Low Power embedded DRAM Caches using BCH code Partitioning

Pedro Reviriego, *Member IEEE,* Alfonso Sánchez-Macian, *Member, IEEE,* and Juan Antonio Maestro, *Member, IEEE*

***Abstract*— Technology advances have recently enabled the use of DRAMs into logic integrated circuits. These embedded DRAMs can be used to efficiently implement caches since DRAMs require substantially less area than SRAMs. One challenge for DRAM based caches is that a small time between refreshes is needed to ensure data retention. These refreshes increase the power consumption even when the cache is idle. To mitigate this issue, the use of longer times between refreshes combined with the use of Error Correction Codes (ECCs) has been recently proposed. The idea is that the time between refreshes can be increased significantly while only causing data retention failures on a small percentage of the cells. Then those errors can be corrected by the ECC. For this scheme to be efficient the number of additional bits required by the ECC should be small. This is achieved by using large data blocks for the ECC which in turns means that a large data block has to be accessed even when only a small portion of it is needed. This has no effect on idle power consumption but increases the dynamic power consumption and reduces the effective memory bandwidth. In this paper, a technique to mitigate this issue is proposed. It enables better granularity in the read data accesses by partitioning the ECC block into two sub-blocks and modifying the error detection and correction processes. This reduces the dynamic power consumption and increases the available memory bandwidth while requiring only a moderate increase in the number of additional bits.**

***Index Terms*— ECC, BCH codes, Caches, eDRAM, memory, Low power.**

I. Introduction

THE use of new process technologies enables the integration of embedded DRAM (eDRAM) on a logic process [1]. This is interesting as eDRAMs can be used to build on-chip caches that are much denser than their SRAM counterparts. As an example a 32MB on-chip eDRAM cache has been incorporated in some IBM processors [2].

A key challenge for eDRAMs is that the refresh time is much smaller than that of traditional DRAMs [3]. This substantially increases the power consumption of the memory when it is idle. To reduce the amount of refreshes, techniques that keep track of recent accesses have been proposed [4]. However those techniques provide no benefit when the memory is idle.

When the time between refreshes is increased, some memory cells will suffer data retention failures. The data retention time of the cells varies significantly due to process variations [5], [6]. It has been shown that the time between refreshes can be increased substantially while keeping the number of cells that suffer data retention failures low [5]. This observation has motivated the use of Error Correction Codes (ECCs) to correct the errors caused by the increased time between refreshes [6]. Alternative schemes to detect errors have also been proposed [7].

Error Correction Codes have been used for decades to protect memories from radiation induced soft errors [8],[9]. Typically per word Single Error Correction Double Error Detection (SEC-DED) codes are used to that end [10]. This is sufficient as the error rates for terrestrial applications are low. However, when the ECCs are used to correct errors caused by data retention failures in eDRAMs, the error rates can be much larger. This means that more advanced ECCs that can correct multiple bit errors are needed. For example, in [11] the use of Bose–Chaudhuri–Hocquenghem (BCH) codes that can correct multiple bit errors is proposed for eDRAM caches.

The use of more advanced ECCs also poses implementation challenges. For example, the number of additional redundant bits increases with the number of errors that a code can correct. The decoder complexity and latency also increase with the error correction capabilities of the code. Techniques to mitigate those issues were studied in [11] where a scheme named Hi-ECC was proposed.

In Hi-ECC the issue of the decoding latency is solved by first checking if there are errors. When there are no errors, the rest of the decoding stages are not needed. As most of the blocks will have no errors, the average decoding latency is low. This technique is complemented with a fast decoding for single errors and an option to disable the blocks that suffer multiple errors.

To reduce the number of additional bits, Hi-ECC uses large blocks (1KB versus 64B). This leverages the fact that the number of additional bits to achieve a given error correction capability in an ECCs grows less than the block size. In many cases, the number of additional bits is related to the logarithm

This work was supported by the Spanish Ministry of Science and Innovation under Grant AYA2009-13300-C03-01.

P. Reviriego, A. Sánchez-Macian and J.A. Maestro are with Universidad Antonio de Nebrija, C/Pirineos, 55, E-28040, Madrid, Spain, (phone: +34 914521100; fax: +34 914521110; email: {previrie, asanchep jmaestro}@nebrija.es).

of the block size. The use of large blocks has an important drawback: read and write operations involve more cells and therefore require more power and memory bandwidth. This means that to access a given sub-block of 64B a complete block of 1KB has to be accessed. For write operations, the ECC bits can be calculated using only the old and new sub-blocks and the old ECC bits such that there is no need to access the whole block. In addition, the encoding of BCH codes is much faster than the decoding. Therefore the main problem occurs for read operations. To overcome this issue, in Hi-ECC a table of recently accessed blocks is used. Since the contents of this table have been recently refreshed, there will not be data retention errors and there is no need to use the ECC for the block such that sub-blocks can be accessed directly.

In this paper, techniques to allow read operations of data units smaller than a block using BCH codes are presented. The proposed scheme preserves the multi bit error correction capabilities of the block ECC while allowing access to half of the block. This will reduce the dynamic power consumption associated with the use of large blocks. Also when a table of recently accessed data is used, its entries can be half the size enabling a better granularity that can provide better performance.

The rest of the paper is organized as follows, in Section II an overview of BCH codes is presented focusing on the aspects relevant to the proposed scheme. Then in section III, the new approach is presented in a general form and compared with the traditional use of BCH codes. Section IV presents a case study discussing in detail the proposed scheme for a given block size and error correction capability. Finally the conclusions of this work are presented in Section V that ends the paper.

II. BCH CODES

Bose–Chaudhuri–Hocquenghem (BCH) codes are cyclic codes for which a large number of block sizes and error correction capabilities are available [12]. The main parameters of BCH codes are summarized in the following equation:

$$\begin{aligned} &n = 2^m - 1 \\ &n - k \le m \cdot t \\ &d_{\min} \ge 2 \cdot t + 1 \end{aligned} \tag{1}$$

where n is the block size, k the number of data bits, n-k the number of additional bits added by the code, d_{min} the minimum distance of the code and t the number of errors that the code can correct.

BCH codes can be systematized such that the data bits are not modified in the encoding process and only the additional bits are added. This is needed to use the procedure in which the blocks are checked for errors first and only decoded when there are errors.

To check if there are errors, the parity check matrix H of the code can be used. The encoded data block is multiplied by the H matrix and an array is obtained. The array is known as the syndrome, if there are any non zero bits in the syndrome, the block contains errors. The H matrix for BCH codes has the following form:

$$H = \begin{bmatrix} 1 & \alpha & \alpha^2 & \alpha^3 & \ldots & \alpha^{n-1} \\ 1 & (\alpha)^3 & (\alpha^2)^3 & (\alpha^3)^3 & \ldots & (\alpha^{n-1})^3 \\ \ldots & \ldots & \ldots & \ldots & \ldots & \ldots \\ 1 & (\alpha)^{2t-1} & (\alpha^2)^{2t-1} & (\alpha^3)^{2t-1} & \ldots & (\alpha^{n-1})^{2t-1} \end{bmatrix} \tag{2}$$

where α is a primitive element in the Galois Field GF(2^m). It can be observed that the H matrix is constructed in such a way that the matrix for a code that can correct t errors contains the matrixes for the codes that can correct *1,2,..,t-1* errors. As an example, for $t=2$, $m=4$ and $n = 2^4-1=15$, the following H matrix is obtained:

$$H = \begin{bmatrix} 1&0&0&0&1&0&0&1&1&0&1&0&1&1&1 \\ 0&1&0&0&1&1&0&1&0&1&1&1&1&0&0 \\ 0&0&1&0&0&1&1&0&1&0&1&1&1&1&0 \\ 0&0&0&1&0&0&1&1&0&1&0&1&1&1&1 \\ 1&0&0&0&1&1&0&0&0&1&1&0&0&0&1 \\ 0&0&0&1&1&0&0&0&1&1&0&0&0&1&1 \\ 0&0&1&0&1&0&0&1&0&1&0&0&1&0&1 \\ 0&1&1&1&1&0&1&1&1&1&0&1&1&1&1 \end{bmatrix} \tag{3}$$

In this case, the values over the GF(2^4) are represented by four bits in binary form such that α={0100}. Then, the first four rows of this matrix are precisely the H matrix for the BCH code with the same m and n but with a value of $t=1$. This code has a minimum distance of $2t+1 = 3$ and therefore can detect double errors. Those are precise the errors that can correct the code with $t=2$. Therefore one interesting observation is that for a given BCH code that can correct t errors the upper $m \cdot \lceil t/2 \rceil$ rows of the H matrix are sufficient to detect t errors. This means that correctable errors can be detected by computing only part of the syndrome. This observation is used in the following section during the derivation of the proposed scheme.

III. BCH CODE PARTITIONING SCHEME

The proposed technique is based on dividing the data block in two parts and storing additional information that allows us to detect errors in any of the two parts. This can be done by storing the partial result of the syndrome computation for the first half of the block. Then if the first half is accessed that partial result can be used to check if there are errors. If the second half is accessed, the same partial result can also be used to check if there are errors. This is so because the overall syndrome is zero in the absence of errors. Therefore the partial syndrome has to be the same in the two halves so that the XOR of both results is zero.

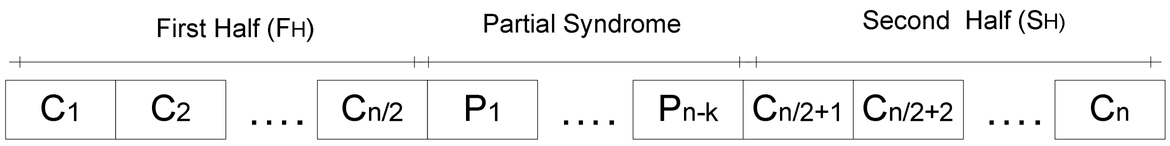

Figure. 1: Illustration of the proposed partitioning scheme.

An illustration of the proposed scheme is shown in Figure 1. In the proposed scheme the coded word is divided in two halves $F_H = \{C_1, C_2, \ldots, C_{n/2}\}$and $S_H = \{C_{n/2+1}, C_2, \ldots, C_n\}$ and a partial syndrome P is also stored. The syndrome is computed as:

$$S = H \cdot c \tag{4}$$

where c is the coded block composed of bits $C_1, C_2, \ldots, C_n$ in column form. The partial syndrome P for the first half block is computed as

$$P = H_{FH} \cdot F_H \tag{5}$$

Where H_{FH} is a matrix composed of the first n/2 columns of H. In the absence of errors the syndrome will be zero and since

$$S = H \cdot c = H_{FH} \cdot F_H \oplus H_{SH} \cdot S_H \tag{6}$$

it follows that

$$P = H_{FH} \cdot F_H = H_{SH} \cdot S_H \tag{7}$$

This means that the partial syndrome can be used to check if there are errors in the first or the second half of the block. Therefore in the proposed scheme half blocks can be read and the partial syndrome is checked against the stored bits to detect errors. If there are errors, then the whole block is read and the errors are corrected using the BCH code. Since most reads will have no errors, the percentage of times that the full block has to be read will be small.

In Figure 1, all the bits in the syndrome are also stored in the partial syndrome. This would double the number of additional bits required for the ECC. So it would be more effective to use a block of half the size protected with a BCH code that can correct the same number of errors. However, the observation about the H matrix of BCH codes in the previous section enables a reduction of the number of bits that need to be stored in the partial syndrome. This number is given by $m \cdot \lceil t/2 \rceil$ and the bits that have to be stored are the ones that correspond to the top $m \cdot \lceil t/2 \rceil$ rows of the H matrix. With this optimization the number of additional bits is reduced compared to using a traditional BCH with half the block size. As an example the three options are compared in Table I in terms of the number of additional bits required for 128B data bits and t =2. It can be observed that the proposed scheme provides an intermediate option between using two blocks of 64B or a block of 128B.

TABLE I. NUMBER OF ADDITIONAL BITS

BCH block of 128B	22
Two BCH blocks of 64B	40
Proposed Scheme	33

In the general case and assuming that the number of additional bits for a BCH code is $n - k = m \cdot t$ for a block size k the proposed scheme requires:

$$B_{proposed} = m \cdot t + m \cdot \left\lceil \frac{t}{2} \right\rceil \tag{8}$$

additional bits for n bits. This compares to $m \cdot t$ bits for a BCH code of the same block size and $2 \cdot (m-1) \cdot t$ for a BCH code of half the block size.

The discussion so far has not considered errors that affect the partial syndrome bits. Those however can potentially lead to undetected errors. As an example consider again the case $t=2$. Suppose that an error affects one data bit C_i for which the corresponding column in H has only a non zero bit in the rows that are stored in the partial syndrome. Then if another error affects the bit in the partial syndrome that corresponds to the row that has the non zero value for the first bit in error, the error will not be detected. Therefore, this double error will not be corrected. Since the BCH code has $t=2$ the error would in principle be correctable if it is detected. For the H matrix in (3) this would occur for example if an error affects the first bit of the word and the first bit of the partial syndrome.

To address the issue of errors affecting the partial syndrome, a parity bit could be used to protect the partial syndrome. This would ensure that single errors in the partial syndrome are detected. When that occurs, the complete block is read and the full BCH code is used to ensure that the error on the partial syndrome does not mask errors on the data bits. When t is larger than 2 a SEC or SEC-DED code can be used to protect the partial syndrome to ensure that multiple errors are detected.

The protection of the partial syndrome solves the issue of undetected errors but adds more additional bits to the block. This can be avoided using a different approach. Since the number of data bits in a block *d* is typically a power of two (for example 64B), from (1) the block size *n* has to be *2•d-1* to accommodate *d* data bits. This means that the code is shortened as *k* will be much larger than *d*. The shortening is done by setting to zero some of the data bits in all blocks. Those bits are not stored in the cache and the decoder automatically sets then to zero. When *t=2*, the shortening can be used to remove the bits in H for which the first *m* rows have only a bit that is one. If all those columns are removed, then a single error in a data bit and another in the partial syndrome will always be detected. The same approach could be used when *t* is larger than two but in that case, with a number of errors of *t*, multiple errors can occur in the data bits and the partial syndrome. This complicates the analysis that is not developed further in this paper.

In this section the proposed approach has been presented in a general form. In the next section a specific case is illustrated covering all the details and showing its applicability to practical situations.

IV. CASE STUDY

A cache that uses 64B sub-blocks in the access operations has been selected as a case study. This is the same configuration used in [11]. Using the proposed scheme the cache is structured in blocks of 128B. A BCH code with *t =2* (i.e. a double error correction code) is used in each block. This code is obtained by shortening the (*n=2047, k= 2045*) BCH code so that the number of data bits is 128B. To that end 1011 bits are set to zero. The partial syndrome is composed of 10 bits that are also stored in the memory as illustrated in Fig.1.

The shortening of the code is performed in such a way that the issue of undetected errors discussed in the previous section is avoided and the implementation is optimized. The optimizations enable the reduction of the number of additional bits and the simplification of the decoding logic. The process starts with the first m rows of the original H matrix that will be denoted as H_{SEC}. For the *(n= 15, k =7)* code in (3) for which *m=4* this would be:

$$H_{SEC} = \begin{bmatrix} 1&0&0&0&1&0&0&1&1&0&1&0&1&1&1 \\ 0&1&0&0&1&1&0&1&0&1&1&1&1&0&0 \\ 0&0&1&0&0&1&1&0&1&0&1&1&1&1&0 \\ 0&0&0&1&0&0&1&1&0&1&0&1&1&1&1 \end{bmatrix} \quad (9)$$

and the seven data bits correspond to the seven leftmost columns in the matrix.

Now, the columns in H_{SEC} are rearranged such that the m^{th} row has all the zero values on the first columns. This reordered matrix will be denoted as $H_{SEC\text{-}Reordered}$ and its shown below for the *(n= 15 ,k =7)* code. This reordering will allow us to reduce the number of bits stored in the partial syndrome as discussed later.

$$H_{SEC-\mathrm{Re}ordered} = \begin{bmatrix} 1&0&0&1&0&1&1&0&0&1&0&0&1&1&1 \\ 0&1&0&1&1&0&1&0&0&1&1&1&1&0&0 \\ 0&0&1&0&1&1&1&0&1&0&0&1&1&1&0 \\ 0&0&0&0&0&0&0&1&1&1&1&1&1&1&1 \end{bmatrix} \quad (10)$$

The shortening is now done over the reordered data block. To that end the columns that have only a one in the first *m-1* rows are eliminated (i.e. the value of the associated bit is fixed to zero). This can be done as long as those columns correspond to data bits and not to parity bits whose value cannot be fixed as it depends on other data bits. For large block sizes this is not an issue as most bits are data bits (for example for a 64B block we have 512 data bits and 10 parity bits). For the *(n= 15 ,k =7)* code the shortening is done to get a block with four data bits with the following $H_{SEC\text{-}Final}$ matrix:

$$H_{SEC-Final} = \begin{bmatrix} 1&0&1&1&0&0&1&0&0&1&1&1 \\ 1&1&0&1&0&0&1&1&1&1&0&0 \\ 0&1&1&1&0&1&0&0&1&1&1&0 \\ 0&0&0&0&1&1&1&1&1&1&1&1 \end{bmatrix} \quad (11)$$

which has data bits on columns 1,2,5 and 6. Now the block can be divided in two sub-blocks, the first comprises bits 1 to 4 and the second the rest. The partial syndrome is also stored, but only the first *m-1 =3* bits are needed as the m^{th} bit will always have a value of zero in the first block and therefore from (7) also in the second. This reduces the number of additional bits to *m-1* instead of *m*. In fact for this scheme to work, the fifth column also needs to be removed as otherwise an error on that bit would not be detected.

For larger data blocks, many bits have to be removed in the shortening process. The bits removed can be selected in such a way that the two sub-blocks have:

1. The same number of data bits.
2. Similar columns in the $H_{SEC\text{-}final}$ matrix.

and

1. All the columns with only one bit with a value of one in the first m-1 rows are removed.
2. The column that has all the bits with a value of zero in the first m-1 rows is removed.

Then the word can be reordered again so that the columns in the two sub-blocks are matched. This enables the use of similar equations to compute the partial syndrome in both sub-blocks thus reducing the implementation cost. This matrix will be denoted as $H_{SEC\text{-}Opt}$ and is illustrated below for a BCH code with *(n = 63, k=51)*.

$$H_{SEC\text{-}Opt} = \begin{array}{c} \qquad F_H \qquad\qquad\qquad S_H \\ \left[\begin{array}{l} 00110110010101111011111\,0011011001010111101111 \\ 10111011001111000101111\,1011101100111100010110 \\ 01011011100010101011100\,0101101110001010101110 \\ 10011001110101111100101\,1001100111010111110010 \\ 01101100101011110111100\,0110110010101111011110 \\ 00000000000000000000000\,1111111111111111111111 \end{array}\right] \end{array} \quad (12)$$

It can be observed that most columns in the first (F_H) and second (S_H) sub-block are the same.

The described procedure has been applied to the (*n=2047, k= 2025*) BCH code to obtain a block of 1046 bits of which 22 are parity bits and 1024 data bits. This block of 128B data bits is divided in two sub-blocks of 523 bits containing 512 data bits and 11 parity bits. The corresponding columns in the $H_{SEC\text{-}Opt}$ matrix for each sub-block are the same except for two columns. This enables the use of very similar logic to perform error detection in the two sub-blocks.

The proposed scheme requires 22 bits for the BCH code and 10 for the partial syndrome giving a total of 32 bits. This compares with the 40 bits required to implement the BCH code in 64B blocks and 22 bits to implement the BCH for 128B blocks. Therefore, it provides an intermediate option in terms of cost. In terms of performance, it enables read access to 64B blocks unless they have errors. Since errors are rare, the impact of those cases should be negligible. This means that the overhead of reading 128B blocks and its impact on dynamic power consumption is avoided.

The proposed scheme for a 128 block with two 64B sub-blocks has been implemented and tested using fault injection to ensure that all single and double errors are detected. All possible combinations for single and double errors have been exhaustively checked. Once an error is detected, the complete block would be read and the DEC BCH code used to correct the errors.

The error detection process for some errors is described in the following:

- A single error on any of the sub-blocks will be detected as all the columns have no zero values in the first m-1 rows.
- A double error on any of the sub-blocks will be detected as the first m-1 rows are different in the columns in each sub-block.
- A single or double error in the partial syndrome will be detected as the check for the affected row (or rows) will give an error.
- A single error in a sub-block combined with a single error in the partial syndrome will be detected as by construction, all the columns have more than a bit with a value of one in the first m-1 rows.

In summary, all single and double errors are detected when reading a sub-block. This ensures that those errors will be corrected by accessing the complete block and using the BCH code.

V. Conclusions

This paper has presented a technique to enable a better granularity for read accesses to a memory protected with BCH codes. This is done by partitioning the data block in two sub-blocks and storing additional information to perform error detection per sub-block. This enables read operations to access each sub-block and only if there are errors, the complete block has to be read.

The proposed scheme can be useful when BCH codes are used to protect embedded DRAM caches from the errors caused by using large periods between refreshes. In that case, BCH codes for large blocks are used to minimize the number of additional bits. However, this requires to access large blocks even when only a small part of the block is needed. This increases the dynamic power consumption and reduces the effective memory bandwidth. In this context the proposed scheme enables to read sub-blocks independently thus mitigating the problem.

Future work will focus on evaluating the power savings for a given processor and for different software benchmarks as done in [11]. The application of the technique to BCH codes that can correct more errors will also be considered. The first step will be to consider codes that can correct four errors and derive the optimum check matrixes for each of the sub-blocks.

References

[1] R. Matick and S. Schuster, "Logic based eDRAM: origins and rationale for use," IBM Journal of Research and Development, vol. 49, no. 1, pp. 145 – 165, Jan. 2005.

[2] R. Kalla, "Power7: IBM's next generation POWER microprocessor," Presentation at Hot Chips 21, Stanford, CA, Aug. 2009.

[3] J. Barth, et al., "A 500 MHz random cycle, 1.5 ns latency, SOI embedded DRAM macro featuring a three-transistor micro sense amplifier", IEEE Journal of Solid State Circuits, vol. 43, no. 1, pp. 86–95, Jan. 2008.

[4] M. Ghosh and H. Lee, "Smart refresh: An enhanced memory controller design for reducing energy in conventional and 3D die-stacked DRAMs," in Proceedings of the 40th International Symposium on Microarchitecture, pp. 134–145, Dec. 2007.

[5] W. Kong, et al., "Analysis of retention time distribution of embedded DRAM - A new method to characterize across chip threshold voltage variation," in Proceedings of IEEE International Test Conference (ITC 2008), pp. 1-7, Oct. 2008.

[6] P. Emma, W. Reohr and M. Meterelliyoz, "Rethinking refresh: Increasing availability and reducing power in DRAM for cache applications," IEEE Micro, vol. 28, no. 6, pp. 47-56, Nov 2008.

[7] P. Oehler, S. Hellebrand: Low Power Embedded DRAMs with High Quality Error Correcting Capabilities, 10th IEEE European Test Symposium (ETS05) , pp. 148-153, May 2005.

[8] R.C. Baumann, "Radiation-induced soft errors in advanced semiconductor technologies", IEEE Trans. On Device and Materials Reliability, Vol. 5, No. 3, 2005, pp. 301-316.

[9] C. L. Chen and M. Y. Hsiao, "Error-correcting codes for semiconductor memory applications: a state-of-the-art review", IBM Journal of Research and Development 28(2), 1984, pp. 124-134.

[10] J.A. Maestro, P. Reviriego, "Reliability of Single-Error Correction Protected Memories", IEEE Transactions on Reliability, Vol. 58, No 1, March 2009, pp. 193-201.

[11] C. Wilkerson, A. R. Alameldeen, Z. Chishti, W. Wu, D. Somasekhar and S. Lu, "Reducing Cache Power with Low Cost, Multi-bit Error-Correcting Codes", International Symposium on Computer Architecture, pp. 83-93, Jun. 2010.

[12] S. Lin and D. J. Costello, "Error Control Coding, 2nd Ed.", Englewood Cliffs, NJ: Prentice-Hall. 2004.