CL) increase. This is due to the fact that, if CL increases, the complexity of the ECC circuit increases as well, thus implying a higher delay. This holds true independently of the memory array size.

Tab. 1. Hsiao SEC/DED code: Code Size Analysis.

Data Bits	Check Bits	CA (%)	Delay (EqG)	AO (%)	
				32kB	6MB
8	5	74.2	13.5	62,55%	62,50%
16	6	73.8	16.0	37,59%	37,50%
32	7	72.6	18.5	22,04%	21,88%
64	8	70.1	19.5	12,82%	12,50%
128	9	67.7	23.0	7,71%	7,03%
256	10	57.9	25.5	5,35%	3,91%
512	11	44.8	28.0	5,20%	2,16%
1024	12	27.6	29.5	7,60%	1,21%
2048	13	10.4	35.0	15,00%	0,71%

As for area overhead, different considerations can be made for the two considered memory array sizes.

For the 32kB array area overhead initially decreases with the increase in CL, then reaches a minimum for a word size approximately equal to 256 bits, to then start increasing. This is due to the fact that, for small CLs, as CL increases, the area overhead due to check bits diminishes considerably (due to their lower number), and this reduction exceeds the area overhead increase due to the more complex ECC encoding/decoding circuitry. Overall, we have an area overhead decrease. For long CLs (larger than 256 bits), instead, the significant increase in area overhead due to the considerably more complex ECC circuitry exceeds the reduction in area overhead due to the lower number of the total check bits, leading to an overall area overhead increase.

For the 6MB array, area overhead decreases monotonically with longer codewords. In fact, if CL increases, the memory array contains fewer codewords, thus a lower number of total check bits. Therefore, the area overhead due to check bits decreases. Moreover, longer codewords imply a more complex ECC encoding/decoding circuitry, but its area overhead increase is smaller compared to the previously discussed decrease due to fewer check bits. The area overhead decrease reduces its pace for larger CLs. This because, for large CLs, the impact of the ECC encoding/decoding circuitry complexity on area overhead increases. If CL increase further, we can expect a non-monotonic behavior qualitatively similar to that found for the 32kB array.

Finally, as for the correction ability (CA), as expected, it decreases with the increase in the CL. In fact, with longer codewords, the probability to have multiple errors in the same codeword increases, leading to a reduction of CA.

2) Partition Size Analysis

As previously introduced, in order to increase the correction ability, ECC segmentation can be adopted. Particularly, a read-out memory word can be partitioned in two or more "sub-words" (code-segments), and each of this sub-words can be decoded at the same time by means of a dedicated ECC circuitry, that is different ECC circuits can operate in parallel to encode/decode different sub-words. Analogously to the case of smaller codeword sizes, this approach requires an increase in the number of stored check bits. Moreover, it requires also extra ECC circuitry, due to the dedicated ECC circuitry for each code-segment.

Now let us investigate how memory word partition impacts the ECC metrics defined in Sect. 2. We have considered the case of the Hsiao code with a 2048-bit memory word segmented (partitioned) in sub-words, whose length ranges from 8 data bits (256 partitions) to 2048 data bits (1 partition). Tab. 2 reports the values obtained by our tool, when the partition size analysis is performed.

Tab. 2. Hsiao SEC/DED code: Partition Size Analysis.

Data Bits	Check Bits	CA (%)	Delay (EqG)	AO (%)	
				32kB	6MB
8	5	74.2	13.5	76,07%	62,57%
16	6	73.8	16.0	49,22%	37,56%
32	7	72.6	18.5	32,50%	21,93%
64	8	70.1	19.5	23,00%	12,55%
128	9	67.7	23.0	18,04%	7,09%
256	10	57.9	25.5	15,52%	3,97%
512	11	44.8	28.0	14,38%	2,21%
1024	12	27.6	29.5	14,02%	1,24%
2048	13	10.4	35.0	14,74%	0,71%

Similarly to the case of the codeword size analysis previously described, the delay of the ECC encoding/ decoding circuitry increases for larger partition sizes. It should be noted that, for small partition sizes, the longest path for which the delay has been evaluated is represented by the additional logic that collects the double error detection (DED) signals from all the ECC circuits (one for each partition), in order to generate a single DED signal.

Also for area overhead, similar qualitative behaviors as those highlighted by the codeword size analysis have been obtained for both the considered memory array sizes. Particularly, the curve relative to the 32kB memory array is non-monotonic, with the minimum reached for 512-1024 partition size. The area overheads for both memory arrays are higher than those highlighted in Tab. 1, since in the partition size analysis we must account for one ECC circuit for each partition (code-segment).

Finally, the correction ability has the same values as in Tab. 1, since the number of detectable/correctable errors in a codeword or code-segment of a given length is the same. Therefore, analogous considerations hold true.

B. DEC Codes' Analysis and Comparison

We have analyzed several DEC codes and compared the achieved results with those obtained for the reference Hsiao code. As an example, we have considered three DEC codes able to correct two random errors within a codeword or code-segment. As an example, we have considered the sum code obtained by combining Hsiao SEC/DED code and a single parity check (SPC) (HSIAO-SPC) code, built as reported in [11], the Origuchi-Morita code [12], and the Cross-Parity Check (CPC) code, that we have obtained by modifying the Origuchi-Morita code in order to reduce its impact on performance. Moreover, it has been considered also the Single byte Error Correcting (SbEC) Bossen code [13]. All the selected DEC codes can be decoded by fast combinational circuits, which is a mandatory property for being implemented in fast cache memories.

In the sum Hsiao-SPC code, the bits of a memory codeword are logically arranged as a matrix. For each matrix row, a Hsiao SEC/DED code is computed, while, for each column, a parity bit is determined. All single errors are detected and corrected by the Hsiao code, while the correction of double errors is performed into two steps: as a first step, all codewords containing double random errors are identified by the Hsiao code; as a second step, the two erroneous bits within the selected codeword are localized by the column parities, and then corrected.

Also for the Origuchi-Morita code, the bits of a memory codeword are logically arranged as matrix. In this case, however, the matrix has to be squared, and should have an odd number of rows/columns [12]. Row (horizontal), column (vertical), secondary diagonals and overall parities are computed. Correction is performed by majority voting over a properly selected sub-set of syndrome bits.

The CPC code construction is similar to that of the Origuchi-Morita code, with the difference that the calculation of the overall parity, which can require a considerable latency, is replaced by the calculation of the parities obtained from the primary matrix diagonals. The reduction of the latency is however counterbalanced by an increase in the check bit redundancy. Similarly to the case of the Origuchi-Morita code, correction is performed by majority voting over a properly selected sub-set of syndrome bits.

Finally, as for the considered Bossen SbEC code [13], which is able to correct single 2-bit clustered errors, it has been implemented by the scheme shown in [14].

Let us now go through the results obtained by our tool for the partition size analysis. As previously discussed, these results are qualitatively similar to those of the codeword size analysis.

In Fig. 1 we report the results obtained for the delay analysis of the considered ECCs.

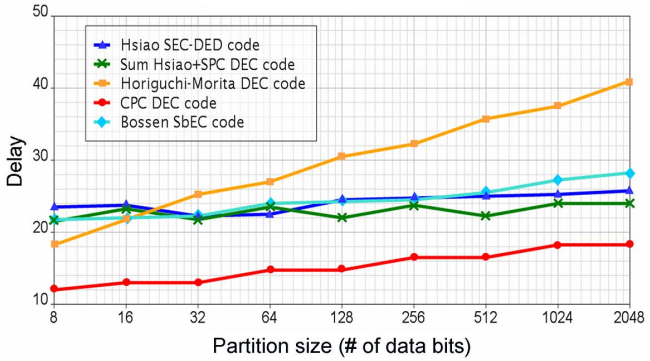

Fig. 1. Considered ECCs' delay comparison.

As can be seen, the Origuchi-Morita code is the worst one for partition size larger than 32 bits. Sum Hsiao-SPC, Hsiao SEC/DED and Bossen codes introduce a similar latency for all partition sizes. As for the CPC code, it presents a delay considerably lower than all other codes. To summarize, sum Hsiao-SPC and CPC codes, which are DEC codes, may show a lower impact on memory access time than the Hsiao code, which is only SEC/DED code. Moreover, the CPC code implies a delay which is approximately a half of that due to the Hsiao SEC/DED code for 8-bit partitions, and which is lower of more than 30% for all other partition sizes.

Fig. 2 reports the plots depicting the behavior of area overheads as a function of partition size, obtained by our tool for the 32kB memory array.

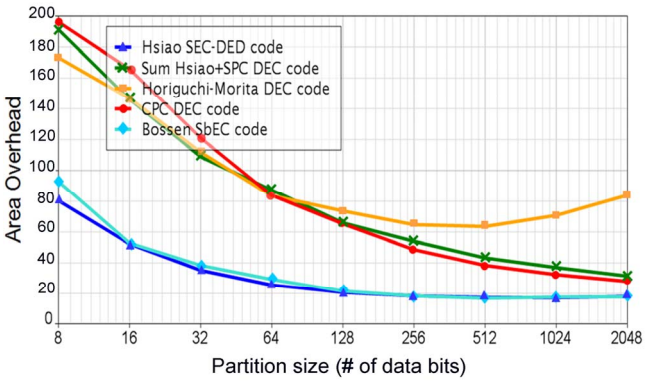

Fig. 2. Considered ECCs' area overhead comparison.

Hsiao SEC/DED and Bossen codes imply a very similar area overhead for all partition sizes, while all other considered ECCs introduce a higher area overhead for all partition sizes. Among these latter, the Origuchi-Morita code is the one requiring the higher overhead for partition sizes larger than 64 bits due to its encoding/decoding circuitry, whose complexity increases considerably for large partition sizes. The area overhead due to the DEC Hsiao-SPC and CPC codes may be twice the overhead of the Hsiao SEC/DED code, but it reduces to approximately +30% for 2048-bit partitions.

Finally, as for correction ability, the plots shown in Fig. 3 have been obtained considering, as an example, an error occurrence equal to 1 error per kB of memory.

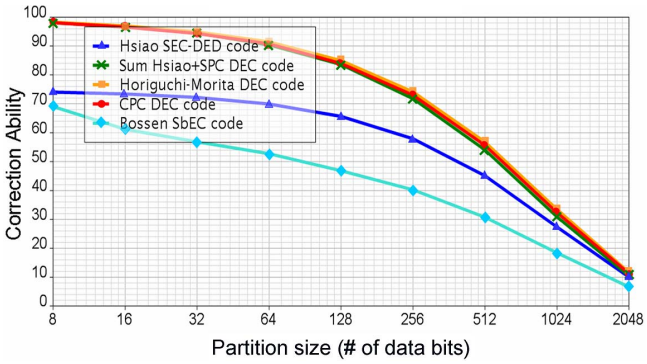

Fig. 3. Considered ECCs' correction ability comparison.

It can be noticed that, for all DEC codes, the correction ability approaches 100% for small partition sizes. Moreover, for all partition sizes, the correction ability of the Bossen code is even lower that that of the Hsiao code, since the probability that multiple errors belong to the same symbol, thus being correctable by the Bossen code, is rather low and decreases with the increase in the codeword length.

The difference between the correction ability of the Hsiao SEC/DED code and that of the DEC codes decreases with the increase in partition size.

This is due to the fact that, for large partition sizes, the probability to have a number of errors in the same codeword exceeding the ECC correction ability is rather high for all considered ECCs.

IV. Conclusions

We have analyzed the impact of several SEC/DED and DEC codes on area overhead and memory access time for different codeword sizes and code-segment sizes, as well as their correction ability as a function of codeword/code-segment sizes. We have shown the different trade-offs that can be achieved in terms of impact on area overhead, performance and correction ability, thus giving precious hints to designers for the selection of the optimal combination of ECC and codeword organization/code-segment size for a considered application.

References

[1] H. Sharangpani, H. Arora, "Itanium processor microarchitecture", *IEEE Micro*, vol. 20 , no. 5, Sept.-Oct. 2000 , pp. 24 – 43.

[2] J. M. Tendler, J. S. Dodson, J. S. Fields, Jr., H. Le, B. Sinharoy, "POWER4 system microarchitecture", *IBM Journal of Research and Development*, vol. 46, no. 1, Jan. 2002, pp. 5-25.

[3] M. Y. Hsiao, "Class of Optimal Minimum odd-Weight-Column SEC-DED Codes", *IBM Journal of Research and Development*, vol. 14, no. 4, July 1970, pp. 395-401.

[4] J. Maiz, S. Hareland, K. Zhang, P. Armstrong, "Characterization of Multi-bit Soft Error Events in Advanced SRAMs 130nm", in *Proc. of IEEE Int'l Electron Devices Meeting*, 2003, pp. 21.4.1 - 21.4.4.

[5] M. Blaum, J. Bruck, A. Vardy, "Interleaving Schemes for Multidimensional Cluster Errors", *IEEE Trans. on Information Theory*, vol. 44, no. 2, Mar. 1998, pp. 730-743.

[6] Y. Q. Shi, X. M. Zhang, Z.-C. Ni, N. Ansari, "Interleaving for Combating Burst of Errors ", *IEEE Circuist and Systems Magazine*, vol. 4, no. 1, 2004, pp. 29-42.

[7] D. Radaelli, H. Puchner, S. Wong, S. Daniel, "Investigation of Multi-Bit Upsets in a 150 nm Technology SRAM Device", *IEEE Trans. on Nuclear Science*, vol. 52, no. 6, Dec. 2005, pp. 2433-2437.

[8] P. Reviriego, J. A. Maestro, C. Cervantes, "Reliability Analysis of Memories Suffering Multiple Bit Upsets", IEEE Trans. on Device and Materials Reliability, vol. 7, no. 4, Dec. 2007, pp. 592 – 601.

[9] Y.-H. Kwon, M.-K. Oh, D.-J. Park, *"A New LDPC Decoding Algorithm Aided by Segmented Cyclic Redundancy Checks for Magnetic Recording Channels", IEEE Trans. on Magnetic, vol. 41, no. 7, July 2005, pp. 2318-2320.*

[10] C.W. Slayman,"Cache and Memory Error Detection, Correction, and Reduction Techniques for Terrestrial Servers and Workstations", *IEEE Trans. on Device and Materials Reliability*, vol. 5 , no. 3, September 2005, pp. 397-404.

[11] T. Fuja, C. Heegard, R. Goodman, "Linear Sum Codes for Random Access Memories", *IEEE Trans. on Computers*, vol. 37, no. 9, Sept. 1988, pp. 1030-1042.

[12] T. Horiguchi, K. Morita, "A parallel Memory with Double Error Correction Capability", *Paper of Technical Group EC 75-42, IECE* Japan, Nov. 1975.

[13] D. C. Bossen, "b-Adjacent Error Correction", IBM Journal of Research and Development, vol. 14, no. 4, July 1970.

[14] T.R.N. Rao, E. Fujiwara, *Error Control Coding for Computer Systems*, Prentice Hall: Englewood Cliffs, NJ, 1989.