Exploiting Unused Spare Columns and Replaced Columns to Enhance Memory ECC

Hyunseung Han, *Student Member, IEEE*, Nur A. Touba, *Fellow, IEEE*, and Joon-Sung Yang, *Member, IEEE*

***Abstract*—Due to the emergence of extremely high density memory along with the growing number of embedded memories, memory yield is an important issue. Memory self-repair using redundancies to increase the yield of memories is widely used. Because high density memories are vulnerable to soft errors, memory error correction code (ECC) plays an important role in memory design. In this paper, methods to exploit spare columns including replaced defective columns are proposed to improve memory ECC. To utilize replaced defective columns, the defect information needs to be stored. Two approaches to store defect information are proposed—one is to use a spare column and the other is to use a content-addressable-memory. Experimental results show that the proposed method can significantly enhance the ECC performance.**

***Index Terms*—Content-addressable-memory (CAM), defect information, memory error correction code (ECC), soft error, spare column, yield.**

I. Introduction

As MEMORY density has grown, memories have become very susceptible to transient errors caused by electromagnetic interference, static electricity, cosmic ray, alpha particles, and so on. To overcome the occurrence of transient memory errors, an error correction process is needed. Memory error correction code (ECC) such as single-error-correcting, double-error-detecting (SEC-DED) codes and single-error-correcting, double-adjacent-error-correcting (SEC-DAEC) codes [3], are an essential part of a memory design.

Due to their dense structure, memories are also inherently susceptible to defects. To prevent yield loss, spare rows and columns are included in memories to facilitate memory repair. After the memory is tested, an attempt is made to repair any defective cells by reconfiguring to replace them with spare rows and columns instead. In many cases, not all of the spares are used in the repair process. This paper proposes a methodology for exploiting unused memory spare columns to improve a memory ECC thereby increasing reliability (preliminary results were reported in [1]). Unused spare columns are used to store additional check bit information and this increases the number of memory check bits per memory location. By storing additional check bits in the unused spare columns, the miscorrection probability of memory ECC can be reduced. This concept is extended further in this paper by also proposing a scheme for using the columns that have been replaced by spare columns. These columns may have one or more defects, but they still contain many good memory cells. A scheme is presented for utilizing these good memory cells to store check bits to further enhance the memory ECC and increase overall system reliability. The proposed scheme does not require any additional columns to be added to the memory, but rather it simply exploits unused spare columns left over after memory repair and unused memory cells in replaced columns.

SEC-DED codes are always able to correct single errors and detect double errors. However, if there are more than two-bit errors, the error detection is not guaranteed and error miscorrection can happen. For SEC-DAEC codes [3], nonadjacent double errors can also cause miscorrection. Additional check bits can be used to reduce the miscorrection probability at the cost of longer codewords. With the proposed scheme, additional check bits can be stored in unused spare columns or in replaced columns, so reliability is improved without the need for adding more columns to the memory. It simply makes use of existing unused memory cells. One drawback is that it provides nonuniform ECC capability. Memories in which there are many unused spare columns will get a larger boost in ECC capability in comparison to memories where there are no unused spare columns. Moreover, rows in which there are no defects will get a larger boost in ECC than rows where there are defects such that even the replaced columns do not have any working memory cells. However, the proposed scheme requires very little overhead and hence boosts reliability where it can with very low cost.

This paper is organized as follows. Section II introduces previous related works. Proposed schemes will be introduced in Sections III–V. Experimental results will be shown in Section VI, and this paper will be concluded in Section VII.

II. Linear Block Codes and Previous Works

The conventional SEC-DED codes [4], [5] are systematic linear block codes [9], [14]. In a (n, k) linear block code, k data bits are encoded by n-bit codewords. The number of check

Manuscript received March 17, 2015; revised July 8, 2015 and September 30, 2015; accepted November 6, 2015. Date of publication March 15, 2017; date of current version August 18, 2017. This work was supported in part by the Basic Science Research Program through the National Research Foundation of Korea by the Ministry of Education under Grant NRF-2015R1D1A1A01058856, in part by Samsung Research Fund, and in part by the National Science Foundation under Grant CCF-1217750. This paper was recommended by Associate Editor H.-G. Stratigopoulos.

H. Han and J.-S. Yang are with Sungkyunkwan University, Suwon 440-746, South Korea (e-mail: js.yang@skku.edu).

N. A. Touba is with the University of Texas at Austin, Austin, TX 78712 USA.

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TCAD.2017.2682639

bits is $r = (n - k)$. The (rxn) parity-check matrix (H-matrix) completely defines the code. C is a codeword of the code if and only if

$$H \cdot C^T = 0 \tag{1}$$

where C^T is the transpose of the codeword C. Let each element in the error vector E be a 1 if the corresponding bit is in error and a 0 if the bit is error-free, then an erroneous message can be represented as $V_{\text{error}} = V \oplus E$. The syndrome, S, is defined as

$$S = H \cdot V_{\text{error}} = H \cdot (V \oplus E) = H \cdot V \oplus H \cdot E = H \cdot E. \tag{2}$$

If there is no error (i.e., $E = 0$), then the syndrome is all zero (i.e., $S = 0$). If the syndrome is nonzero, then an error is detected. Let h_i represent the ith bit being a one. The syndrome, which is equal to the product of H and E, will be equal to h_i. For an SEC Hamming code, each column vector in the H-matrix is nonzero and distinct [4]. This ensures that the syndrome for any single-bit error will result in a unique syndrome. By decoding the syndrome, it is possible to determine which bit the error is in and flip the value of that bit to correct the error.

For a double-bit error, the syndrome is equal to the XOR of two columns of the H-matrix. If the XOR of any two columns is equal to the syndrome for any single-bit error (i.e., equal to any column in the H-matrix), then the double-bit error syndrome would alias with the single-bit error syndrome. The correction logic would miscorrect the double-bit error thereby missing the error. To avoid this, it was shown in [5], that if every column of the H-matrix has an odd number of 1s and is distinct, the code will be SEC-DED. The reason is that the XOR of any two columns with an odd number of 1s will produce a syndrome with an even number of 1s and hence is guaranteed to be different from any single column. This means that the syndromes for double-bit errors will always be different from the syndromes for single-bit errors, so the code will always detect double-bit errors and not miscorrect them. Hsiao codes are also called *odd-weight column codes*. Note that many double-bit errors have the same syndrome, so it is generally not possible to correct double-bit errors since their syndromes cannot be distinguished.

For triple-bit errors, the syndrome is formed from three columns being XORed together. If the syndrome matches one of columns of the H-matrix, then it will be miscorrected as a single-bit error. The number of possible triple-bit errors is C_3^n, and the fraction of those that match columns of the H-matrix is the miscorrection probability for triple-errors. For most conventional SEC-DED codes, it is in excess of 50%.

In [3], the H-matrix is constructed using odd-weight columns where the columns are carefully ordered so that adjacent columns when XORed together give a syndrome that is not equal to the other $n - 1$ syndromes and the number of single-bit errors is n, so the combined set of $2n - 1$ syndromes must all be distinct from each other. This permits correction of both single-bit errors and adjacent double-bit errors (i.e., SEC-DAEC). However, nonadjacent double-bit errors may match one of the $(n-1)$ syndromes of the adjacent double-bit errors and hence may result in miscorrection.

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

Fig. 1. Example of (7, 3) SEC-DED Hsiao code.

New linear SEC-DED codes which have ECC improvement by reducing triple error miscorrection probability were proposed in [10]. Ishaq *et al.* [12] and Gherman *et al.* [13] introduced ways to select the H-matrix for additional check bits. Dutta [2] proposed an SEC-DED, and double-adjacent-error-correcting code with complete elimination of miscorrection of nonadjacent double errors separated by up to $d-1$ bits (SEC-DED-DAEC-CEM-d).

III. Improving ECC By Storing Check Bits in Unused Memory Cells

The proposed scheme involves exploiting unused spare columns and replaced columns in the memory to store additional check bits. These additional check bits add extra rows to the H-matrix and increases the dimension of the syndrome. This makes it easier to distinguish syndromes thereby reducing the chance of miscorrection as well as reducing the chance of a multibit error's syndrome aliasing with the error-free all-zero syndrome and not being detected at all.

Note that in some cases, it may not be possible to store any additional check bits. Thus, the H-matrix that is selected should be such that if no additional check bits are available, it still retains the SEC-DED property. The easiest way to ensure this is to start with an SEC-DED code, and then incrementally add the extra rows to it.

The rows are added one at a time in a greedy fashion so that if only one spare is available after repair, then the maximum benefit for that one row is achieved. Consider the example in Fig. 1 which is a (7, 3) SEC-DED Hsiao code. It has $C_3^7 = 35$ possible 3-bit errors, and 28 of those will result in miscorrection. In Fig. 2, an extra check bit is added to the H-matrix from Fig. 1. This results in an additional row (the bottom-most one) and an additional column (the right-most one). The last five columns in Fig. 2 correspond to check bits and hence form an identity matrix. The left three columns correspond to message bits. The bottom-most bit in the first three columns may be set to any value so as to minimize the miscorrection probability. In Fig. 2, the bottom-most bit in the second column is set to 1 and the others to 0. Now only 12 of the 56 possible triple-bit errors will result in miscorrection.

Starting from an SEC-DED code, the proposed scheme adds rows one at a time. The columns corresponding to check bits form an identity matrix, so the degree of freedom is in selecting the 1s and 0s in the row for the columns corresponding to message bits. There are few different strategies that can be used. If the number of message bits is less than say 30, it is possible to do an exhaustive search. Each possible combination of 1s and 0s for the row can be tried and the miscorrection

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

Fig. 2. Adding one row to example in Fig. 1.

probability computed. The one that minimizes the miscorrection probability is then selected. As the number of message bits gets larger, however, then an exhaustive search is no longer possible.

For larger codes, an alternative to an exhaustive search would be to do a random search and simply keep the best code found. The number of triple-errors is equal to C_3^n which is manageable for n up to hundreds. It is feasible to enumerate all the triple-errors and compute the exact miscorrection probability for each candidate row.

The procedure is the same for SEC-DAEC codes. In this case, the goal is to minimize the number of nonadjacent double-bit errors that miscorrect. This is even faster to evaluate since there are fewer possibilities.

When searching the codes, other criteria can be optimized as well such as total number of XOR gates or logic depth of the syndrome generator.

Each row is added one at a time up to the maximum number of spare columns available in the memory. In the best-case, if no spare columns are used for repair, then all the extra rows will be active for error detection and correction. In the worst-case, when all spare columns are used for repair, then none of the extra rows will be active, and hence only the original SEC-DED code that was used as the starting point will remain.

IV. Scheme for Storing Check Bits in Unused Spare Columns

The proposed scheme can be implemented with very little modification to a normal memory that uses spare columns and is protected with an SEC-DED code. Fig. 3 shows an example of the scheme assuming a single spare column. The additional logic that is added to support the scheme is the following.

1) An extra XOR tree in the check bit generator and syndrome generator to support one additional check bit.
2) An extra 2-input AND gate to disable the extra syndrome bit when determining error detection if the spare is used for repair.
3) An extra 2-input OR gate in the correction logic for each data bit to disregard the extra syndrome bit if the spare is used for repair.

Other than what is listed above, the rest of the circuitry is already present in a conventional memory with a spare column and SEC-DED ECC.

If the spare is used for repair, then the MUXes at the input and output of the memory will shift the bits so that the defective column is bypassed. The control signal for the MUX on the far right will be a "1" if the spare is used for repair or if

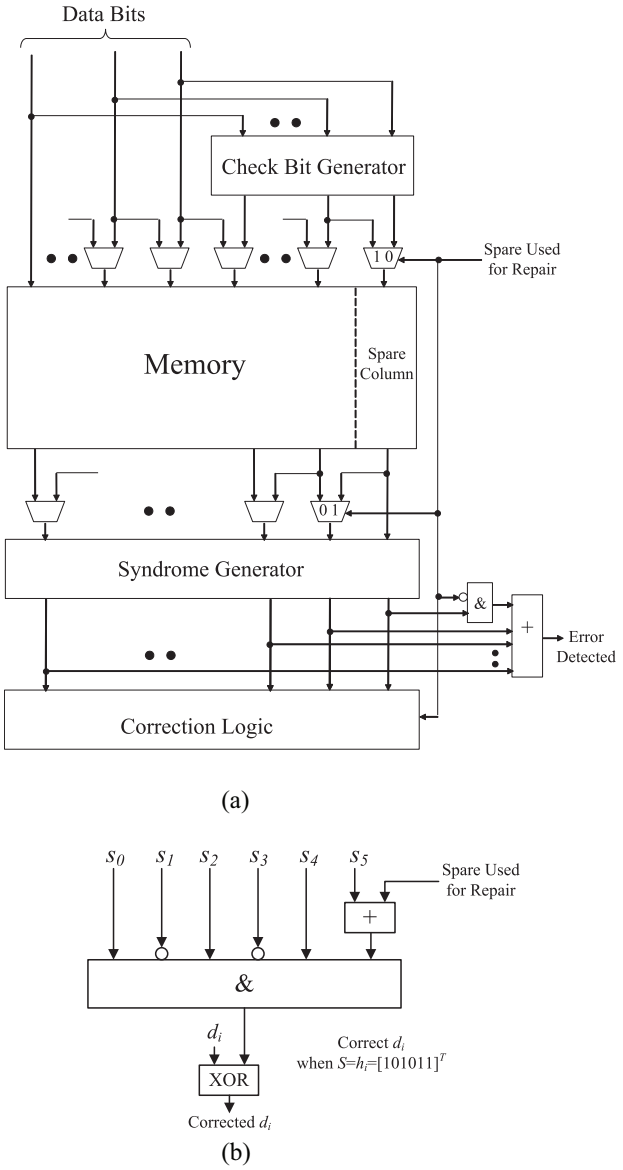

Fig. 3. Storing check bits in unused spare column method. (a) Block diagram for one spare column utilization. (b) Example of bit-slice of correction logic.

the spare column itself has a defect. If this control signal is a "0," then the spare is available for storing the extra check bit.

So if the spare is not used for repair, then the extra check bit generated by the check bit generator is stored in the spare column, otherwise, it is simply ignored. At the output of the memory, the extra syndrome bit that is generated is ignored if the spare is used for repair in which case error detection and correction are performed just as if that extra syndrome bit did not exist. However, if the spare is not used for repair, then the extra syndrome bit is used to help increase the chance of detecting a multibit error as well as reduce the probability of miscorrection.

V. Scheme for Storing Check Bits in Replaced Columns

Manufacturing based memory defect rates has been shown to vary [6], hence, the number of spare columns used to

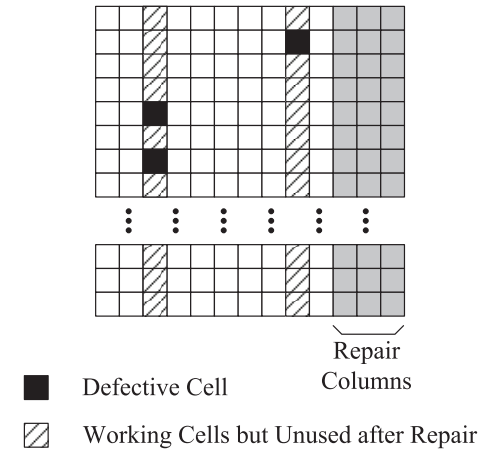

Fig. 4. Memory array with column repairs.

replace the defective columns changes. If there are remaining unused spare columns, the proposed method tries to use them to store additional check bits. In this section, a method to try to further enhance the memory ECC by exploiting replaced columns is described. After repair, the replaced column(s) has a number of nondefective and working memory cells left unused. Memory ECC performance can be further enhanced by utilizing them.

A. Exploiting Unused Spare Columns and Replaced Columns for ECC Enhancement

The number of available (i.e., unused) spare columns varies and this determines the number of additional check bits. If all spare columns are used to replace defective memory columns, no additional check bits can be used. Note that the repair process needs to be conducted even when there is only one defective cell in a memory column. In this case, a spare column is used to replace the defective column with one memory defect. If each memory column is composed of 100 memory cells, there are still 99 nondefective cells in the defective column. Memory ECC performance can be further enhanced if the remaining good cells in the replaced columns can be utilized. The proposed method utilizes them to increase the check bit length of memory ECC. Fig. 4 illustrates an example of a memory array with a defect and spare columns.

Fig. 5(a) shows two memory rows with two defective cells and repair columns highlighted in black and gray, respectively. If two column repairs are used to replace defective cells, the memory array can be logically drawn as in Fig. 5(b) after repair. The repair process generates two working cell but unused after repair (WEARs) in the example (they are highlighted in a dashed rectangle). If they are not utilized, the 18-bit codeword is stored in memory rows excluding last two memory cells in Fig. 5(b). The last two bits need to be discarded in an error correction process. The 18-bit codeword vectors, v_1 and v_2, are represented in Fig. 5(c) and the syndrome generator generates a syndrome by multiplying an H-matrix with the transposed codeword vector. H-matrix is constructed as $[P_{kx(n-k)} : I_{kxk}]$ where P is a transpose of the

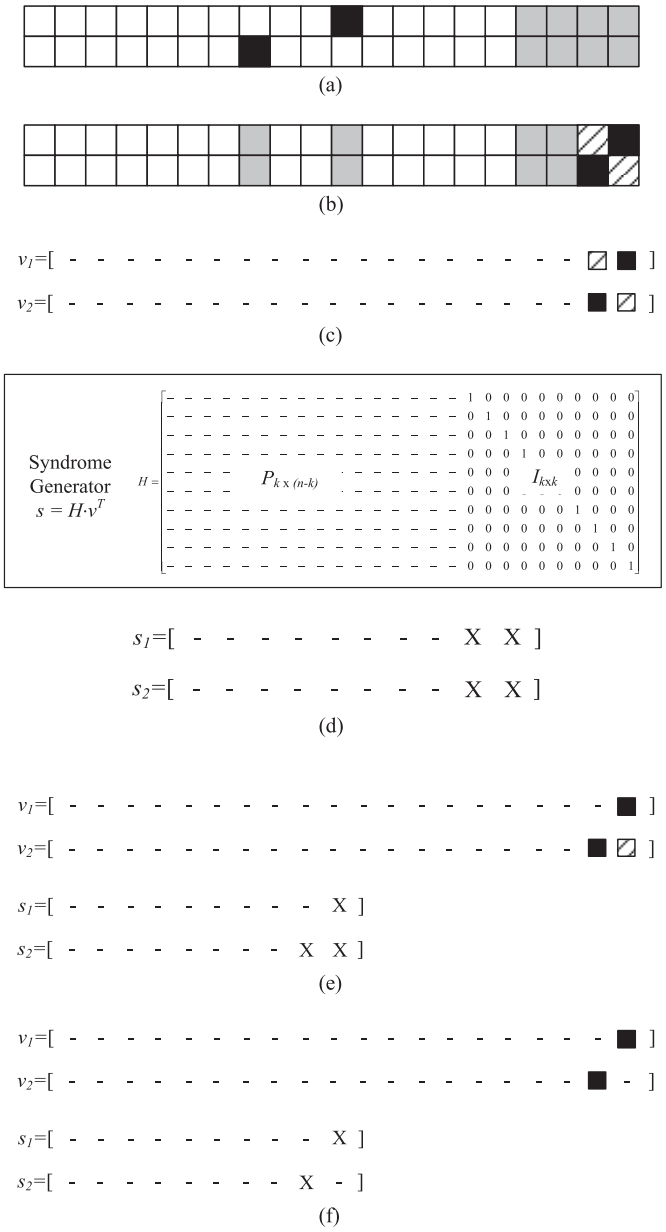

Fig. 5. Memory repair and syndrome generation examples with WEARs. (a) Before repair process. (b) After repair process (logically illustrated memory array). (c) Codewords from (b). (d) Syndrome generation with v_1 and v_2 and syndrome s_1 and s_2 with last two bits (X) not used for error correction. (e) Utilizing WEAR in v_1 enhancing ECC capability. (f) Utilizing WEAR in v_1 and v_2 enhancing ECC capability.

generator matrix (G) and I is an identity matrix. The syndromes generated by the codewords in Fig. 5(b) are given in Fig. 5(d). Because of the identity element in H-matrix, the position of not-used bits for error correction in syndrome can be predicted with the defective cell location in the codeword. In this example, since the last two bits of v_1 and v_2 cannot hold valid check-bits, the last two bits (X) in the syndromes, s_1 and s_2, are discarded from error correction. If the WEARs are used to store check bits to further enhance memory ECC, the additional check bits can be stored in WEAR. As shown in Fig. 5(e) and (f), depending on how WEARs are utilized, the additional check bits can be increased. The proposed

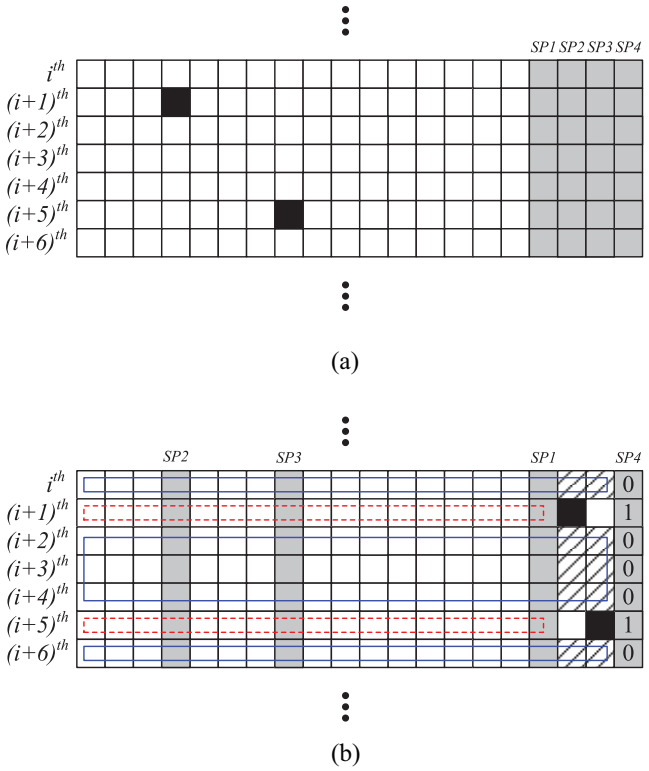

Fig. 6. Example of repair process by spare columns. (a) Memory with two defective cells and four spare columns (SP1–SP4). (b) Logically described memory array after repair process with defect information stored in SP4.

scheme can be applied to existing repair methods using fuses or multiplexers. Fig. 3 shows an example of multiplexer based repair.

Two methods to utilize *WEAR*s are proposed and the following sections describe them in detail.

B. Method to Store Repair Information in Memory

Defective memory cells are replaced either by spare rows or spare columns. For the proposed method, because using spare rows first helps to leave more spare columns unused, it is preferred to perform row repair first. The proposed method to store repair information in memory requires one unused spare column. One unused spare column is used to store the repair information indicating rows with or without a defective cell.

Fig. 6(a) shows an example with four repair columns (SP1 − SP4) and two defective cells located on $(i+1)$th and $(i+5)$th rows. SP2 and SP3 are selected to replace the defective columns. After repair, the memory can be logically redrawn as shown in Fig. 6(b). There is one unused spare column, SP1, which is used for one additional check bit. The repair information is stored in SP4 where 0 denotes the row with no defective cells and 1 indicates the row with defective cells. Because the defective cells are found in $(i+1)$th and $(i+5)$th rows, the corresponding locations in SP4 have 1 while the rest has 0s. The defect information can be written during a booting or reset sequence by a firmware. Based on the defect information, more memory cells can be used to enhance an ECC capability with the proposed method. The ith row does not include any defective cells and it can utilize two *WEARs* in the example. Hence, if the 16-bit codeword is used, the proposed method can add three more check bits, two *WEARs* and one unused spare column (i.e., 16-bit codeword + 3 more check bits) for rows with no defective cells and they are highlighted with solid line rectangles in Fig. 6(b). For rows with SP4 containing 1, because two spare columns are used, one additional bit can be used for ECC enhancement (i.e., 16-bit codeword + 1 check bit) and the dashed rectangles indicate them. In this manner, the proposed method enhances the ECC capabilities by utilizing *WEARs* and unused spare columns. If a memory row is composed of n regular columns and m used spare columns out of total r spare columns, the n-bit codeword can be increased as follows.

1) Increased codeword size for row without defective cells—n-bit codeword + $(r-1)$-bit additional check bit.
2) Increased codeword size for row with defective cells—n-bit codeword + $(r - m - 1)$-bit additional check bit.

In Fig. 6, n, r, and m are 16, 4, and 2, respectively. Hence, ith, $(i+2)$th, $(i+3)$th, $(i+4)$th, and $(i+6)$th which are the rows without defective cells have 16-*bit codeword* + 3 *additional check bits* and $(i+1)$th and $(i+5)$th, the rows with defective cells, have 16-*bit codeword* + 1 *additional check bit*. The proposed method provides nonuniform ECC capabilities for the rows depending on the location of defective cells and this falls within a class of unequal error protection codes.

The hardware block diagram for the proposed method is given in Fig. 7. The ECC enhanced codeword from reconfiguration logic is written to the memory with spare columns. Note that the last spare column dedicated to repair information bit is fed into the *Masking Info* block that generates masking bits for syndrome generator. *Masking Info* block generates $(r-1)$ bits which corresponds to the size of additional check bits. Depending on the location of defective cells, they are masked before entering the syndrome generator. For example, "100" masking pattern is generated for the case in Fig. 6(b) to mask last two bits when the row with defective cells is read and "111" masking pattern is generated when the row without defective cell is read. Since the masking pattern is determined by fault location, Masking Info block needs to be programmed during repair phase.

The proposed method requires one spare column to store repair information. This reduces the number of available spare columns for additional check-bit.

C. Method to Use Content-Addressable Memory for Storing Repair Information

The second method to utilize WEARs for enhancing ECC capabilities is to use a content-addressable memory (CAM). In this method, instead of dedicating one unused spare column for repair information, CAM is used to store the word addresses of the rows with a defective cell. CAM provides a performance advantage over other memory search algorithms. The CAM-based method hardware block diagram is depicted in Fig. 8.

The address goes into both an address decoder and CAM in the proposed method. The CAM finds out if there is a matching word address. If there is no matching data, it means that the

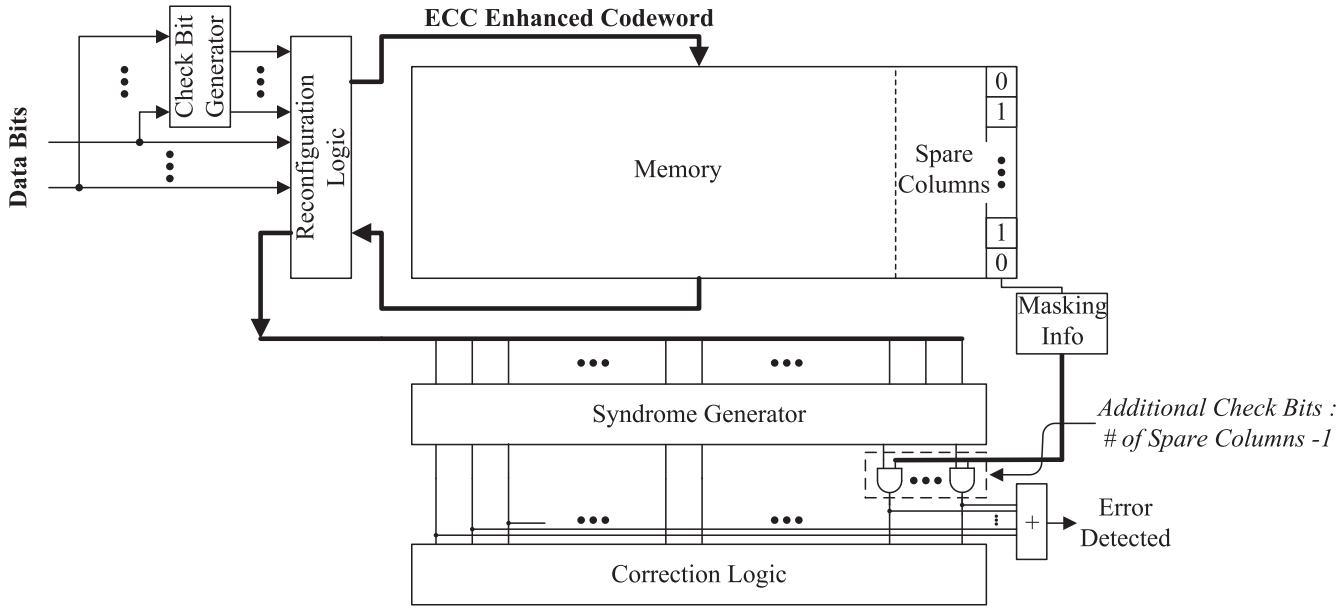

Fig. 7. Block diagram of proposed scheme for storing repair information in memory.

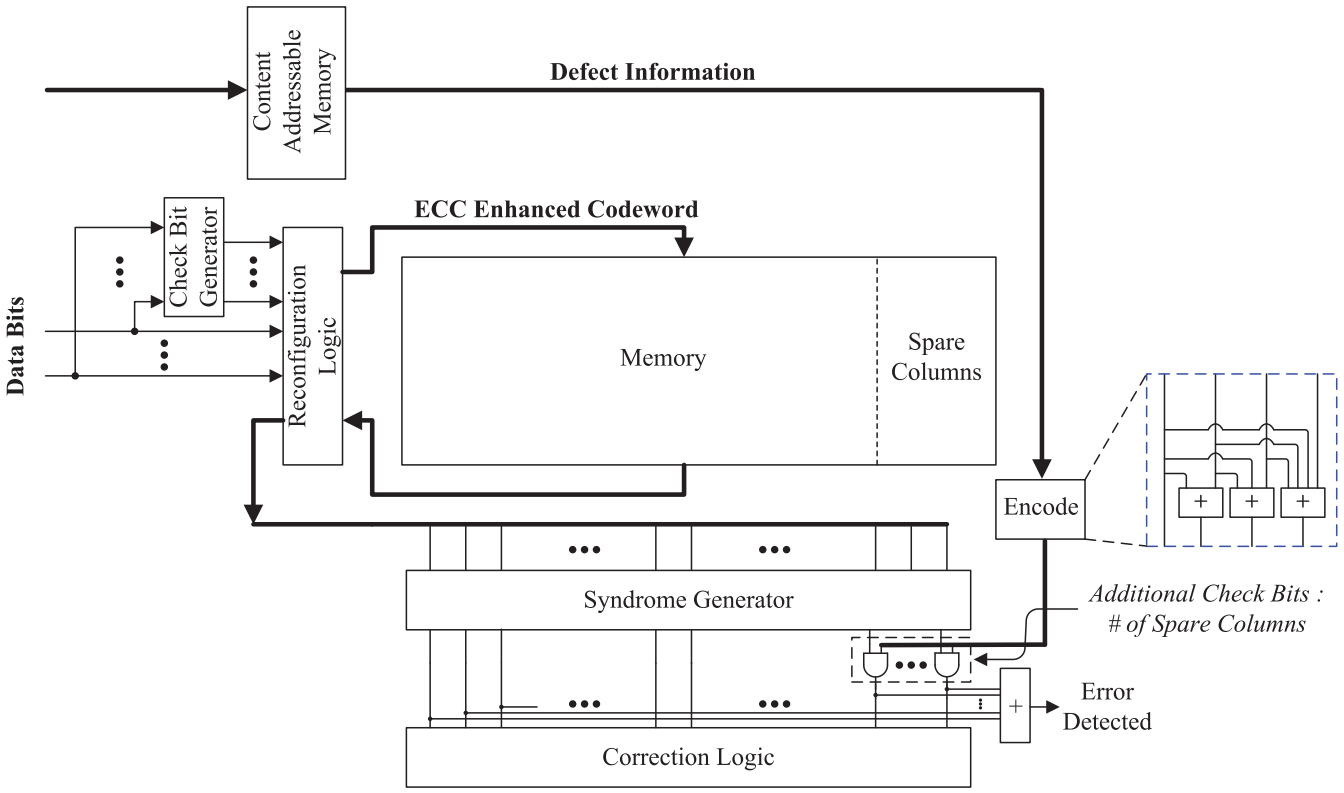

Fig. 8. Block diagram of proposed scheme for using CAM.

addressed row has no defect and the error correction logic works normally (i.e., *Masking Info* block generates r masking bits with all 1s for ANDing since there is no defective cells in the row). If there is a matching word address, the *Masking Info* block generates masking bits and the defective bits are masked when they flow into a syndrome generator.

In the CAM-based method, ECC capability can be higher than the repair information stored in the unused spare column method, since no repair column is used for storing defect information and it can be used to store an additional check-bit. Fig. 9 describes an example of CAM based method.

Fig. 10 shows a CAM structure which is used in the proposed scheme. It has a similar structure with conventional CAM [16], except for its encoder. The output of conventional CAM is address information, which is encoded from matchlines. However, the CAM structure used in the proposed

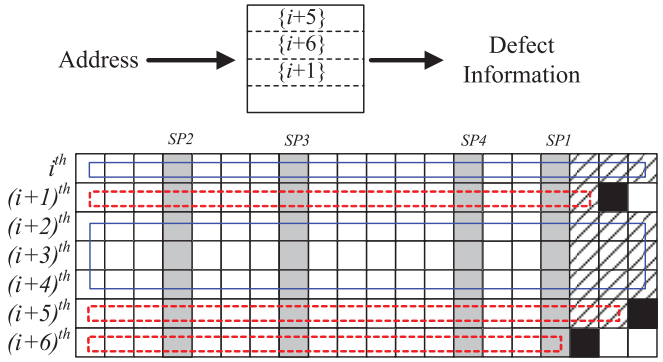

Fig. 9. Example of the second method which utilizes CAM.

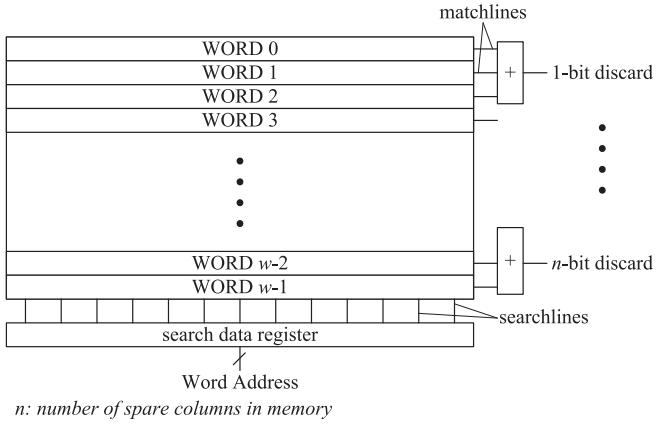

Fig. 10. CAM structure used to store addresses of rows with a defective cell.

TABLE I
MISCORRECTION PROBABILITY

Additional check-bit (32-bit)	0	1	2	3	4
3-bit error	0.59663	0.27571	0.12812	0.05925	0.02766
4-bit error	0.07112	0.04476	0.03700	0.02446	0.01451
5-bit error	0.60989	0.26890	0.12172	0.05645	0.02667
Additional check-bit (64-bit)	0	1	2	3	4
3-bit error	0.55594	0.26662	0.12781	0.06148	0.02947
4-bit error	0.02867	0.02293	0.01973	0.01366	0.00843
5-bit error	0.52089	0.24060	0.11214	0.05295	0.02521

scheme has OR gates which determine the number of bits to discard. The number of OR gates is equal to the number of spare columns in memory.

As can be seen in Fig. 9, the location of defective cells restricts the utilization of WEARs. To overcome the limitation, each partition of CAM stores the word addresses depending on the defective cell location after column repair. When $(i+5)$th row is accessed, the rightmost one check-bit should be discarded. An address of the $(i+5)$th row is stored in the first partition of CAM, and 1-bit masking signal is generated. In the same manner, the addresses of $(i+1)$th row and $(i+6)$th

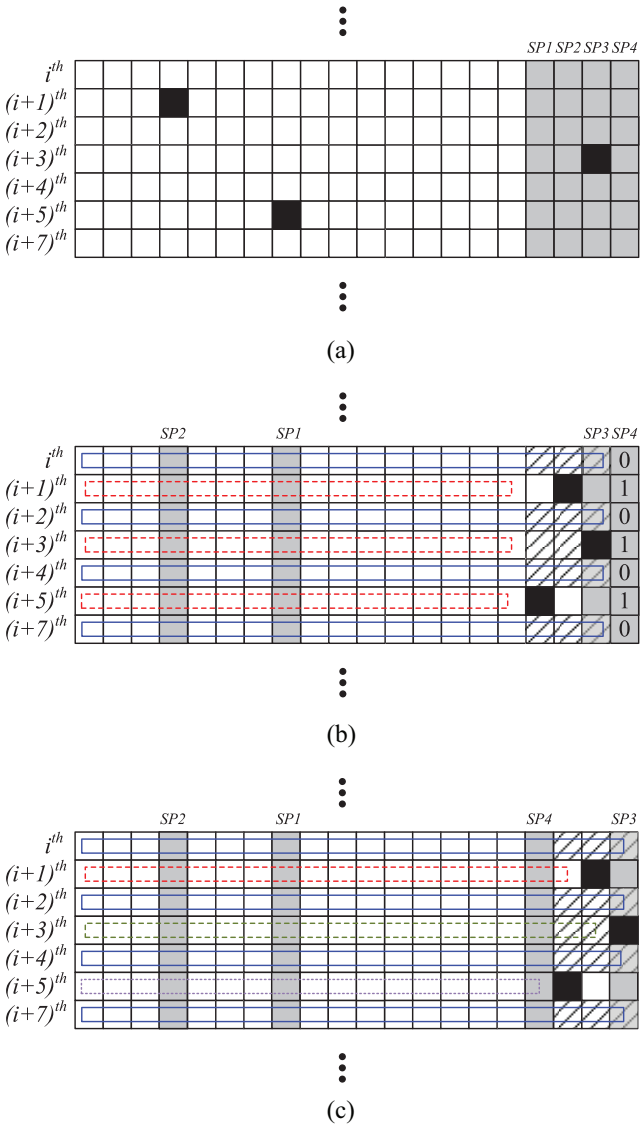

Fig. 11. Example of ECC enhancement with defective spare column case. (a) Memory with two defective cells, one defective spare column (SP3), and three good spare columns (SP1, SP2, SP4). (b) Logically described memory array after repair process with proposed scheme for storing repair information in memory. (c) Logically described memory array after repair process with proposed scheme for using CAM.

row are stored in the second and third partition of CAM. They generate 2-bit discard and 3-bit discard signals, respectively. Since there are three spare columns replaced, the last partition contains no addresses.

As a result, 15 out of 18 WEARs are used and 15 additional bits besides of one entire spare column can be utilized to enhance the ECC capability. Depending on a defect map which is generated during repair process, OR gate encoder in the CAM can be configured and the size of partitioned can be adjusted.

Memory defect locations can be classified as three categories: 1) only in memory array; 2) memory array and spare column; and 3) only in spare column. The proposed method can be applied to a memory with any defect scenarios. The main idea is initially explained with a case 1) that has defects

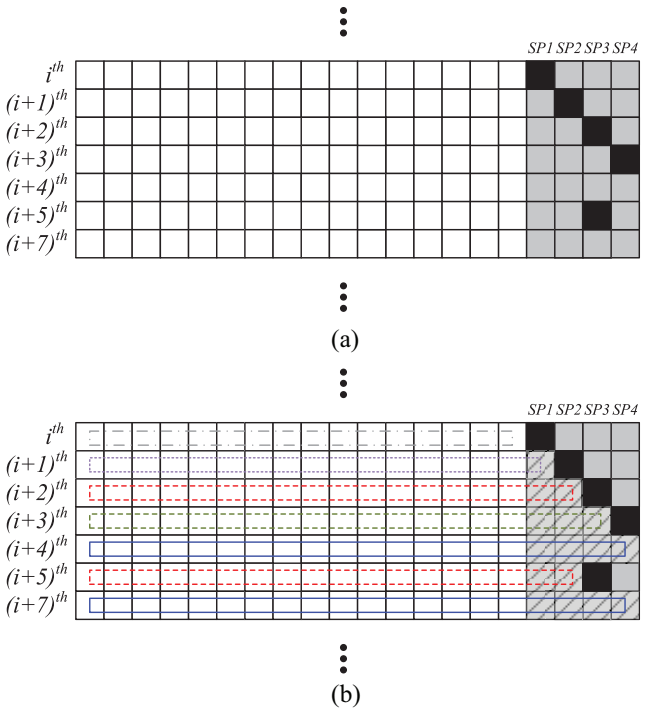

Fig. 12. Example of ECC enhancement with only defective spare columns case. (a) Memory with defects only in spare columns (SP1–SP4). (b) Logically described memory array with proposed scheme for storing repair information in memory.

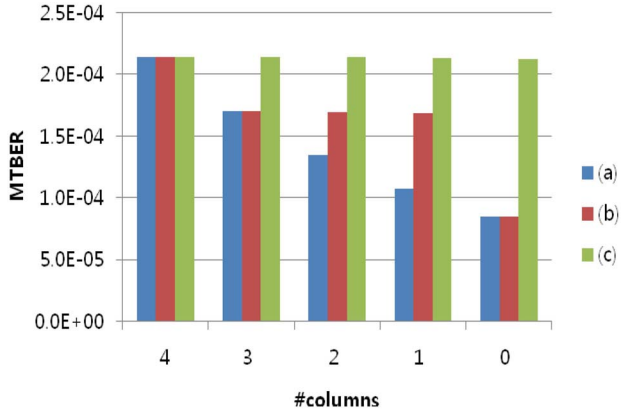

Fig. 13. MTBER degradations in 1K×32 memory.

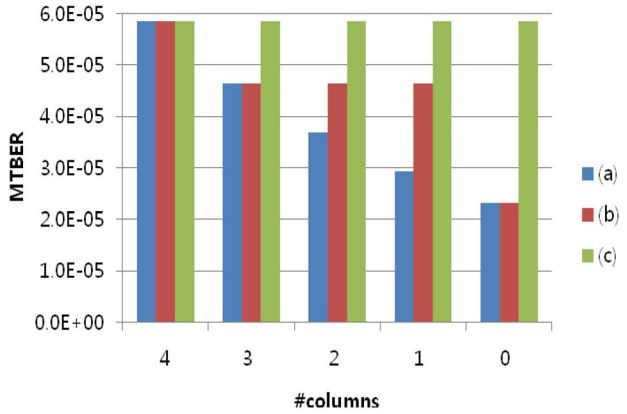

Fig. 14. MTBER degradations in 8K×64 memory.

only in memory array. In manufacturing, memory array and spare columns/rows are tested. If there is a defective memory cell, it can be repaired either by a spare row or column. If there is a defective spare column—case 2), it cannot be used for repair process. The proposed methods still utilize it for memory reliability improvement. The defective spare column can be considered as a column with WEARs. Fig. 11 shows a similar example with Fig. 6. In Fig. 11(a), there is one defect found in a spare column, SP3, hence, it cannot be used for repair. SP1 and SP2 are used to replace defective columns. There is still one unused spare column available, SP4, in the example, SP4 is used to store repair information in the proposed method. Fig. 11(b) shows the logically described memory array after repair. *i*th, $(i+2)$th, $(i+4)$th, and $(i+7)$th rows in SP3 are used to store additional check-bits, therefore, this can be considered as WEARs in defective columns. If the proposed method for using CAM is used with a defective spare column case, more additional check-bits can be utilized and Fig. 11(c) shows how the memory can be logically described after repair. Depending on the address information in CAM, the encoding bits are generated accordingly. $(i+1)$th, $(i+3)$th, and $(i+5)$th addresses are stored in CAM. For example, when $(i+1)$th is accessed, the bits for SP3 and defective memory cell replaced by SP2 are generated by encoding logic to mask the corresponding syndrome bits. In a worst case, defective cells can only be found from spare columns—case 3). Fig. 12 shows an example with defects found in SP1–SP4. Because there is no defect in the memory array, a repair process will not occur. However, the proposed method still can utilize the spare columns for ECC enhancement. In Fig. 12(a), there is no available defect free spare column. Since the proposed scheme for storing repair information in memory requires one available column repair, this method cannot be applied. However, the proposed method for using CAM still can be used. The defective row addresses, *i*th, $(i+1)$th, $(i+2)$th, $(i+3)$th, and $(i+5)$th are stored in CAM, and the corresponding encoded bits are generated to mask the bits in syndrome analysis. It should be noted that if there is one available spare column with case 3), the proposed method for storing repair information in memory can be applied. In this manner, the proposed method provides a complete treatment of all three defect cases—1) only in memory array; 2) memory array and spare column; and 3) only in spare column.

Once a volume has been ramped up, the manufacturing process becomes mature [6] and the number of defective cells found in a memory would be very small. For the proposed method, the spare rows can be used first for repair and this helps to leave the spare columns unused. The CAM-based method can achieve higher ECC capability and provides more flexibility than the method that stores repair information in the spare column.

VI. Experimental Results

This section shows evaluations of the proposed scheme. In Section VI-A, a reliability improvement of the proposed

TABLE II
ESTIMATED MTBERS OF DIFFERENT MEMORY CONFIGURATIONS

Num. of Available columns	Memory Reliability Enhancement Methods	1K x 32	8K x 32	1K x 64	8K x 64
4-column	(a), (b), (c)	2.139E-4	1.068E-4	1.173E-4	5.858E-5
3- column	(a) Utilizing Unused Spare Column Only	1.699E-4	8.488E-5	9.300E-5	4.645E-5
	(b) Storing Repair Information in Memory	1.699E-4	8.488E-5	9.300E-5	4.645E-5
	(c) Using Content-Addressable Memory	2.139E-4	1.068E-4	1.173E-4	5.858E-5
2- column	(a) Utilizing Unused Spare Column Only	1.346E-4	6.727E-5	7.383E-5	3.689E-5
	(b) Storing Repair Information in Memory	1.696E-4	8.485E-5	9.282E-5	4.644E-5
	(c) Using Content-Addressable Memory	2.137E-4	1.068E-4	1.172E-4	5.857E-5
1- column	(a) Utilizing Unused Spare Column Only	1.069E-4	5.343E-5	5.857E-5	2.926E-5
	(b) Storing Repair Information in Memory	1.688E-4	8.481E-5	9.237E-5	4.641E-5
	(c) Using Content-Addressable Memory	2.132E-4	1.068E-4	1.169E-4	5.855E-5
0- column	(a) Utilizing Unused Spare Column Only	8.483E-5	4.239E-5	4.648E-5	2.323E-5
	(b) Storing Repair Information in Memory	8.483E-5	4.239E-5	4.648E-5	2.323E-5
	(c) Using Content-Addressable Memory	2.122E-4	1.067E-4	1.163E-4	5.852E-5
	SEC-DED only (No Spare Column Utilization)	8.483E-5	4.239E-5	4.648E-5	2.323E-5

scheme is evaluated and the overhead analysis on the proposed CAM based approach is given in Section VI-B.

A. Reliability Evaluation

For reliability evaluation, we used a maximally tolerable bit-error rate (MTBER). MTBER satisfying a specific memory error rate requirement. The MTBER can be used to measure how much BER is tolerated in the memory. Hence, this is used as a reliability index in this paper.

For MTBER estimations, the miscorrection probabilities for Hsiao SEC-DED codes are given in Table I with different additional check-bit length and the number of errors. As can be expected, the miscorrection probabilities are reduced with larger check-bit length. A memory failure probability can be calculated from the miscorrection probability, as

$$P_{\text{failure}}(\text{BER}) = \sum_{e=0}^{n} \text{BER}^{e} \cdot (1-\text{BER})^{n-e} \binom{n}{e} P_{\text{miscorrection}}(e) \tag{3}$$

and MTBER is defined as the largest BER which satisfies

$$P_{\text{failure}}(\text{BER}) < \text{Memeory Error Rate}. \tag{4}$$

In this paper, the memory error rate requirement is assumed as 3.398E-6, which is defined from six-sigma rule.

Table II shows the MTBER values. We assume that the memory initially includes 4 repair columns and MTBER is measured with different number of available repair columns after repair process. The first column gives the number of available column repairs after repair and the second column shows a list of approaches used to enhance the reliability. Method (a) is the conventional approach introduced in [1]. Method (b) is the proposed method for storing repair information in memory, and method (c) is the proposed technique for using CAM given in Sections VI-B and VI-D, respectively. The last four columns show MTBER with different memory sizes from 1K×32 to 8K×64. For experiments, we assumed that the replaced column contains one defect. This leaves 1023 $(1K-1)$ and 8191 $(8K-1)$ WEARs left in the replaced column for 1K × word_size and 8K × word_size memories. It should be noted that MTBER degradation is negligible with different number of defects in the defective column. For example, MTBER becomes worse by 0.5% when there are seven defective cells in a single column with 1K×64 memory and 42 defective cells in a single column for 8K×64 memory. Since there are still a number of WEARs in the replaced column, MTBER is not drastically affected by the number of defects in the column.

The table assumes that there is no defect in the spare column. In manufacturing testing, spare rows and columns are checked first. If there is a defective spare column, it may not be used to replace a defective column. The defective spare column can also be taken care for memory ECC enhancement. The defective spare column can be considered as a column with WEARs.

Figs. 13 and 14 illustrate an MTBER degradation with different numbers of available spare columns for 1K×32 and 8K×64 memories. As can be seen, when there is no defect (4 available spare columns), the same MTBER is achieved from method (a)–(c). As the number of available spare columns decreases, MTBER significantly reduces in method (a). In method (b), MTBER drops from 4 available spare columns to 3 available spare columns since one entire spare column is used to store repair information not for additional ECC check-bits. Because there are a number of WEARs with 3, 2, or 1 available column repairs, the MTBER is negligibly changing and is kept constant in method (b). However, when there is no available spare column, the proposed method (b) achieves the same MTBER as the method (a) since it cannot store repair information. However, the proposed method (c), which stores repair information in CAM, achieves the MTBER regardless of the number of available column repairs.

B. CAM Overhead Analysis

In this section, an overhead by the proposed CAM-based approach is evaluated. Table III describes the details of CAM

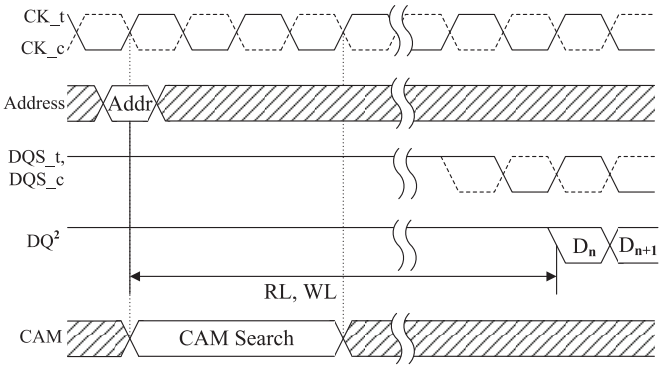

Fig. 15. DRAM timing diagram with CAM search operation.

TABLE III
CAM SPECIFICATIONS

	ISSCC 15 [18]	TVLSI 15 [17]	ESSCIRC 11 [16]	SACAS 12 [15]
Technology	90nm	65nm	32nm	90nm
Capacity	128x64b	512x128b	128x128b	256x144b
Size	0.8704 μm^2	NaN	NaN	NaN
Latency	0.96ns	0.6ns	1.07ns	0.26ns
Energy/bit	0.51fJ	0.078fJ	0.3fJ	0.162fJ
Energy/32b	16.32fJ	2.496fJ	9.6fJ	5.184fJ

TABLE IV
TIMING CONSTRAINTS OF DRAM [20]

Param.	DDR4-1600	DDR4-1866	DDR4-2133	DDR4-2400
tCK	1.250 ns	1.071 ns	0.938 ns	0.833 ns
tCL	13.75 ns	12.85 ns	13.13 ns	13.33 ns
tRCD	13.75 ns	13.92 ns	14.06 ns	14.16 ns
tRP	13.75 ns	13.92 ns	14.06 ns	14.16 ns
tRAS	35.00 ns	34.00 ns	33.00 ns	32.00 ns
tRC	48.75 ns	47.92 ns	47.06 ns	46.16 ns
RL	9-16 CK	9-18 CK	9-20 CK	9-23 CK
WL	9-15 CK	10-16 CK	11-18 CK	12-21 CK

TABLE V
ENERGY CONSUMPTION IN DRAM [19]

Command	DDR3-1333	DDR3-800
Read (array)	18 nJ	18 nJ
Write (array)	20 nJ	20 nJ
Read I/O	1 nJ	1.7 nJ
Write I/O	4 nJ	7 nJ
I/O additional termination	12 nJ	20 nJ
Activate+Precharge	25 nJ	25 nJ

examples [15]–[18]. This table is used to estimate the hardware and power consumption overhead caused by CAM in the proposed scheme for using CAM.

To analyze a performance overhead by the CAM, an example of DRAM timing constraints is described in Table IV. It shows that main timing components are longer than 10 ns, which is significantly larger than CAM latency given in Table III.

The proposed scheme for using CAM in Fig. 8 receives the address and the address is fed by address decoder and the CAM. Hence, the search operation in CAM can be performed

TABLE VI
ESTIMATED AREA OVERHEAD

Defect ratio	1.00E-5	7.50E-6	5.00E-6	1.00E-6
memory size/CAM	1.280E7	1.707E7	2.560E7	1.280E8
μm^2/MB	0.5704	0.4278	0.2852	0.0570
μm^2/GB	584.1	438.2	292.1	58.41

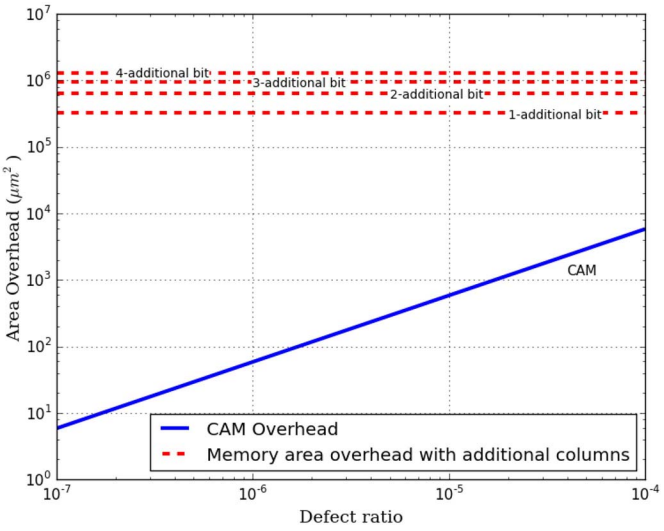

Fig. 16. Area overhead comparison: additional column versus implementing CAM.

in parallel with DRAM array access. Fig. 15 describes a timing diagram of DRAM with CAM. Table III shows that the CAM search latency ranges from 0.26 ns to 1.07 ns which can be covered by 2 clock cycles (2-CK) of DRAM in Table IV. Based on the specification data sheet [20], the data read or write latency (RL or WL) takes at least 9 clock cycles (9-CK). Hence, it can be understood that the CAM operation occurs in parallel with memory read or write operations and does not cause any performance overhead.

To evaluate a power overhead, the energy consumption in DRAM [19] is shown in Table V. While the energy consumption for DRAM array access is around 20 nJ, the energy consumed by CAM operation does not exceed 20 fJ assuming 32-bit long address, which is smaller by orders of magnitude than DRAM. The power overhead imposed by CAM operation is estimated to be less than 1e-6 (20 fJ/20 nJ) and, therefore, this can be generally considered insignificant.

In the proposed method, the size of CAM is determined by the average number of defects. If there are two defective cells in a word, the defective word just occupies one CAM entry, instead of two entries. In worst case, the number of CAM entry is determined when every defect is distributed to different word. However, as a manufacturing process becomes stable, the number of defects significantly decreases and this may leave spare columns unused. In this regard, the CAM would not require many entries. Hence, the size of CAM is expected to be reasonably small.

To show the benefit of adding a CAM instead of enlarging the memory array for additional check-bit, the CAM area

overhead is estimated. A maximum capacity of memory which can be handled by a single CAM is calculated as follows:

$$\text{memory size}/\text{CAM} = \frac{\#\text{ of (CAM entry)}}{\text{(cell defect ratio)}} \tag{5}$$

where a defect cell ratio is calculated as the number of defective cells divided by the entire memory capacity. As the equation indicates, the maximum memory capacity that can be handled by a single CAM can increase with a large number of CAM entry and low defective cell ratio. Table VI shows an estimated CAM area overhead with a CAM configuration introduced in [18]. The first row indicates a memory cell defect ratio, and it determines a CAM area overhead. The second row shows the maximum memory capacity handled by a single CAM by (5). The third and last row indicates the average CAM area overhead required for 1 MB and 1 GB array.

Fig. 16 shows an area overhead comparison between the proposed CAM based approach and the way to widen memory array for additional check-bits. For DRAM memory array size increase analysis, 1–4 additional check-bits are considered and the area overhead is illustrated as dashed lines. The area overhead estimation assumes a 20-nm technology node with 64 bit word DRAM. The following shows estimation details.

1) *$6F^2$ Cell:* $6 \cdot (20\text{ nm})^2 = 2400\text{ nm}^2 = 0.0024\ \mu\text{m}^2$.
2) *1GB Memory (No ECC):* $2^{33} \cdot 0.0024\ \mu\text{m}^2 = 20\,615\,843\ \mu\text{m}^2$.
3) *One Additional Column:* $20\,615\,843 \cdot (1/64) = 322\,123\ \mu\text{m}^2$.
4) *n-Additional Column:* $n \cdot 322\,123\ \mu\text{m}^2$.

If four additional columns are added to an array, the size is estimated as $1\,288\,490\ \mu\text{m}^2$ which is more than 2000 times larger than the proposed CAM-based method.

VII. Conclusion

In this paper, the methods to enhance a memory ECC capability are proposed exploiting unused repair columns and replaced columns. Experimental results show that the proposed methods achieve remarkable a memory reliability enhancement. Better MTBER implies that the memory can be designed with lower power and higher speed since BER margins for design requirements are increased. CAM-based methods achieve better and more consistent reliability than the memory column-based scheme with an area overhead for CAM. However, it is proven that the hardware overheads such as latency, power, and area overheads come from implementing CAM are negligible.

As simulation results show, the memory reliability by the proposed methods are not much influenced by the number of defects. Because the proposed methods utilize both unused spare columns and WEARs, the number of defective cells hardly affects ECC capability and all memories would have relatively similar ECC performances.

References

[1] R. Datta and N. A. Touba, "Exploiting unused spare columns to improve memory ECC," in *Proc. VLSI Test Symp.*, Santa Cruz, CA, USA, May 2009, pp. 47–52.

[2] A. Dutta, "Low cost adjacent double error correcting code with complete elimination of miscorrection within a dispersion window for multiple bit upset tolerant memory," in *Proc. IEEE/IFIP 20th Int. Conf. VLSI Syst.-On-Chip*, Santa Cruz, CA, USA, Oct. 2012, pp. 287–290.

[3] A. Dutta and N. A. Touba, "Multiple bit upset tolerant memory using a selective cycle avoidance based SEC-DED-DAEC code," in *Proc. VLSI Test Symp.*, Berkeley, CA, USA, May 2007, pp. 349–354.

[4] R. W. Hamming, "Error correcting and error detecting codes," *Bell Syst. Tech. J.*, vol. 29, pp. 147–160, Apr. 1950.

[5] M. Y. Hsiao, "A class of optimal minimum odd-weight-column SEC-DED codes," *IBM J. Res. Develop.*, vol. 14, no. 4, pp. 395–401, Jul. 1970.

[6] L. D. Hung, H. Irie, M. Goshima, and S. Sakai, "Utilization of SECDED for soft error and variation-induced defect tolerance in caches," in *Proc. Design Autom. Test Europe Conf. Exhibit.*, Nice, France, 2007, pp. 1–6.

[7] I. Y. Kim *et al.*, "Built in self repair for embedded high density SRAM," in *Proc. Int. Test Conf.*, Washington, DC, USA, Oct. 1998, pp. 1112–1119.

[8] K. Pagiamtzis and A. Sheikholeslami, "Content-addressable memory (CAM) circuits and architectures: A tutorial and survey," *IEEE J. Solid-State Circuits*, vol. 41, no. 3, pp. 712–727, Mar. 2006.

[9] W. W. Peterson and E. J. Weldon, *Error Correcting Codes*. Cambridge, MA, USA: MIT Press, 1972.

[10] M. Richter, K. Oberlaender, and M. Goessel, "New linear SEC-DED codes with reduced triple bit error miscorrection probability," in *Proc. IEEE Int. On-Line Test. Symp.*, Jul. 2008, pp. 37–42.

[11] N. A. Touba, "Fault-tolerant design" in *System-On-Chip Test Architectures*. San Francisco, CA, USA: Morgan Kaufmann, 2007, pp. 123–168.

[12] I. Ishaq, J. Jung, J. Song, and S. Park, "Efficient use of unused spare columns to improve memory error correcting rate," in *Proc. IEEE Asian Test Symp.*, New Delhi, India, Nov. 2011, pp. 335–340.

[13] V. Gherman, S. Evain, F. Auzanneau, and Y. Bonhomme, "Programmable extended SEC-DED codes for memory errors," in *Proc. VLSI Test Symp.*, Dana Point, CA, USA, May 2011, pp. 140–145.

[14] D. K. Pradhan, *Fault-Tolerant System Design*. Upper Saddle River, NJ, USA: Prentice-Hall, 1996.

[15] A. Agarwal *et al.*, "A 128×128b high-speed wide-and match-line content addressable memory in 32nm CMOS," in *Proc. ESSCIRC*, Helsinki, Finland, Sep. 2011, pp. 83–86.

[16] N. Onizawa, S. Matsunaga, V. C. Gaudet, and T. Hanyu, "High-throughput low-energy content-addressable memory based on self-timed overlapped search mechanism," in *Proc. Symp. Asynchronous Circuits Syst.*, Kongens Lyngby, Denmark, May 2012, pp. 41–48.

[17] H. Jarollahi, V. Gripon, N. Onizawa, and W. J. Gross, "Algorithm and architecture for a low-power content-addressable memory based on sparse clustered networks," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 23, no. 4, pp. 642–653, Apr. 2015.

[18] M.-F. Chang *et al.*, "17.5 A 3T1R nonvolatile TCAM using MLC ReRAM with sub-1ns search time," in *Proc. Int. Solid-State Circuits Conf.*, San Francisco, CA, USA, Feb. 2015.

[19] H. David, C. Fallin, E. Gorbatov, U. R. Hanebutte, and O. Mutlu, "Memory power management via dynamic voltage/frequency scaling," in *Proc. Int. Conf. Autonom. Comput.*, Karlsruhe, Germany, Jun. 2011, pp. 31–40.

[20] *Samsung DRAM Data Sheet*. Accessed on Jun. 2015. [Online]. Available: http://www.samsung.com/global/business/semiconductor/

Hyunseung Han (S'15) received the B.S. and M.S. degrees from Sungkyunkwan University, Seoul, South Korea, in 2014 and 2016, respectively, all in semiconductor engineering.

He has been with Samsung Electronics, Memory Division, Hwaseong, South Korea, since 2014, as an Engineer researching on Flash memory controller development. His current research interests include memory systems, system-on-a-chip design, and embedded systems.

Nur A. Touba (SM'05–F'09) received the B.S. degree from the University of Minnesota, Minneapolis, MN, USA, in 1990, and the M.S. and Ph.D. degrees from Stanford University, Stanford, CA, USA, in 1991 and 1996, respectively, all in electrical engineering.

He is currently a Professor with the Department of Electrical and Computer Engineering, University of Texas at Austin, Austin, TX, USA.

Dr. Touba was a recipient of the National Science Foundation Early Faulty CAREER Award in 1997, the Best Paper Award at the 2001 VLSI Test Symposium, and the 2008 Defect and Fault Tolerance Symposium. He served as the Program Chair for the 2008 International Test Conference and the General Chair for the 2007 Defect and Fault Tolerance Symposium. He currently serves on the program committee for the Design Automation and Test in Europe Conference, International On-Line Test Symposium, European Test Symposium, Asian Test Symposium, and Defect and Fault Tolerance Symposium.

Joon-Sung Yang (S'05–M'09) received the B.S. degree from Yonsei University, Seoul, South Korea, in 2003, and the M.S. and Ph.D. degrees from the University of Texas at Austin, Austin, TX, USA, in 2007 and 2009, respectively, all in electrical and computer engineering.

He was with Intel Corporation, Austin, TX, USA, for four years. He is currently an Associate Professor with Sungkyunkwan University, Seoul, South Korea. His current research interests include very large scale integration (VLSI) testing, silicon debug, and nanometer scale test and design methodologies.

Dr. Yang was a recipient of Korea Science and Engineering Foundation Scholarship in 2005, the Best Paper Award at 2008 IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems and at 2016 IEEE International SoC Design Conference, and was nominated for the Best Paper Award at 2013 IEEE VLSI Test Symposium.