A Very Efficient Redundancy Analysis Method Using Fault Grouping

Hyungjun Cho, Wooheon Kang, and Sungho Kang

To increase device memory yield, many manufacturers use incorporated redundancy to replace faulty cells. In this redundancy technology, the implementation of an effective redundancy analysis (RA) algorithm is essential. Various RA algorithms have been developed to repair faults in memory. However, nearly all of these RA algorithms have low analysis speeds. The more densely compacted the memory is, the more testing and repair time is needed. Even if the analysis speed is very high, the RA algorithm would be useless if it did not have a normalized repair rate of 100%. In addition, when the number of added spares is increased in the memory, then the memory space that must be searched with the RA algorithms can exceed the memory space within the automatic test equipment. A very efficient RA algorithm using simple calculations is proposed in this work so as to minimize both the repair time and memory consumption. In addition, the proposed algorithm generates an optimal solution using a tree-based algorithm in each fault group. Our experiment results show that the proposed RA algorithm is very efficient in terms of speed and repair.

Keywords: Redundancy analysis (RA) algorithm, repair rate, normalized repair rate, automatic test equipment (ATE).

Manuscript received July 17, 2012; revised Oct. 7, 2012; accepted Nov. 6, 2012.

This work was supported by a grant from the National Research Foundation of Korea (NRF) funded by the Korean government (MEST) (No. 2010-0024707).

Hyungjun Cho (phone: +82 2 2123 2775, chj0937@soc.yonsei.ac.kr), Wooheon Kang (sudal@soc.yonsei.ac.kr), and Sungho Kang (corresponding author, shkang@yonsei.ac.kr) are with the Department of Electrical & Electronic Engineering, Yonsei University, Seoul, Rep. of Korea.

http://dx.doi.org/10.4218/etrij.13.0112.0467

I. Introduction

As the density of the memory in a device increases, the number of faults also increases. Therefore, to increase the memory yield, many manufacturers use incorporated redundancy to replace faulty cells. In this redundancy technology, the implementation of an effective redundancy algorithm is essential [1].

An effective redundancy algorithm stipulates the manner in which to replace failed bits with spare elements after the bit failure patterns of the die configurations are determined. External automatic test equipment (ATE) is primarily used to test semiconductor memory. The ATE first gathers fault information and provides the repair solution using a redundancy algorithm. To repair faulty memory, spare elements are used at different levels of the design hierarchy. The ATE should provide a very high repair rate, and a repair solution should be generated in a timely manner.

Many researchers have shown that the redundancy algorithm is nondeterministic polynomial time complete. Therefore, Day proposed an exhaustive search algorithm that was capable of generating all possible solutions [2]. However, since this algorithm generates an entire tree for all possible solutions (repair row/repair column), it has a very long search time. Consequently, various early-termination methods have been proposed [3]-[6]. If the reparability is determined before the repair analysis, the amount of analysis needed to create an exhaustive tree search can be reduced. However, these early-termination methods for reducing the analysis time are not complete solutions in cases of large memory amounts. As a result, redundancy analysis (RA) algorithms that are based on exhaustive binary search trees require a long search time for nearly all serious cases. Therefore, the FAST RA algorithm [7]

was proposed to reduce the analysis time. The FAST RA algorithm can generate a solution in a short time, but, although the FAST RA algorithm should be able to repair all reparable memory, the solution is not optimal.

Recently, a memory testing methodology using built-in self-repair (BISR) was suggested [8]-[15]. However, due to the large hardware overhead associated with BISR, a lot of memory is fabricated. For this reason, ATE with RA algorithms is extensively employed to test and repair memory. The simultaneous testing and repair of a lot of memory for mass production requires a significant amount of time. Therefore, the analysis speed is very important in RA algorithms for ATE, and manufacturers should minimize the time for testing/repairing.

Three-dimensional (3D) memory using through silicon via (TSV) has been presented recently [16]-[19]. Also, the capacity of a 3D memory can be extended by increasing the number of stacked dies. However, to guarantee the yield and the quality of the 3D memory, the RA algorithm rapidly finds an optimal solution for repairing the 3D faulty cells.

In this paper, the very efficient RA (VERA) algorithm, which has a 100% normalized repair rate, is proposed. In the analysis, a bitmap that stores information on faulty cells and indicates the grouped addresses is utilized. Because the fault groups are independent of each other, the time needed to find solutions can be greatly reduced. Therefore, the proposed algorithm determines an optimal solution for repairing memory faults in a short amount of time. It is also the fastest and easiest algorithm to develop or implement in ATE tests. Furthermore, the proposed algorithm uses a small amount of memory to solve the spare location problem.

II. Background

An RA algorithm is generally divided into two phases: the must-repair RA and the final-repair RA [2], [3]. While the must-repair RA cannot provide complete solutions, it does yield inevitable redundancy solutions prior to the final analysis. The final-repair RA algorithm analyzes the fault bitmap and determines the reparability. If the targeted memory is reparable, then the final-repair RA algorithm outputs the addresses to be repaired.

1. Definition of Repair Rate and Normalized Repair Rate

The repair rate indicates the ability of an RA algorithm to find a correct repair solution [13], [14]. The definitions of the repair rate and the normalized repair rate are as follows:

$$\text{Repair rate} = \frac{\text{\# of repaired chips}}{\text{\# of total tested chips}},$$

$$\text{Normalized repair rate} = \frac{\text{\# of repaired chips}}{\text{\# of reparable chips}}.$$

The total number of tested chips includes the number of irreparable chips. The repair rate is influenced by the number of chips that cannot be repaired using redundant cells. Irreparable chips are caused by many factors, including process variations, design-related faults, and human error. Variations that arise as a result of these unknown factors degrade the effectiveness of the RA algorithm, causing it to have a low repair rate. However, the normalized repair rate is independent of these variations. Therefore, the normalized repair rate is more appropriate for estimating the ability of an RA algorithm to obtain a correct repair solution.

2. Existing RA Algorithms

The repair-most (RM) algorithm [12] is a greedy algorithm. While the RM algorithm is not an optimal solution, it is very simple and has thus been adopted as an RA algorithm in many cases.

Kuo and Fuchs proposed the branch-and-bound algorithm, which is a heuristic method based on a binary search tree [3]. Although the branch-and-bound algorithm [2] is faster than the exhaustive binary search algorithm, it does not significantly reduce the time or memory space needed to search for a solution. As such, various early-termination methods for reducing the analysis time have been proposed [3]-[5]. However, early-termination methods for reducing the analysis time cannot be considered to be absolute solutions in cases of large amounts of memory. "On the Repair of Redundant RAMs" [6] proposed a feasibility-check procedure called Quick-Terminate as well as an exhaustive search algorithm to obtain an optimum cost solution. The Quick-Terminate method is one of the early-termination methods using the maximum-matching technique, but it does not result in complete solutions. To address this problem, Hemmady and Reddy performed this early-termination method to reduce the running time before executing a final exhaustive binary algorithm. According to their method, a binary search algorithm such as the branch-and-bound algorithm should be employed after performing the Quick-Terminate method.

The intelligent solve (IS) and intelligent solve first (ISF) algorithms [8], the minimized binary search tree algorithms [11], BRANCH [9], and ReBIRA [15] are BISR algorithms that use a binary search tree structure. IS and ISF each have an optimal repair rate, but their RA speeds are much slower than the RA speeds of other binary search algorithms. The algorithm presented in [11] is a BISR algorithm that uses a minimized binary search tree to reduce the search time. BRANCH

analyzes all nodes of a branch for the binary search tree in parallel. As a result, BRANCH enables the algorithm to find the solution faster than ISF. ReBIRA was proposed to reduce the hardware cost for BIRA. However, due to the large hardware overhead associated with BISR, a lot of memory is generated. For this reason, ATE with RA algorithms is used to test and repair memory. PAGEB [20] creates a solution that transforms a spare allocation problem into Boolean functions. A binary decision diagram is then used to manipulate the functions. PAGEB consumes less memory space and requires less time compared to the branch-and-bound methods. However, because PAGEB determines the defect function (DF), constraint function (CF), and repair function (RF), it is time-consuming and requires a significant amount of memory space. The DF is a Boolean function that encodes the locations of all faulty cells, while the CF is a Boolean function that encodes all combinations of faulty lines that are replaceable with spare lines. The RF is a Boolean function that encodes all repair solutions for a space allocation problem. Therefore, PAGEB must calculate the RF generated by the Boolean-AND operation between the stored DF and CF.

The FAST algorithm [7] is the fastest RA algorithm and is very simple; however, it does not have a 100% normalized repair rate. Therefore, an RA algorithm that can produce an optimal solution in a reasonable time should be developed.

In this work, VERA using grouped faults is compared with the above RA algorithms with 100% normalized repair rates. The proposed scheme determines a solution for repairing memory faults with a time of nearly zero. Moreover, the proposed algorithm can find an optimal solution. VERA determines a solution using a tree-based algorithm in each fault group. Since the proposed algorithm only requires the memory space for one fault group, it can also greatly reduce the amount of memory used.

III. Proposed RA Algorithm

1. Grouping of Faults in Memory

The VERA algorithm creates fault groups that are independent of each other. The fault bit map and fault group information are simultaneously generated during the detection of faults by the ATE.

Definition (Fault Group). A fault group is defined as a set containing a single fault or multiple faults. Let $F(r_1, c_1)$, $F(r_2, c_2)$, …, $F(r_n, c_n)$ denote n faulty cells, while r_k and c_k denote the row and column address of the k-th faulty cell, respectively.

1) $F(r_k, c_k)$ is categorized into a single-fault group if $r_k \neq (r_m$ of any other faulty cell, $F(r_m, c_m))$ and $c_k \neq (c_m$ of any other faulty cell, $F(r_m, c_m))$.
2) $F(r_k, c_k)$ and $F(r_m, c_m)$ are categorized into a multi-fault group if $r_k = (r_m$ of any other faulty cell, $F(r_m, c_m))$ or $c_k = (c_m$ of any other faulty cell, $F(r_m, c_m))$.

The proposed algorithm provides a solution for repairing faulty memory using the following fault group properties.

Property 1. In a multi-fault group, there are no faults that have the same row or column address as faults in other fault groups.

Property 1 implies that a multi-fault group is independent of any other fault group, regardless of its status as a single-fault group or a multi-fault group. In other words, the proposed algorithm regards a fault group as an independent problem. As such, the proposed algorithm can provide repair solutions in each fault group irrespective of the solutions for other groups in the remaining spare cells.

Property 2. The single-fault groups are repaired by the remaining redundancy cells after the faults in the multi-fault groups are repaired.

The single-fault groups can be repaired using the spare row and column cells (R_S and C_S) remaining after other faulty cells are repaired. Therefore, the proposed algorithm repairs the single-fault groups after repairing the multi-fault groups.

Property 3. If the number of groups is greater than $R_S + C_S$, then the memory cannot be repaired (early-termination solution).

At least one spare row or column cell is needed to repair the faults in a fault group. To repair a single-fault group, only one redundancy cell is needed. However, to repair a multi-fault group, one or more redundancy cells are required. Therefore, if the number of groups is the same or less than $R_S + C_S$, the RA algorithm should determine whether or not the memory is reparable. This way, irreparable memory can be found early and terminated before the RA algorithm is performed. Property 3 is applicable to other RA procedures that occur prior to the RA algorithm search for solutions. An example of Property 3

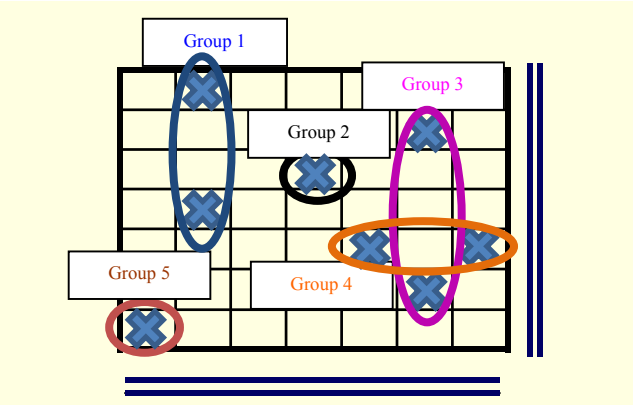


Fig. 1. Example: Property 3 (# of fault groups > $R_S + C_S$).

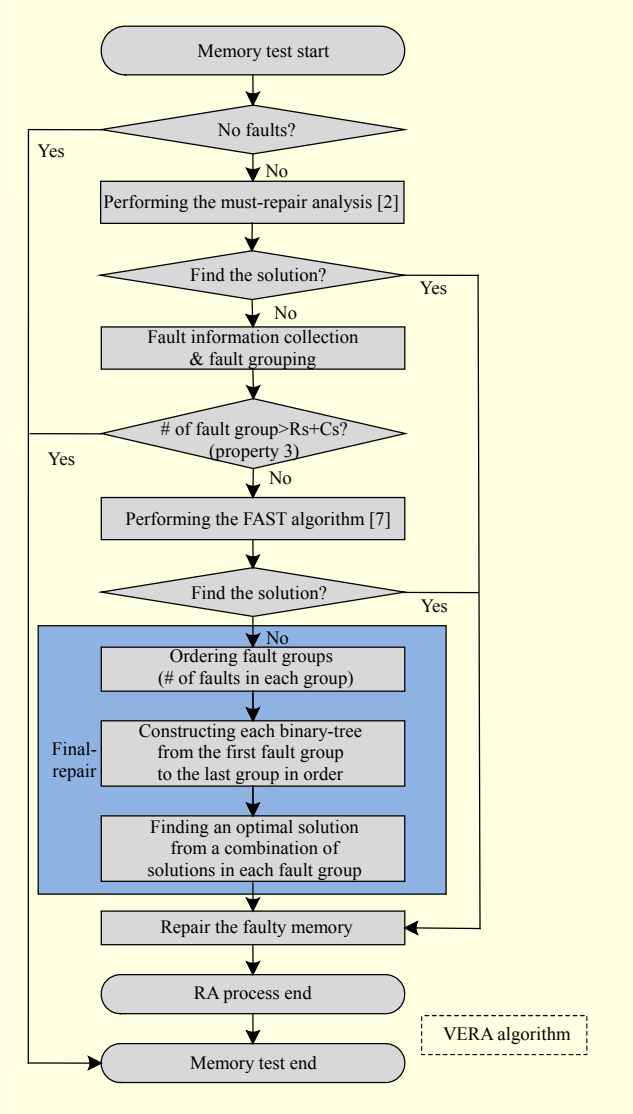


Fig. 2. Flow chart of proposed algorithm.

with five fault groups greater than $R_S + C_S$ (where R_S: 2 + C_S: 2) is shown in Fig. 1. In this case, the memory cannot be repaired, but the must-repair algorithm is unable to determine whether or not the memory is reparable.

2. Proposed VERA Algorithm

A flow chart of the VERA algorithm is shown in Fig. 2. If any faults are found, then the VERA algorithm is performed. In the first step, our approach is similar to that of previously proposed must-repair analysis [2], [3]. The must-repair analysis begins by comparing the number of faulty cells in a row or a column to the number of available redundant columns or rows. If the number of faulty cells in a row or column is greater than the number of available rows or columns, then the line becomes a must-repair line, and a redundant row or column line is used to repair that line. This is the must-repair rule. If the must-repair method cannot find a solution, the information of the faulty cells is collected, and each fault is classified as part of a single-fault group or a multi-fault group.

For the next step, the number of groups and remaining redundant cells are analyzed to see if they satisfy Property 3. If the number of fault groups is greater than the number of redundant cells ($R_S + C_S$), the faulty memory cannot be repaired. Therefore, the use of this early-termination solution allows irreparable memory to be terminated prior to the performance of an RA algorithm. If Property 3 is not satisfied, the faulty memory can be repaired, and the next step, which is performing the FAST algorithm to determine the solution in the shortest amount of time, is conducted. For the analysis of the FAST algorithm, the distribution of faults is studied to see if it satisfies the condition of the FAST algorithm.

The FAST algorithm generates a solution using Properties 1 and 2 of the fault groups. It also determines the number of faulty row or column lines in all of the multi-fault groups. Subsequently, the FAST algorithm combines the number of faulty row or column lines in all of the multi-fault groups. As a result, a solution is generated using a combination of the number of faulty row or column lines, excluding the number of single-fault groups. The sum of the number of row or column lines of the solution and the number of single-fault groups must be less than the number of remaining redundancy cells. Finally, the FAST algorithm repairs the single-fault groups using the remaining redundancy cells.

The VERA algorithm, if it cannot repair the memory using the FAST algorithm, then moves on to the final-repair phase to determine an optimal solution. The final-repair algorithm also uses the fault groups and has a 100% normalized repair rate to construct a tree to search for solutions according to Property 1. The fault groups are first ordered according to the number of faults in each fault group. Trees are then sequentially constructed using the branch-and-bound algorithm from the first fault group to the last fault group. Since trees are generated in each ordered fault group, memory space is only needed for the first, largest fault group. Since the final-repair algorithm constructs binary trees in each group, the depths of the trees are much smaller than those in other tree-based algorithms having full depths. Therefore, the time needed to determine an optimal solution and the amount of memory space used for trees are reduced. The generated solutions that do not exceed the number of spare rows and spare columns are stored in the memory space. Finally, the final-repair algorithm adopts an optimal solution from the combination of all possible solutions in each fault group.

Because of this, the final solution consumes a minimum number of spare rows and spare columns. Unlike the FAST

```
Algorithm Final-Repair

/* ordering */
begin
   queue = fault groups;
   while (fault groups left and the queue not empty) {
      i = # of faults in maximum group from the queue;
      j = # of faults in first group from the queue;
      group tmp = group j;
      group j = group i;
      group i = group tmp;
      remove the first group from the queue;
   }
end

/*Finding an optimal solution */
begin
   queue1= fault groups;
   while (fault groups left and the queue1 not empty) {
      queue2 = solutions;
            while (solutions left and the queue2 not empty) {
            if (RS >= R + # of used spare row of this queue2)
            R = R + # of used spare row of this queue2;
            if (CS >= C + # of used spare column of this queue2)
            C = C + # of used spare column of this queue2;
                        combined solutions are stored;
            remove the first record from the queue2;
            }
            remove the first record from the queue1;
   }
   if (the queue1 is empty and RS >= R and CS >= C)
   an optimal solution is selected ;
   else fail;
end
```

Fig. 3. Pseudocode of final-repair algorithm (ordering and finding optimal solution).

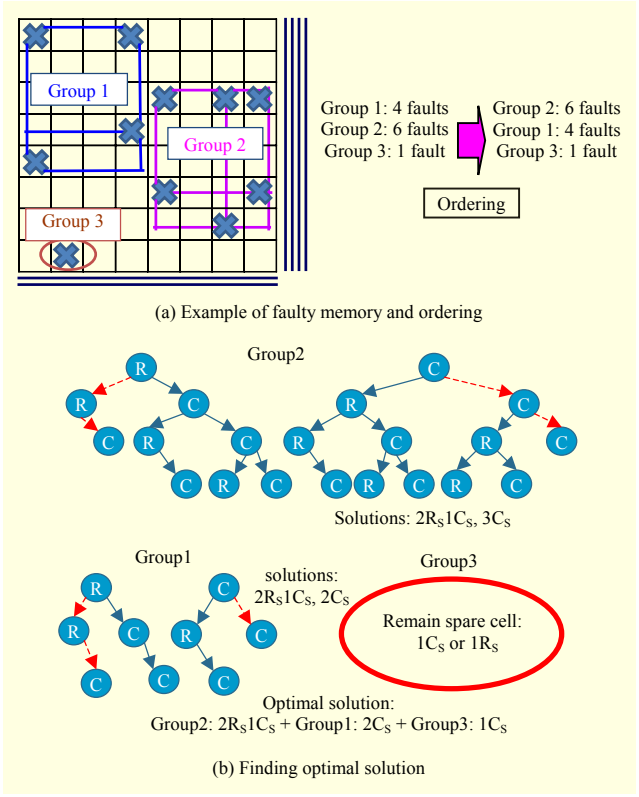


Fig. 4. Flow chart of proposed algorithm.

Table 1. Distribution of defect types.

Fault type	Distribution	
	D1 (%)	D2 (%)
Single faulty cells	50-60	30-40
Line faulty cells (row/column)	10-20/10-20	15-25/15-25
Rectangle of faulty cells	5-10	10-15

algorithm, the final-repair algorithm finds all solutions in each fault group and generates an optimal solution. Figure 3 shows the pseudocode of the final-repair algorithm. Since the algorithm for constructing trees in each group is similar to that in the branch-and-bound algorithm, it is omitted.

An example of searching for a repair solution using the VERA algorithm is shown in Fig. 4. With the distribution of faulty cells shown in Fig. 4, the VERA algorithm cannot repair the memory using the FAST algorithm. Therefore, after ordering the fault groups, binary trees are constructed in each ordered fault group. In this example, the optimal solutions of Group 2 are $2R_S1C_S$ and $3C_S$ (dotted lines), while the optimal solutions of Group 1 are $2R_S1C_S$ and $2C_S$ (dotted lines). Since Group 3 is a single-fault group, a repair can be performed using just one spare cell among the remaining spare cells. Therefore, after combining the solutions of the fault groups, the optimal solution is $2R_S1C_S + 2C_S + 1C_S$. VERA generates binary trees with small depths compared to those of other whole binary trees in the fault group order. The solutions of the first group are stored in the memory space, and the small binary tree is deleted from the memory space. After the flow described above is repeated for the remaining groups, an optimal solution can be found from the combination of solutions for each fault group. Therefore, VERA can generate optimal solutions for faulty memory in the shortest amount of time and reduces the amount of memory used.

IV. Simulation Results and Analysis

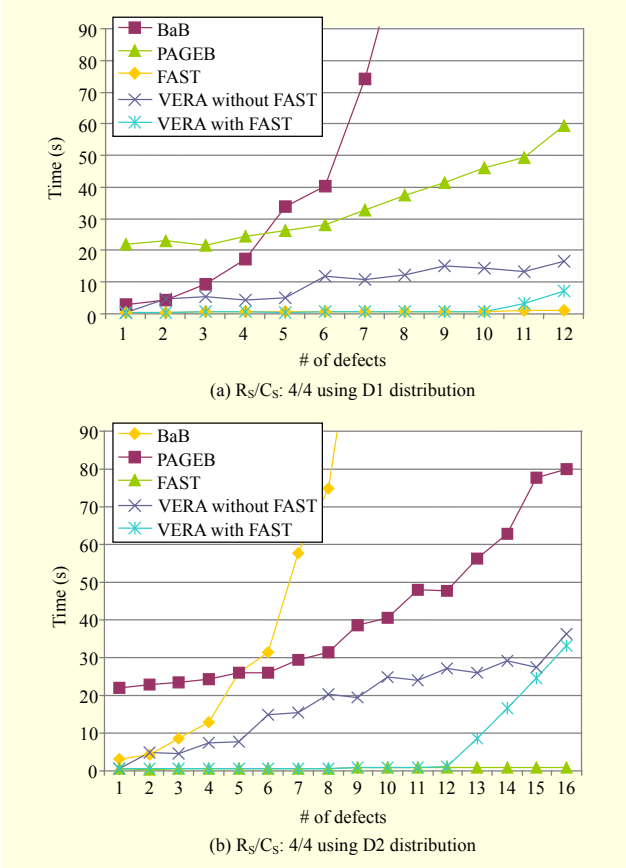
In these experiments, all algorithms are made in C languages. All program files are compiled with GNU g++ 3.4.6, and the optimization flag is used. The hardware system has 8 GB of memory and a dual UltraSPARC-III processor running at 1.2 GHz.

In this study, 1 megabit (1 block × 1,024 × 1,024) of memory is used for the experiments so as to ensure a fair comparison.

Table 2. Normalized repair rates using D2 distribution (R_S/C_S: 5/5).

(%)

# of faults	14	15	16	17	18	19	20
FAST [7]	100	96.24	85.68	73.69	63.96	65.97	61.98
VERA	100	100	100	100	100	100	100



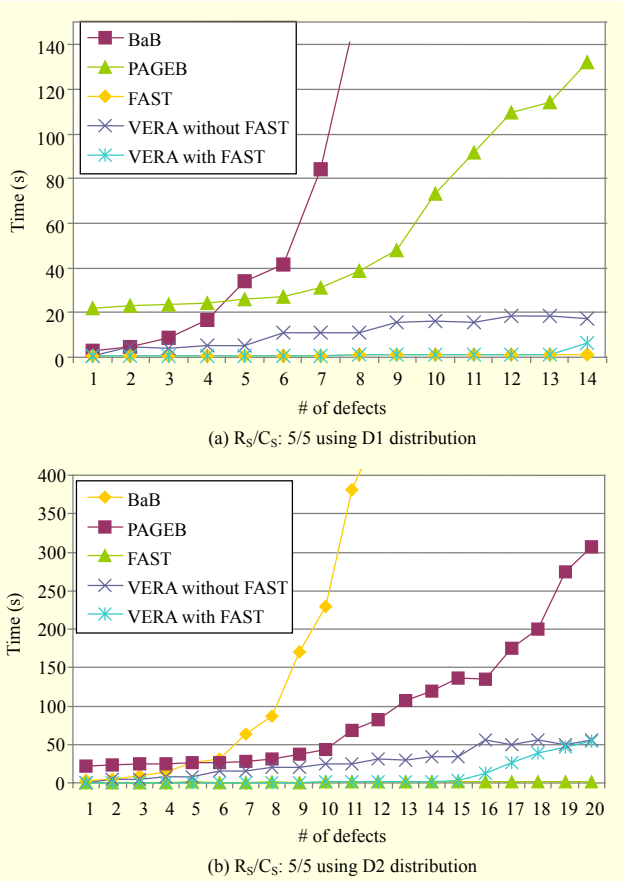
(a) R_S/C_S: 4/4 using D1 distribution

(b) R_S/C_S: 4/4 using D2 distribution

Fig. 5. Total time searching for solution (R_S/C_S: 4/4).

Each experiment is repeated 10,000 times with randomly generated addresses for the faulty cells. Experiments are performed for different redundancy configurations and numbers of random faults. The faults can generate a single faulty cell, a row or column line of faulty cells consisting of several adjacent faulty cells in a row or column, or a rectangle of faulty cells affecting 2×2 cells. The generated distributions of the fault types that are similar to that of [8] are shown in Table 1. The experiments are simulated using the "row first strategy" in which the faulty cells are repaired by row spare cells when the faulty cells can be repaired, regardless of the use of spare cells.

Faults are scattered around the entire memory area. In these experiments, the FAST algorithm and the RA algorithms having 100% repair rates are compared with the proposed



(a) R_S/C_S: 5/5 using D1 distribution

(b) R_S/C_S: 5/5 using D2 distribution

Fig. 6. Total time searching for solution (R_S/C_S: 5/5).

VERA algorithm.

The normalized repair rates according to changes in the number of faulty cells are shown in Table 2. As shown in Table 2, the VERA algorithm has a 100% normalized repair rate, and it finds an optimal solution using a minimum amount of spare cells since all combinations of solutions are analyzed.

The total time needed to search for a solution according to changes in the number of faults is shown in Figs. 5 and 6.

The data in Fig. 5(a) is generated using the D1 distribution and the data in Fig. 5(b) is generated using the D2 distribution, each with four spare rows/columns. The data in Fig. 6(a) is generated using the D1 distribution and the data in Fig. 6(b) is generated using the D2 distribution, each with five spare rows/columns. Since having many spare cells results in an excessive burden for the hardware overhead of memory, the created memory has only a small number of spare cells. Therefore, the experiments are performed within 4/4 and 5/5 spare cells. As shown in the figures, the branch-and-bound algorithm needs a significant amount of time to find a solution as the number of faults increases. Consequently, the curve for the branch-and-bound algorithm increases exponentially. It is evident that PAGEB requires a long time to calculate the DF,

Table 3. Repair rates using D1 distribution (R_S/C_S: 4/4).

(%)

# of faults	8	9	10	11	12	13
BaB	100	100	100	73.77	57.76	0
PAGEB	100	100	100	73.77	57.76	0
FAST [7]	100	100	100	70.48	53.47	0
VERA	100	100	100	73.77	57.76	0

Table 4. Repair rates using D2 distribution (R_S/C_S: 4/4).

(%)

# of faults	11	12	13	14	15	16
BaB	100	100	78.65	60.17	14.12	12.25
PAGEB	100	100	78.65	60.17	14.12	12.25
FAST [7]	99.99	99.79	68.51	47.08	11.58	8.09
VERA	100	100	78.65	60.17	14.12	12.25

Table 5. Repair rates using D1 distribution (R_S/C_S: 5/5).

(%)

# of faults	10	11	12	13	14	15
BaB	100	100	100	100	65.37	0
PAGEB	100	100	100	100	65.37	0
FAST [7]	100	100	100	100	61.85	0
VERA	100	100	100	100	65.37	0

Table 6. Repair rates using D2 distribution (R_S/C_S: 5/5).

(%)

# of faults	14	15	16	17	18	19	20
BaB	100	97.00	95.87	64.47	50.77	11.02	6.68
PAGEB	100	97.00	95.87	64.47	50.77	11.02	6.68
FAST [7]	100	93.35	82.14	47.51	32.47	7.27	4.14
VERA	100	97.00	95.87	64.47	50.77	11.02	6.68

CF, and RF even if it is much faster than the branch-and-bound algorithms [20]. Specifically, the calculation time of the CF exponentially increases as the number of spares increases. Therefore, the graph of PAGEB has a highly pitched increasing curve.

As shown in Figs. 5 and 6, two VERA algorithms are implemented, that is, one with and one without the FAST algorithm. We can see that using either of the VERA algorithms is much faster than using PAGEB. Furthermore, a VERA algorithm creates a solution in a time of nearly zero, regardless of the number of spares and faults, as shown in Fig. 5(a) and Fig. 6(a). As shown in Fig. 5(b) and Fig. 6(b), if the repair rate is too low, then the search time for finding a solution also increases. We can also see that each of the graphs (with four spare rows/columns) in Fig. 5 has a highly pitched increasing curve compared to those in Fig. 6 (with five spare rows/columns) since more time is consumed searching for the solution when the memory has more spare cells. Since the performance of the VERA algorithm is best when it is performed after the FAST algorithm is done, the entire flow of the VERA algorithm is shown in Fig. 3. However, if a user wants only optimal solutions, then the final repair of the VERA algorithm should be performed since the VERA algorithm without the FAST phase is also faster than other RA algorithms, as shown in experiment results.

Tables 3 and 4 show the repair rates according to changes in the percentages of fault distribution when the memory has four spare rows/columns. Tables 5 and 6 show the repair rates according to changes in the percentages of fault distribution when the memory has five spare rows/columns.

As shown in the tables, when the experiments are performed using the D2 distribution, VERA can repair a greater number of faulty cells compared to that using the D1 distribution. However, if the memory has many faulty cells, then the yield of the memory production line is too poor. Practically, the yield of the memory production line is greater than 90% in the industry, a situation which allows the VERA algorithm to create a solution in a time of nearly zero.

The experiment results demonstrate that the proposed VERA algorithm can generate a repair solution in a time of nearly zero while minimizing the amount of memory used. In addition, a normalized repair rate of 100% is achieved with the VERA algorithm. VERA can repair faulty memory with optimal solutions, consuming a minimal number of spare cells. Therefore, it is highly recommended that the VERA algorithm be utilized to test and repair large amounts of memory using ATE.

V. Conclusion

As the size of the memory increases, the testing time and repair time for the memory increases accordingly. Therefore, the time to repair faulty memory must be reduced. The proposed algorithm repairs faulty memory cells more quickly than other RA algorithms. Furthermore, the algorithm has a 100% normalized repair rate and is able to quickly search solutions in a nearly zero CPU time. The proposed algorithm can repair faulty memory with optimal solutions, that is, the number of spare cells wasted to repair faulty cells can be minimized. Therefore, the VERA algorithm is very useful

when simultaneously testing and repairing a large-sized memory having faults and many redundant cells. In addition, the VERA algorithm using fault grouping will be useful to repair faulty cells of 3D memories in the future since it is a very fast and exact algorithm.

References

[1] Y. Zorian, "Embedded Memory Test & Repair: Infrastructure IP for SoC Yield," *Proc. IEEE Int. Test Conf. (ITC)*, Baltimore, MD, USA, Oct. 2002, pp. 340-349.

[2] J.R. Day, "A Fault-Driven, Comprehensive Redundancy Algorithm," *IEEE Design Test*, vol. 2, no. 3, June 1985, pp. 35-44.

[3] S.-Y. Kuo and W.K. Fuchs, "Efficient Spare Allocation in Reconfigurable Arrays," *Proc. 23rd ACM/IEEE Design Autom. Conf. (DAC)*, Las Vegas, NV, USA, June 1986, pp. 385-390.

[4] C.-L. Wey and F. Lombardi, "On the Repair of Redundant RAM's," *IEEE Trans. Computer-Aided Design*, vol. 6, no. 2, Mar. 1987, pp. 222-231.

[5] R.W. Haddad, A.T. Dahbura, and A.B. Sharama, "Increased Throughput for the Testing and Repair of RAM's with Redundancy," *IEEE Trans. Computers*, vol. 40, no. 2, Feb. 1991, pp. 154-166.

[6] V.G. Hemmady and S.M. Reddy, "On the Repair of Redundant RAMs," *Proc. Design Autom. Conf.*, June 1989, pp. 710-713.

[7] H. Cho, W. Kang, and S. Kang, "A Fast Redundancy Analysis Algorithm in ATE for Repairing Faulty Memories," *ETRI J.*, vol. 34, no. 3, June 2012, pp. 478-481.

[8] P. Öhler, S. Hellebrand, and H.-J. Wunderlich, "An Integrated Built-in Test and Repair Approach for Memories with 2D Redundancy," *Proc. European Test Symp. (ETS)*, May 2007, pp. 91-96.

[9] W. Jeong et al., "An Advanced BIRA for Memories with an Optimal Repair Rate and Fast Analysis Speed by Using a Branch Analyzer," *IEEE Trans. Computer-Aided Design Integrated Circuits Syst.*, vol. 29, no. 12, Dec. 2010, pp. 2014-2026.

[10] T. Kawagoe et al., "A Built-in Self Repair Analyzer (CRESTA) for Embedded DRAMs," *Proc. IEEE Int. Test Conf. (ITC)*, Oct. 2000, pp. 567-574.

[11] H. Cho, W. Kang, and S. Kang, "A Built-In Redundancy Analysis with a Minimized Binary Search Tree," *ETRI J.*, vol. 32, no. 4, Aug. 2010, pp. 638-641.

[12] M. Tarr, D. Boudreau, and R. Murphy, "Defect Analysis System Speeds Test and Repair of Redundant Memories," *Electronics*, Jan. 1984, pp. 175-179.

[13] C.-T. Huang et al., "Built-In Redundancy Analysis for Memory Yield Improvement," *IEEE Trans. Reliability*, vol. 52, no. 4, Dec. 2003, pp. 386-399.

[14] W. Jeong et al., "A Fast Built-in Redundancy Analysis for Memories With Optimal Repair Rate Using a Line-Based Search Tree," *IEEE Trans. Very Large Scale Integration (VLSI) Syst.*, vol. 17, no. 12, Dec. 2009, pp. 1665-1678.

[15] T.-J. Chen, J.-F. Li, and T.-W. Tseng, "Cost-Efficient Built-In Redundancy Analysis With Optimal Repair Rate for RAMs," *IEEE Trans. Computer-Aided Design Integrated Circuits Syst.*, vol. 31, no. 6, June 2012, pp. 930-940.

[16] M.B. Healy and et al., "Design and Analysis of 3D-MAPS: A Many-Core 3D Processor with Stacked Memory," *Proc. IEEE Custom Integrated Circuits Conf. (CICC)*, San Jose, CA, USA, Sept. 2010, pp. 1-4.

[17] H. Saito et al., "A Chip-Stacked Memory for On-Chip SRAM-Rich SoCs and Processors," *IEEE J. Solid-State Circuits*, vol. 45, no. 1, Jan. 2010, pp. 15-22.

[18] C.-W. Wu, S.-K. Lu, and J.-F. Li, "On Test and Repair of 3D Random Access Memory," *Proc. Design Automation Conf. (ASP-DAC)*, Jan. 2012, pp. 744-749.

[19] J. Lee, K. Park, and S. Kang, "Yield Enhancement Techniques for 3D Memories by Redundancy Sharing among All Layers," *ETRI J.*, vol. 34, no. 3, June 2012, pp. 388-398.

[20] H.-Y. Lin, F.-M. Yeh, and S.Y. Kuo, "An Efficient Algorithm for Spare Allocation Problems," *IEEE Trans. Reliability*, vol. 55, no. 2, June 2006, pp. 369-378.

Hyungjun Cho received his BS and MS in electrical and electronics engineering from Yonsei University, Seoul, Rep. of Korea in 2005 and 2007, respectively. He was also a research engineer with the ASIC Research Center, Yonsei University. He is currently working toward his PhD in electrical and electronics engineering at Yonsei University. His main research interests include very large scale integration (VLSI) design, fault simulation, fault diagnosis, built-in self-repair (BISR), built-in self-test (BIST), built-in redundancy analysis (BIRA), RA algorithm, and reliability.

Wooheon Kang received his BS in electrical and electronics engineering from Yonsei University, Seoul, Rep. of Korea, in 2009. He is currently working toward a combined PhD from the Department of Electrical and Electronics Engineering, Yonsei University. His current research interests include built-in self-repair, built-in self-testing, built-in self redundancy analysis, redundancy analysis algorithms, reliability, and very large scale integration design.

Sungho Kang received his BS from Seoul National University, Seoul, Rep. of Korea, and his MS and PhD in Electrical and Computer Engineering from the University of Texas at Austin, Austin, TX, USA, in 1992. He was a research scientist with the Schlumberger Laboratory for Computer Science, Schlumberger Inc., and a senior staff engineer with the Semiconductor Systems Design Technology, Motorola Inc. Since 1994, he has been a professor with the Department of Electrical and Electronics Engineering, Yonsei University, Seoul, Rep. of Korea. His main research interests include VLSI design and testing, design for testability, BIST, defect diagnosis, and design for manufacturability.