Efficient Use of Unused Spare Columns for Reducing Memory Miscorrections

Jihun Jung*, Umair Ishaq*, Jaehoon Song, and Sungju Park***

***Abstract*—In the deep sub-micron ICs, growing amounts of on-die memory and scaling effects make embedded memories increasingly vulnerable to reliability and yield problems. Spare columns are often included in memories to repair defective cells or bit lines during production test. In many cases, the repair process will not use all spare columns. Schemes have been proposed to exploit these unused spare columns to store additional check bits which can be used to reduce the miscorrection probability for triple errors in single error correction–double error detection (SEC-DED). These additional check bits increase the dimensions of the parity check matrix (H-matrix) requiring extra area overhead. A method is proposed in this paper to efficiently fill the extra rows of the H-matrix on the basis of similarity of logic between the other rows. Optimization of the whole H-matrix is accomplished through logic sharing within a feasible operating time resulting in reduced area overhead. A detailed implementation using fuse technology is also proposed in this paper.**

***Index Terms*—Memory ECC, SEC-DED, logic sharing, parity check matrix, misscorrection probability, built-in self repair**

Manuscript received Dec. 15, 2011; revised Jun. 29, 2012.
* Dept. of Computer Science & Engineering, Hanyang University ERICA Campus, Ansan-si, Gyunggi-do, Korea
** TranSono Inc., Seoul, Korea
S. Park (parksj@mslab.hanyang.ac.kr) is a corresponding author.
Earlier version of this paper has been presented in the Asian Test Symposium 2011.

I. INTRODUCTION

With the scaling of process technologies into the nanometer regime, the reliability of embedded memory systems becomes an increasingly important concern for digital system designers. Nano-scale components themselves are increasingly likely to fail, and the growing amount of on-chip memory creates more possible points of failure. As designers integrate more of the memory hierarchy onto the processing die, the number and size of memory arrays on these system-on-chip (SoC) and microprocessors will increase [1]. Therefore, errors occurring in the embedded memory systems are a growing threat to the overall SoC and processor reliability and yield.

To protect the integrity of the data in the memory, error correcting code (ECC) plays a vital role [2]. The most common codes used are single error correction-double error detection codes. The most popular are Hamming [3], and Hsiao [4]. These codes can correct single bit errors in a codeword and can detect double bit errors. These codes require storing additional check bits in the memory. It is desired that a SEC-DED code reduces the probability of miscorrection for triple bit errors. Miscorrection occurs when an erroneous word is decoded in a wrong codeword without signaling an error. The majority of memories do not possess an exact number of data bits; hence, shortened codes must be used. The right choice of the columns of the check matrix of the shortened codes offers an opportunity to reduce the probability of miscorrection. In [5] a unique search procedure was described for the selection of columns in the H-matrix such that there is at least one even-weight column in the check matrix. It was also shown that how

codes for larger code length with a small triple bit error miscorrection can be generated from smaller matrix already determined. Along with the columns needed to carry out ECC, the memory includes additional spare columns for repair. In some cases, check bits are used along with spare rows and columns to provide a combined fault-tolerance. Current memory designs contain redundant rows, columns, and sub-arrays to tolerate manufacture-time hard errors and thus improve yields [6, 7]. When faulty bits are detected during product testing, the faulty addresses are remapped to redundant spare rows or columns using built-in self repair (BISR) techniques [8]. While in the worst-case most defective memories on the tail end of the statistical curve may use all of the spare resources, most memories will have unused spare resources after the repair.

A methodology has been proposed to exploit these unused resources, when available, to improve the reliability of the memory by enhancing its existing error coding [9]. This scheme exploited unused spare columns to store additional check bits which significantly reduced the miscorrection probability (MP) for triple-errors in SEC-DED codes. The additional check bits add extra rows to the H-matrix and increase the dimension of the syndrome. The increase in the dimension adds extra area and delay overhead to the entire system.

In this paper we propose a method to efficiently fill the additional rows of the H-matrix on the basis of similarity amongst the other rows of the H-matrix. For 16 bits of data our proposed method reduces the number of combination to 2^5 in comparison to 2^{16} for an exhaustive search. For larger codes such as 32 or 64 bits, exhaustive search is not possible. The proposed method plays a vital role in drastically reducing the number of combinations in selecting a best combination for additional row. Furthermore, we present a logic sharing method that not only reduces the hardware area but also delay of the overall design. Lastly, a new improved fuse base architecture design for the whole system is also proposed.

The paper is organized as follows. Section II provides a background and properties of SEC-DED codes. Section III describes the proposed methods. Experimental results are shown in Section IV and Section V concludes our discussion.

II. BACKGROUND

In this section we focus our attention on the conventional systematic linear block SEC-DED codes [3, 4, 10, 11]. The length of the code words, the number of information bits and the number of check bits are denoted by n, k and $r=(n-k)$, respectively. The H-matrix is:

$$H=[\mathrm{A}^T \mid \mathrm{I}_{n-k}] \tag{1}$$

Where A is a k-by-(n-k) parity check generator matrix and I_{n-k} is an (n-k)-by-(n-k) identity matrix. The code generator matrix denoted as **G** is defined as follows:

$$G=[\mathrm{I}_k \mid \mathrm{A}] \tag{2}$$

If **u** is a 1-by-k data bit vector, then its corresponding n bit codeword vector x is formed as $\mathbf{x} = \mathbf{u} \cdot \mathbf{G}$. In this paper, the codes are described by their (r-by-n) parity check matrix (H-matrix). C is a codeword of the code if and only if:

$$H.C^T = 0 \tag{3}$$

An error vector **E** is defined as an r-bit vector where the bits that are in error have a value 1 and all the other bits are 0. An erroneous message W_{error} can be represented as follows:

$$W_{error} = C \oplus E \tag{4}$$

The syndrome, S, is defined as follows:

$$S = H \cdot W_{error} = H \cdot (C \oplus E) = H \cdot E \tag{5}$$

The value of the syndrome is equal to zero if the transmitted codeword is not corrupted. If the received codeword contains detectable errors then the syndrome is non-zero. If the received codeword contains correctable errors, then the syndrome identifies the error pattern corrupting the transmitted codeword, and these errors can then be corrected.

For single error correction (SEC) Hamming code, each column vector in the H-matrix is non-zero and distinct [3]. 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.

$$h_{i_1} \oplus h_{i_2} = S(i_1, i_2) = h_j \quad (6)$$

Similarly, for triple-bit errors, the syndrome is formed from three columns being XORed together.

$$h_{i_1} \oplus h_{i_2} \oplus h_{i_3} = S(i_1, i_2, i_3) = h_j \quad (7)$$

If h_j is equal to the syndrome for any single bit error (i.e., equal to any column in the H-matrix), then the double-bit or triple-bit error syndrome, in Eq. (6) and (7) respectively, would be the alias with the single-bit error syndrome resulting in a miscorrection.

Hsiao provides a solution for double bit error miscorrection by using an H-matrix in which every column has an odd number of 1's and is distinct. The XOR of any 2 columns with odd 1's results in a syndrome with even number of 1's, ensuring syndrome different from any single column.

A major problem for Hsiao codes is the miscorrected triple bit errors. The number of possible triple-bit errors is C_3^n. For most conventional SEC-DED codes this percentage exceeds 50% [9]. The scheme proposed in [9] exploited unused spare columns to store additional check bits which significantly reduced the MP for triple-errors in SEC-DED codes

III. PROPOSED METHODS

In this paper we propose the following four methods;

1. Logic Sharing Method
2. Check Bit Addition(CBA) Method
3. Local Augmentation (LA) of Miscorrection
4. Probability
5. Spare Memory Architecture

1. Logic Sharing Method

In this section, an iterative method is proposed to find shared terms to minimize the circuit area built by H-matrix equations.

As discussed earlier, the additional check bits add extra rows to the H-matrix and increase the dimensions of the H-matrix as shown Fig. 1 and Fig. 2 [9].

The additional extra row affects the circuit area as well as the miscorrection rate; thus, logic sharing is proposed in this paper to fill the extra row with meticulous care to minimize both area and timing penalties. Our proposed method consists of six basic steps.

In the Step 1, output equations are transformed into an (n-by-m) matrix. In the Step 2, we compute the similarity matrix taking each row and checking the degree of similarity between the other rows. In the Step 3, we simply repeat the second step for all the following rows. In the Step 4, we optimize the similarity matrix by sorting it and deleting the duplicate rows and also the rows containing less than two 1's. In the Step 5 equations are replaced by the optimized variables. We further check the number of the optimized similarity matrix rows that we obtained. If the number of optimized rows is not 0 or 1 then the process from the Step 2 to the Step 5 is repeated again on the optimized similarity matrix to further optimize the matrix and equations will be further replaced by the optimized variables. Otherwise, the loop breaks and jumps to the Step 6 in which we get the optimized output equations which can be used instead of the original equations.

To better understand our method, an example is explained in detail. Let us consider a generalized matrix that produces a certain set of equations as follows;

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

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

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

Fig. 2. Adding One Row to Example in Fig. 1.

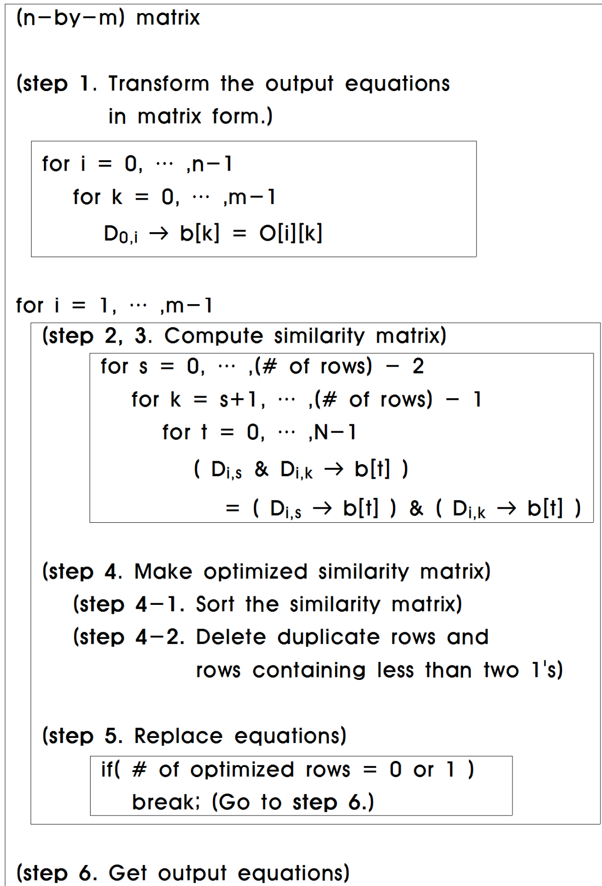

Fig. 3. Algorithm to find shared logics.

$$\begin{bmatrix} O[0] \\ O[1] \\ O[2] \\ O[3] \\ O[4] \\ O[5] \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 & 1 & 1 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} b[0] \\ b[1] \\ b[2] \\ b[3] \\ b[4] \\ b[5] \\ b[6] \\ b[7] \\ b[8] \end{bmatrix} \quad (8)$$

Step 1: Transform the output equations in matrix form.

$$\begin{aligned} O[0] &= b[1] \oplus b[3] \oplus b[4] \oplus b[5] \oplus b[8] \\ O[1] &= b[3] \oplus b[4] \oplus b[5] \oplus b[6] \oplus b[7] \oplus b[8] \\ O[2] &= b[0] \oplus b[1] \oplus b[2] \oplus b[3] \oplus b[6] \oplus b[7] \oplus b[8] \\ O[3] &= b[0] \oplus b[1] \oplus b[2] \oplus b[6] \\ O[4] &= b[0] \oplus b[4] \oplus b[7] \\ O[5] &= b[2] \oplus b[5] \oplus b[8] \end{aligned} \quad (9)$$

These output equations are transformed as equation matrix shown in Table 1.

Table 1. Equation Matrix

	b[0]	b[1]	b[2]	b[3]	b[4]	b[5]	b[6]	b[7]	b[8]
$D_{0,0}$	0	1	0	1	1	1	0	0	1
$D_{0,1}$	0	0	0	1	1	1	1	1	1
$D_{0,2}$	1	1	1	1	0	0	1	1	1
$D_{0,3}$	1	1	1	0	0	0	1	0	0
$D_{0,4}$	1	0	0	0	1	0	0	1	0
$D_{0,5}$	0	0	1	0	0	1	0	0	1

Step 2: Compute the similarity matrix, take each row and check degree of similarity between other rows. For example, the 1st row is taken and similarity between the 1st row and the 2nd row is $b[3] \oplus b[4] \oplus b[5] \oplus b[8]$. Similarly, check similarity between the 1st and other rows. The similarities we get between 1st and other rows are

$$\begin{aligned} & b[3] \oplus b[4] \oplus b[5] \oplus b[8] \\ & b[1] \oplus b[3] \oplus b[8] \\ & b[1] \\ & b[4] \\ & b[5] \oplus b[8] \end{aligned} \quad (10)$$

Step 3: Repeat Step 2 for $D_{0,1}$, $D_{0,2}$, $D_{0,3}$, $D_{0,4}$ and $D_{0,5}$ and map all these similarities into a matrix form, called a similarity matrix as shown in Table 2.

Table 2. Similarity Matrix

Similarity	b[0]	b[1]	b[2]	b[3]	b[4]	b[5]	b[3]	b[4]	b[5]
$D_{0,0}$ & $D_{0,1}$	0	0	0	1	1	1	0	0	1
$D_{0,0}$ & $D_{0,2}$	0	1	0	1	0	0	0	0	1
$D_{0,0}$ & $D_{0,3}$	0	1	0	0	0	0	0	0	0
$D_{0,0}$ & $D_{0,4}$	0	0	0	0	1	0	0	0	0
$D_{0,0}$ & $D_{0,5}$	0	0	0	0	0	1	0	0	1
$D_{0,1}$ & $D_{0,2}$	0	0	0	1	0	0	1	1	1
$D_{0,1}$ & $D_{0,3}$	0	0	0	0	0	0	1	0	0
$D_{0,1}$ & $D_{0,4}$	0	0	0	0	1	0	0	1	0
$D_{0,1}$ & $D_{0,5}$	0	0	0	0	0	1	0	0	1
$D_{0,2}$ & $D_{0,3}$	1	1	1	0	0	0	1	0	0
$D_{0,2}$ & $D_{0,4}$	1	0	0	0	0	0	0	1	0
$D_{0,2}$ & $D_{0,5}$	0	0	1	0	0	0	0	0	1
$D_{0,3}$ & $D_{0,4}$	1	0	0	0	0	0	0	0	0
$D_{0,3}$ & $D_{0,5}$	0	0	1	0	0	0	0	0	0
$D_{0,4}$ & $D_{0,5}$	0	0	0	0	0	0	0	0	0

Step 4: Optimizes the similarity matrix.

Step 4-1: Sort the similarity matrix in ascending order based upon the number of 1s in each row.
Step 4-2: Delete duplicate rows and rows containing less than two 1's. This is depicted in Table 3.

Table 3. Optimized Similarity Matrix

	b[0]	b[1]	b[2]	b[3]	b[4]	b[5]	b[6]	b[7]	b[8]
$D_{1,0}$	0	0	0	0	0	1	0	0	1
$D_{1,1}$	0	0	0	0	1	0	0	1	0
$D_{1,2}$	0	0	0	1	0	0	1	1	1
$D_{1,3}$	0	0	0	1	1	1	0	0	1
$D_{1,4}$	0	0	1	0	0	0	0	0	1
$D_{1,5}$	0	1	0	1	0	0	0	0	1
$D_{1,6}$	1	0	0	0	0	0	0	1	0
$D_{1,7}$	1	1	1	0	0	0	1	0	0

Step 5: After replacing equations by optimized variables, we get four equations for $D_{0,0}$. We may represent $D_{0,0}$ by using any one of these four equations. $D_{0,1}$, $D_{0,2}$, $D_{0,3}$, $D_{0,4}$ and $D_{0,5}$ can be obtained using the same procedure as that of $D_{0,0}$.

In Step 5 we further check the size of the optimized similarity matrix rows that we obtained. If the number of optimized rows is not 0 or 1 then the process from Step 2 to Step 5 is repeated again on the optimized similarity matrix to further optimize the matrix.

Step 6: We get the optimized set of output equations.

$$\begin{aligned} O[0] &= D_{1,5} \oplus b[3] \\ O[1] &= D_{1,7} \\ O[2] &= D_{1,3} \\ O[3] &= D_{1,5} \oplus b[2] \\ O[4] &= D_{1,6} \oplus b[3] \\ O[5] &= D_{1,7} \oplus b[2] \end{aligned} \tag{11}$$

Compared with the output equations in Step 1, it can be observed the XOR gates are considerably reduced. In building expanded H-matrix utilizing spare columns which are left after production test, the logic minimization technique described in this section will be extensively applied.

2. Check Bit Addition Method

The space of H-matrices that provide SEC-DED capability is very large. As the number of message bits gets larger, an exhaustive search may no longer be possible. Hence it becomes important to derive a method that efficiently fills the extra rows of the H-matrix in a tractable amount of time. To narrow down the selection criteria, we add extra rows on the basis of similarity between the pre-determined rows of the H-matrix. Moreover, the H-matrix must satisfy the following conditions;

1. There are no all 0 columns.
2. Every column is distinct.
3. The total number of 1's in the H-matrix should be a minimum.
4. The number of 1's in each row of the H-matrix is equal or as close as possible to the average number (total number of 1's in H-matrix divided by the number of rows).
5. All the 1's in the extra row depict similarity with the 1's in the pre-existed rows.

Condition 1 ensures that no single bit error case matches the error free case ensuring non-zero error syndrome. Condition 2 ensures that the syndromes of all single bit errors are unique. Every single error syndrome matches one of the columns of the H-matrix. Since all the columns of the H-matrix are distinct, single bit errors are uniquely identified and hence corrected. Condition 2 also ensures that double bit errors are detected.

Conditions 1, 2 and 3 collectively ensure the H-matrix that is selected should be such that if no spare columns are available and no extra row is added, it still retains the SEC-DED property.

Conditions 3 and 4 ensure that the code requires less hardware for implementation. Thus, it guarantees lower cost and better reliability. Furthermore, a balanced number of 1's in each row of the H-matrix minimizes the delay of the H-matrix (the delay is constrained by the maximum weight row).

As the number of message bits gets larger, however, then an exhaustive search to find the best combination of 1's and 0's for the extra row is no longer possible. Condition 5 ensures that the countable number of combinations, to develop an extra row, on the basis of similarity of logic between the pre-existed rows are searched. The MP is computed, and the combination that minimizes the MP is then selected for the extra row.

We know that each row in the H-matrix represents a linear equation involving the bits of the message. Fig. 4 shows a 16 bit Hsiao code with an extra row to be added. Our mission is to fill the spare row with a combination of 1's and 0's providing minimal area as well as MP in a tractable time. If simply the MP is considered, the best way is to exhaustibly calculate the MP for all possible 2^{16} cases, however area optimization is not guaranteed. More importantly the exhaustive calculation is not possible for

$$H=\begin{pmatrix} 1&1&1&1&1&1&0&0&0&0&1&0&0&0&1&0\\ 1&1&1&0&0&0&1&1&1&1&0&0&1&0&0&0\\ 1&0&0&0&1&0&1&1&1&0&0&0&0&1&1&1\\ 0&0&0&0&0&1&1&0&0&1&1&1&0&1&1&1\\ 0&1&0&1&0&0&0&1&0&1&1&1&1&1&0&0\\ 0&0&1&1&1&1&0&0&1&0&0&1&1&0&0&1\\ & a & & & b & & & c & & & d & & f & & e & \end{pmatrix}$$

Fig. 4. Hsiao Matrix with a Spare Row.

$$H=\begin{pmatrix} 1&1&1&1&1&1&0&1&0&0&1&0&0&0&0&0\\ 1&1&1&0&0&0&0&0&1&0&0&0&1&1&1&1\\ 1&0&0&0&1&0&0&0&0&1&1&1&1&1&1&0\\ 0&0&0&0&0&1&1&1&1&1&1&1&0&1&0&0\\ 0&1&0&1&0&0&1&1&1&1&0&0&0&0&1&1\\ 0&0&1&1&1&1&1&0&0&0&0&1&1&0&0&1\\ & a & & & b & & & e & & & d & & & c & & f \end{pmatrix}$$

Fig. 5. Hsiao Matrix after rearranging.

the message whose length is like 64.

In this paper we try to find the similarity between the positions of 1's within the rows of the H-matrix, which can provide maximal logic sharing in implementing H-matrix linear equations. In Fig. 4, by keeping the similarity of logic between the rows of H-matrix in mind, the spare row is to be filled with chunk of 1's and 0's which gives the least MP. An example is depicted to show how the extra row on the basis of similarity of logic is achieved satisfying condition 5.

We highlight the similarity of logic in terms of 1's between the rows of the H-matrix. We then number these highlighted similarities from *a* to *e*, in this case, to find the best combination for the extra row. Each number depicts a chunk of three 1's or 0's in this example. Now we check all the 2^5 combinations to find the best combination that produces the minimum MP, instead of 2^{16} for exhaustive search. Similarly same procedure can be followed to fill other spare row if available. The proposed method plays a critical role in managing the calculation time while minimizing the area and miscorrection ratio when message bits get larger or there is more number of extra rows to be added to the H-matrix.

3. Local Augmentation of Miscorrection Probability

So far spare rows have been filled with chunk of 1's and 0's in a tractable time to minimize the area overhead while providing a reduced MP. Since all the solution space is not exhaustively searched, the minimization of the MP is not guaranteed, thus a local augmentation (LA) algorithm is proposed. For an assignment to a spare row, the bits located between 1's and 0's chunks are targeted to be flipped for possibly augmented MP while maximally preserving the area optimization.

The MP relies on the structure of the H-matrix, but is independent of the sequence of each column. In other words, the MP is not changed by rearranging the columns. Therefore two important properties not requiring expensive calculation time can be observed in H-matrix which does not change the MP.

1. The sequence of chunks of columns can be changed
2. The sequence of columns in each chunk can also be changed.

Starting from the current local optimal assignment on extra rows, we try to further augment the MP by locally rearranging the chunks and columns according to the above properties. In Fig. 4, it can be observed that the neighboring two columns between chunks (*b*,*c*), (*c*,*d*), and (*f*,*e*) include only one (11) cluster, and (*a*,*b*), (*d*,*f*) include two (11) clusters. In Fig. 5 obtained by rearranging the chunks of columns, it can be seen that all neighbors of (*a*,*b*), (*b*,*e*), (*e*,*d*), (*d*,*c*), and (*c*,*f*) include two (11) clusters. Since both H-matrices in Fig. 4 and Fig. 5 are isomorphic, they require the same circuit area and result in the same MP. Instead of assigning all the *a*-*f* chunks as 1s or 0s, the constraint is relaxed such that the neighboring two cells are allowed to be (11) or (00) for possible improvement on the MP with the hope that the area is not at least increased further. For example chunks *a* and *b* which were assigned as (000/000) are assigned as (00/11/00). Section IV shows the experimental results with significant improvement.

4. Spare Memory Architecture

The use of a large number of MUXes at the input and output of the memory may be a burden in real implementation [9]. To enhance the ECC, we implement the

additional check bits using a fuse technology by which the spare cells are reconfigured through BISR during the production test stage.

The block diagram of the proposed spare memory architecture is shown in Fig. 6. Block A in Fig. 6 consists of the fuse architecture used to improve the performance of the whole architecture. The internal architecture of the block A is shown in Fig. 7 and Fig. 8 as described in the patents [12, 13]. If the flag fuse in Fig. 8 is burned, it suggests that the spare is used for repair. Whereas, if the fuse is intact and disables the EN signal then the spare can be used to store the check bit. The output EN signals from the block A are used as the control signal for the MUXes placed between the memory and check bit generator.

The control signals for the MUXes will be '1' if the spare is used for repair or if 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. 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.

In the proposed architecture it is made certain that the spare column associated with the highest check bit is

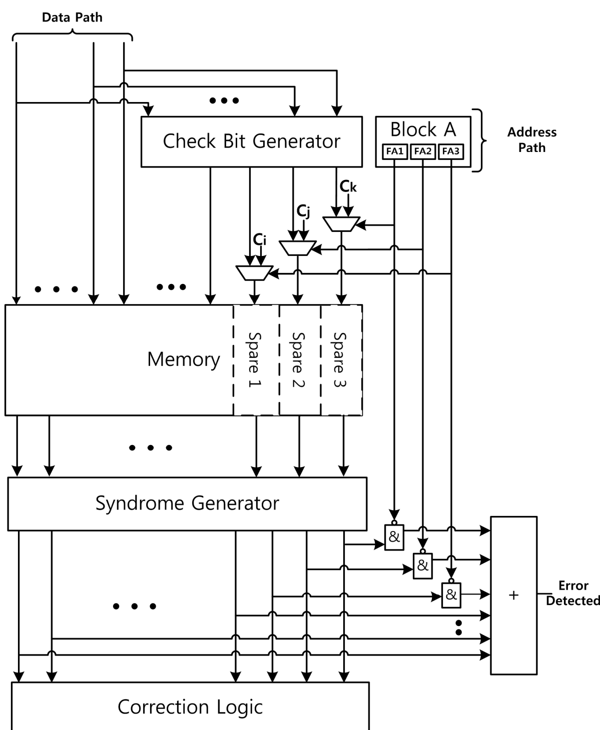

Fig. 6. Block diagram of proposed spare memory architecture.

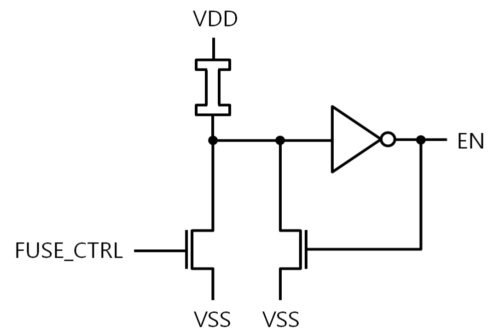

Fig. 7. Fuse architecture.

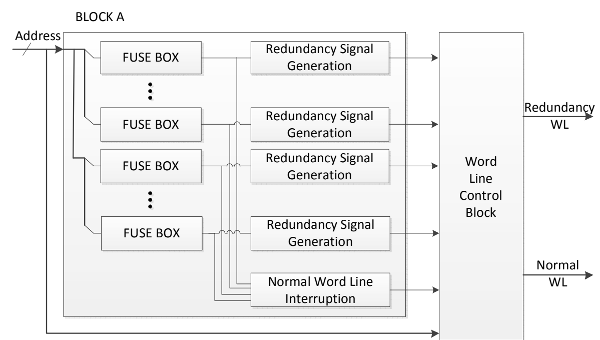

Fig. 8. Internal architecture of the block A.

used first for repair. If, for example, there are three spare columns then the third spare column should be used for repair before the second and first. In this way the remaining spare columns that are left unused may be used to store check bits. Different from the architecture [9] using MUXes for all outputs as well as all inputs of the memory block, which are used to redirect the data paths for faulty columns to repair columns, only the inputs for the spare columns are connected to MUXes in our technique. In production testing the faulty columns are replaced by spare columns using laser fuses in wafer level or electrical fuses in package level [13]. In memory read and write operations, the address is compared with the fused repair addresses of the Fig. 8, and if matched, then the corresponding spare column is chosen for the operation instead of faulty data column. Therefore if our technique is implemented for the memory using fuse based repair [12, 13], as shown in Fig. 6 only inputs for spares need to be attached with MUXes. The data outputs of the memory which are internally redirected by spare columns drive "n" read values in order externally, hence the data inputs of the syndrome generator are directly connected to the outputs. And the spare columns of the

memory, which are not externally available but internally available, are also directly connected to the syndrome generator which is designed to take variable number of check bits.

Our architecture clearly suggests a significant reduction in the area overhead caused by the use of large number of MUXes in the implementation to the fuse based memories.

IV. EXPERIMENTAL RESULTS

For our experiment, we selected common data sizes to reflect the performance of the proposed methods. For the proposed schemes, the results are shown for the case where one, two, and three spare columns are available for repair. At first H-matrices with one spare, two spares, and three spares respectively are generated according to the proposed check bit addition method. Then, the proposed methods are applied to the newly created H-matrices, producing an optimized set of equations. Which are then implemented with Verilog HDL.

Table 4 shows the best results of [9]. Table 5 shows the results using the CBA proposed. Table 6 shows the results by applying LA and CBA. In our proposed method, LA uses number of chunks to find out combinations (timing required) required to compute the output. To compute combinations, we used chunks in three ways, i.e. by converting them to all 0's, all 1's and without any change. For LA, if number of chunks is n, then the total combinations required to compute output will be 3^n, which is approximately equal to $2^{1.6*n}$.

In our proposed method, combinations are the sum of combinations for LA and CBA, but the combinations for CBA are so small as compared to combinations for LA, that it is negligible. In Table 6, for 64 bit data number of chunks "n" is 15, which implies that combinations required to compute output will be equal to $2^{16}+3^{15}$, which is approximately equal to $2^{16}+2^{24}$, but the combinations for CBA i.e. 2^{16} is so small as compared to combinations for LA i.e. 2^{24}, that it can be neglected.

The results of Table 6 are compared with Table 4 and are displayed in Table 7. For each code, the number of 2-input XOR gates and the triple bit error MP is shown along with the combinations required to establish a particular extra row. As can be seen, not only the number of XOR gates but also the number of combinations to formulate the best combination of row is reduced considerably. It is worth noticing that the number of spares as well as the size of the matrices enables the code to detect nearly all the triple errors thus reducing the MP to the minimum.

Table 7 compares the area, timing complexity, and MP of Table 4 with Table 6 for 64 bit data. It can be seen that the area overhead as the number of XOR gates are significantly improved for all the cases. Our CBA followed by the logic sharing contributes to reduce the area as well as the searching combinations but some increase in MP as shown in Table 5. Additionally by applying the LA, which aims to minimize MP with minor sacrifices on searching combinations and area, spares can be added efficiently as shown in Table 7. The timing complexity which depends on the number of combinations to check the MP becomes drastically reduced as 2^{24} in our approach than the 2^{64} which is almost impossible with current computing systems. Also it can

Table 4. Results of triple-Error Miscorrection Probability for Previous Research [9]

Data bits	1 Spare			2 Spares			3 Spares		
	# of XORs	Combinations (Timing required)	MP (%)	# of XORs	Combinations (Timing required)	MP (%)	# of XORs	Combinations (Timing required)	MP (%)
16	58	2^{16}	25.3	70	$2^{16}*2$	8.7	76	$2^{16}*3$	2.3
32	118	2^{32}	25.8	129	$2^{32}*2$	11.3	138	$2^{32}*3$	5.1
64	265	2^{64}	26.0	308	$2^{64}*2$	14.1	351	$2^{64}*3$	11.0

Table 5. Results of triple-Error Miscorrection Probability using Check Bit Addition

Data bits	1 Spare			2 Spares			3 Spares		
	# of XORs	Combinations (Timing required)	MP (%)	# of XORs	Combinations (Timing required)	MP (%)	# of XORs	Combinations (Timing required)	MP (%)
16	54	2^{5}	27	60	$2^{5}*2$	11	66	$2^{5}*3$	5.5
32	112	2^{10}	23.8	127	$2^{10}*2$	10.7	138	$2^{10}*3$	4.7
64	212	2^{16}	26.5	246	$2^{16}*2$	12.6	274	$2^{16}*3$	6.3

Table 6. Results of triple-Error Miscorrection Probability using Proposed Method

Data bits	1 Spare			2 Spares			3 Spares		
	# of XORs	Combinations (Timing required)	MP (%)	# of XORs	Combinations (Timing required)	MP (%)	# of XORs	Combinations (Timing required)	MP (%)
16	41	$< 2^{9.6}$	26.9	43	$< 2^{9.6} * 2$	10.0	45	$< 2^{9.6} * 3$	4.7
32	84	$< 2^{16}$	23.8	88	$< 2^{16} * 2$	10.3	94	$< 2^{16} * 3$	4.5
64	171	$< 2^{24}$	26.0	179	$< 2^{24} * 2$	12.2	187	$< 2^{24} * 3$	5.7

Table 7. Improvement of Area, Timing Complexity and Miscorrection Ratio for 64 data bits

64 Data bits	1 Spare			2 Spares			3 Spares		
	# of XORs	Combi-nations (Timing required)	MP (%)	# of XORs	Combi-nations (Timing required)	MP (%)	# of XORs	Combi-nations (Timing required)	MP (%)
Original Method[9]	265	2^{64}	26.0	308	2^{64}*2	14.1	351	2^{64}*3	11.0
Proposed Method	171	$< 2^{24}$	26.0	179	$< 2^{24} * 2$	12.2	187	$< 2^{24} * 3$	5.7
Improved	35.5	2^{40} times	0	41.9	2^{40} times	1.9	46.7	2^{40} times	5.3

Table 8. Comparison of Non-Adjacent Double-Error Miscorrection Probability with Previous Research [9]

Data bits	Previous Research [9]						Proposed Method					
	1 Spare		2 Spares		3 Spares		1 Spare		2 Spares		3 Spares	
	# of XORS	MP (%)	# of XORS	MP (%)	# of XORS	MP (%)	# of XORS	MP (%)	# of XORS	MP (%)	# of XORS	MP (%)
16	57	18.2	66	3.2	70	0.0	46	20.8	50	7.9	56	2.2
32	117	27.4	130	13.8	140	8.8	84	24.0	91	10.4	100	4.0
64	263	26.9	306	17.8	353	14.6	188	24.7	201	11.9	224	5.5

be seen that the miscorrection ratios are improved up to 5.3% in our approach. Note that the efficiency of the proposed methods is heavily dependent upon the selection of the H-matrix type.

Non-adjacent double-error (NADE) miscorrection ratios and areas are analyzed in Table 8. Similar as triple error cases, less MPs are achieved for the NADE with reduced area overheads by applying our technique. Since our approach does not exhaustively search all the combinations, it can be noted that the MPs for 16 bits data are slightly increased than [9] as in Table 4, Table 5, Table 6, and Table 8.

V. CONCLUSIONS

This paper proposes a method to efficiently fill the extra rows of the H-matrix. Especially for the increased number of spare rows and data size, the results depict a significant reduction in calculation time and area overhead. Instead of using MUXes for all outputs as well as all inputs of the memory block, only the inputs for the spare columns are connected to MUXes in the implementation to the fuse based memory. Optimization of the whole H-matrix is accomplished through logic sharing resulting in the reduced area overhead while keeping the miscorrection ratio relatively low.

ACKNOWLEDGMENTS

This research was supported in part by the National Research Foundation of Korea (NRF) grant (MEST) (No. 2010-0026822).

REFERENCES

[1] U. Schlichtmann, “Tomorrows high-quality SoCs require high-quality embedded memories today”. *In International Symposium on Quality Electronic Design*, Mar., 2002.

[2] J. I. Park, et al, “High-Speed Low-Complexity Reed-Solomon Decoder using Pipelined Berlekamp-Massey Algorithm and Its Folded Architecture,” *Journal of Semiconductor Technology and Science*, pp. 193-202, Vol. 10, No. 3, Sep., 2010.

[3] R. Hamming, “Error Correcting and Error Detecting Codes”, *Bell Sys. Tech. Journal*, Vol. 29, pp.147-160, Apr., 1950.

[4] M. Y. Hsiao, “A Class of Optimal Minimum Oddweight-column SEC-DED codes”, *IBM Journal of Research and Development*, Vol. 14, pp. 395-401, 1970.

[5] M. Richter, et al, “New Linear SEC-DED Codes with Reduced Triple Error Miscorrection Probability,” *Proc. of International On-Line Testing Symposium*, pp. 37-42, 2008.

[6] I. Kim, et al, “Built In Self Repair for Embedded High Density SRAM,” *Proc. of International Test Conference*, pp. 1112-1119, 1998.

[7] Y. Zorian, et al, “Embedded-Memory Test and Repair: Infrastructure IP for SOC Yield,” *IEEE Design & Test of Computers*, Vol. 20, Issue 3, pp. 58-66, May 2003.

[8] W. Jeong, et al, “An Advanced BIRA for Memories with and Optimal Repair Rate and Fast Analysis Speed by Using a Branch Analyzer,” *IEEE Transactions on Computer Aided-Design*, Vol. 29, No. 12, pp. 2014-2026, Dec. 2010

[9] R. Datta, et al, “Exploiting Unused Spare Columns to Improve Memory ECC,” *VLSI Test Symposium, 27th IEEE*, pp. 47-52, 2009.

[10] W. Peterson, et al, Error Correcting Codes, *MIT Press, Cambridge, MA*, 1972.

[11] D. K. Pradhan, Fault-Tolerant Computer System

Design, *Prentice Hall, Upper Saddle River, NJ*, 1996.

[12] Y. K. Kim, et al, “Redundancy fuse control circuit and semiconductor memory device having the same and redundancy process method,” *U. S. Patent 7,184,331*, Oct. 27, 2005.

[13] S. H. Kang, “Redundancy circuit in semiconductor memory device,” *U.S. Patent 7,257,037*, Nov. 2, 2006.

Jihun Jung received the B.S. in computer science and engineering from Hanyang University, Gyunggi - do, Korea in 2010. Since 2010 he has been working toward the M.S. and Ph.D. degree in computer science and engineering at the same University. His interests include Design for Testability, Memory Test, Memory ECC, 3D IC Test, and NoC Design.

Umair Ishaq received the M.S. in computer science and engineering from Hanyang University, Gyunggi - do, Korea in 2011. Since 2011 he has been working for Horizon Tech., Lahore, Pakistan. His interests include Design for Testability, Memory Test, and Memory ECC.

Jaehoon Song received the B.S., M.S., and Ph.D. degrees in computer science and engineering from Hanyang University, Gyunggi-do, Korea in 2000, 2002, and 2009 respectively. Since 2009 he has been working for TranSono Inc., Seoul, Korea. In 2003, he worked for the System-on-a-Chip (SoC) Design Center at Seoul National University in Korea, where he was on the Development Staff in charge of platform-based design. His main research interests are in Design-for-Testability (DfT), signal integrity, and low-power design. Mr. Song is a member of the Institute of Electronics Engineers of Korea and the Korea Information Science Society. He received the Best Paper Award from the Korea Test Association at the Korea Test Conference in 2007.

Sungju Park received the B.S. degree in electronics from Hanyang University, Korea, in 1983 and the M.S and Ph.D. degrees in electrical and computer engineering from the University of Massachusetts at Amherst in 1988 and 1992, respectively. From 1983 to 1986, he was with the Gold Star Company in Korea. From 1992 to 1995, he worked for IBM Microelectronics, Endicott, NY as a Development Staff in charge of boundary scan and LSSD scan design. Since then, he has been a Professor in the department of computer science and engineering in Hanyang University, Korea. His research interests lie in the area of VLSI testing including scan design, built-in self test, test pattern generation, fault simulation, and synthesis of test. Additional interests include graph theory and design verification. Prof. Park is a member of IEEE, the Institute of Electronics Engineers of Korea, the Korea Information Science Society, and the Institute of Electronics and Information and Communication Engineers.