A BIRA for Memories With an Optimal Repair Rate Using Spare Memories for Area Reduction

Wooheon Kang, Hyungjun Cho, Joohwan Lee, and Sungho Kang, Member, IEEE

Abstract—The test cost and yield improvement of embedded memories have become very important as memory capacity and density have grown. For embedded memories, built-in redundancy analysis (BIRA) is widely used to improve yield by replacing faulty cells with a 2-D redundancy architecture. However, the most important factor in BIRA is the reduction of hardware overhead while keeping optimal repair rate. Most BIRA approaches require extra hardware overhead in order to store and analyze faults in the memory. These approaches do not utilize spare memories during the redundancy analysis (RA) procedure. However, the proposed BIRA minimizes area overhead by utilizing a part of the spare memory as an address mapping table (AMT). Since storing the faulty memory addresses take most of the extra hardware overhead, the reduced logical addresses produced by the AMT are used to reduce the extra hardware overhead. In addition, the reduced addresses are stored in content-addressable memories (CAMs) and used in the RA procedure. The proposed BIRA can achieve an optimal repair rate by using an exhaustive search RA algorithm. The proposed RA algorithm compares the repair solution candidates with all the fault addresses stored in the proposed CAMs to guarantee an exhaustive search. The experimental results show that the proposed BIRA requires a smaller area overhead than that of the previous state-of-the-art BIRA with an optimal repair rate.

Index Terms—Address mapping table (AMT), area overhead repair rate, built-in redundancy analysis (BIRA).

I. INTRODUCTION

As the capacity and density of semiconductor memories have rapidly increased with the technological development of semiconductor manufacturing, the probability of memory faults has increased. This causes yield drops and quality degradation. To achieve reasonable yield and quality, the faulty cell repair adds spare rows and columns to the memory to create 2-D redundant cells. The faulty cells are repaired line by line, including faulty cells which are repaired with a spare row or column, instead of using a bit-by-bit approach. This is called a redundancy analysis (RA), which analyzes the faults detected in the memory. RA is complicated because of the restricted spare rows/columns and line-based approach for finding a solution. Therefore, the research into RA has been active to increase the yield of memories.

Most RA algorithms have been based on 2-D redundant cells since the 1980s. 2-D redundant cells become a source for a binary search tree to achieve an optimal repair rate because there are two choices to repair faulty cells. Furthermore, various RA algorithms for 2-D redundancy have been developed [1]–[22]. Repair-most (RM) [3], CRESTA [4], LRM [5], and ESP [5] are the most well-known RA algorithms for built-in redundancy analysis (BIRA). RM is a simple algorithm. It counts the number of faults in each line and allocates the spare lines to a faulty line in descending order of the number of faults. Although its repair rate is high, it is not optimal, because an exhaustive search is not performed. CRESTA focuses on the optimal repair rate and fast analysis. To achieve those goals, it simultaneously analyzes the entire cases of possible solutions with several parallel subanalyzers. Therefore, the number of subanalyzers is proportionate to the number of spare cells because the entire cases of possible solutions are combinations of the available spare cells. LRM and ESP mainly focus on minimizing the area overhead in terms of storage requirements with a simple RA algorithm. To reduce the area overhead, these two algorithms are designed to reduce the area of the failure bitmap to store faulty information. However, the repair rates of the two algorithms are not optimal since faulty information is omitted.

Recently, many tree-based RA algorithms have also been introduced that make an effort to lessen the search space. These algorithms are based on the branch and bound (B&B) algorithm [1], [2], [11]–[14]. The B&B algorithm is a simple fault-driven approach. IntelligentSolve (IS) and Intelligent-SolveFirst (ISF) algorithms achieve both low area overhead and an optimal repair rate [11]. However, the two algorithms take a lot of time to complete the RA in cases with complex fault distributions. In addition, the area overhead they use is not small enough for commercial production purposes, since it is still large.

Conventional algorithms based on B&B try to reduce the search space of the tree for higher RA speed with low area overhead. However, these algorithms reach a limit in terms of the performance improvement because they store whole fault address and analyze each fault. To overcome the restriction of these approaches, SFCC [12] and BRANCH analyzer [15]–[17] were introduced. SFCC achieves a low area overhead and optimal repair rate, and reduces the search space by building a search tree based on the line faults. The BRANCH analyzer analyzes all nodes concurrently within a branch for combinations of 2-D spares. In order to analyze
all nodes concurrently and achieve an optimal repair rate. BRANCH analyzer and SFCC store all fault information into fault-storing content-addressable memories (CAMs). In [16], an optimal repair rate is achieved using a linear feedback shift register which generates repair solution candidates. In [17], a low area overhead is achieved using local bitmap and optimal repair rates by attempting all possible repair solutions. However, there is still an improvement in terms of area overhead, i.e., reduced area overhead, because the full address of the memory is stored.

In [19], built-in self-repair (BISR) with a flexible spare architecture which can configure the same spare to a row, a column, or a rectangle to fit the failure pattern is proposed. However, the method in [19] is too complicated to use a flexible spare architecture. In [20], BIRA using spare memories in order to reduce the area overhead is introduced. However, the method in [20] does not achieve an optimal repair rate since the RA algorithm is based on the RM algorithm.

There is one other important point: i.e., whether the memory under test is repairable or not before the RA procedure. Some faulty memories are not repairable even when all available redundant cells are used during the RA procedure. It is a waste of time to carry out RA for irreparable memories because the result of RA is reported after the RA procedure. If the irreparability status is known before the RA procedure, it saves time which might otherwise be spent on unnecessary actions. Because of this, there are theories of early termination using a repair set of leading elements [21]. Many cases of irreparable memories are detected by six theories using the repair set.

We propose a BIRA method that focuses on improving the area overhead and optimal repair rate. To minimize the additional area overhead, the proposed BIRA uses spare memory to store an address mapping table (AMT) and to minimize the fault-storing CAMs. The new exhaustive search scheme guarantees the optimal repair rate.

The rest of the paper is organized as follows. Background information concerning BIRA and fault classification is described in Section II. The proposed BIRA architecture is described in Section III. An AMT of the proposed BIRA is described in Section IV. The proposed BIRA algorithm with a simple example to help understand the procedure is described in detail in Section V. The experimental results of the area overhead and repair rate are shown and compared with previous studies in Section VI. Finally, the conclusions are given in Section VII.

II. BACKGROUND

Memory is tested by using a built-in self-test (BIST) module or external automatic test equipment (ATE) before RA is performed. According to the testing method, there are two methods to repair faulty memories using a BIRA module and an external ATE. Additional hardware area is needed because BIRA module is implemented using hardware, while RA using ATE does not require any additional hardware. Therefore, if the memory size increases, then the hardware overhead for BIST and BIRA modules increases. This leads to a yield drop per wafer because the trend in manufacturing is mass production and additional modules occupy more memory area.

A. Performance Criteria of BIRA

The three main performance criteria of BIRA are the analysis speed, repair rate, and area overhead. The optimal repair rate, which is the ratio of memories repaired by BIRA to all repairable memories of the total tested memories, is essential because the yield of memories repaired through BIRA is important. However, because the RA algorithm complexity for memory repair using 2-D spare cells is NP-complete [1], an exhaustive search is a unique method to achieve an optimal repair rate. In the exhaustive search, a binary search tree is applied, since a faulty cell is repaired by 2-D redundant cells. All existing faults in the memory are examined in order to find the correct repair solution. Therefore, when the number of faults increases, the analysis time is longer. Although the depth of the tree depends on the number of spare cells, the analysis time depends on the number of faults. Since the area for the memory, spare memory, BIST module, and BIRA module is limited, it has an effect on the yield of the memories. Because there is a tradeoff between these features, however, it is difficult to improve three features at the same time. While the analysis speed is negligible because BIST time takes almost as long as test and repair time, the area overhead and repair rate make a big impact on the yield of memories.

B. Fault Classification

The failure pattern is defined as the distribution of faults caused by a single defect [23]. The failure pattern is classified into six types: word line fault, bit line fault, cluster fault, continuous fault, twin fault, and single fault. Word (bit) line fault is column (row) failures. Cluster fault is formed of various shapes by distribution of faulty cells. Continuous fault is a line fault which has greater than two faults, i.e., faulty cells are continuously distributed. Twin fault is formed by a line fault or a diagonal fault by two faults. A single fault does not share a row and column address with other faults.

Since a memory with 2-D spare architecture is repaired by line replacement, faults (failure pattern) were classified into three types: single fault, sparse line fault, and must-repair line fault [12], [15]. The number of spare rows and columns are $R_s$ and $C_s$, respectively. Single fault does not share a row and column address with other faults. Diagonal fault of twin fault and single fault can be defined as a single fault. Sparse line fault is a faulty row or column line. A faulty row (column) line has more than one and less than or equal to $C_s$ ($R_s$) faults. Cluster fault, continuous fault, and line fault of twin fault can be defined as a sparse line fault. A must-repair faulty row (column) line has greater than $C_s$ ($R_s$) faults. Word line fault, bit line fault, part of cluster fault, and continuous fault can be defined as must-repair line fault.

Because the cluster faults contain many cross line faults, the cross line faults generate complicated cases for RA. For example, there are three faults, cell $(0,0)$, cell $(0,1)$, and $(1,0)$. To repair these faults, there are three repair solutions: two spare rows ($R_0$, $R_1$), one spare row and one column ($R_0$, $C_0$),
Thus, repair solutions are affected by cluster fault (i.e., fault column). According to the number of available spare memories, distribution) and the number of available spare memories. Fig. 1. Block diagram of the proposed BIRA architecture.

KANG et al. developed BIRA to recover storing CAMs. BIRA then finds the repair solution by reducing the length of the information that is stored in the CAMs, and recovers the mapped addresses. In this paper, a method is proposed to reduce the information stored in the CAMs. These mapped addresses are sent to BIRA through the port Mapped_Address. If the address is the same, its mapped addresses are sent to the BIRA_Controller through the port Original_Address. If the address is different, it is stored in the AMT and mapped to the addresses with already stored addresses in the AMT. After the test for the spare memory is done, the RAM is tested. Because the RAM cannot be repaired by a faulty spare memory and we do not use the faulty spare memory as an AMT, the address of faulty spare memory should be stored in the Spare_ID_register. When BIST detects a fault from the spare memory, the signal S_Fail is asserted, and the faulty addresses are sent to the Spare_ID_register through the port S_Address. The BIRA_Controller counts the available spare memories and checks faulty spare memory to perform an adequate RA procedure.

After the test for the spare memory is done, the RAM is tested. Because the RAM cannot be repaired by a faulty spare memory and we do not use the faulty spare memory as an AMT, the address of faulty spare memory should be stored in the Spare_ID_register. When BIST detects a fault from the spare memory, the signal S_Fail is asserted, and the faulty addresses are sent to the Spare_ID_register through the port S_Address. The BIRA_Controller counts the available spare memories and checks faulty spare memory to perform an adequate RA procedure.

III. PROPOSED BIRA ARCHITECTURE

Fig. 1 shows a conceptual block diagram of a built-in self-repair (BISR), which is composed of BIST and BIRA. Because the spare memory is used as an AMT, it is tested by BIST before the RAM is tested. In the test mode, BIST detects the faults in the RAM and sends the fault information to BIRA. BIRA can then change the fault addresses into short-mapped logical addresses. In this paper, a method is proposed to map long physical addresses into short logical addresses to reduce the length of the information that is stored in the faulty storing CAMs. BIRA then finds the repair solution by analyzing these mapped addresses. Finally, BIRA recovers the original addresses after finding the repair solution according to the AMT information. In the normal mode, BIRA remaps the addresses of faulty lines in the RAM to the addresses of available spare memories with a repair solution constructed by the RA algorithm. Although the information of the AMT is stored in the spare memories and it is used during RA procedure, the only availability of the spare memories is important for spare allocation. After the repair solution is decided by RA algorithm, the information of the AMT is not used.

The proposed BIRA consists of BIRA_Controller, fault storing CAMs, Spare_ID_register, and Repair_register. The controller performs several procedures: it compares faulty addresses, stores the original addresses in the AMT, stores the mapped addresses in the CAMs, and recovers the mapped addresses to the original addresses. The mapped addresses are stored in the fault storing CAMs, which are based on the CAMs introduced in [24]. The fault collection procedure of the proposed BIRA will be described in more detail in Section V-A. The faulty addresses of the faulty spare memories are stored in the Spare_ID_register. The faulty spare memory is not suitable to use for the AMT as also to replace faulty memory lines. That is, BIRA schemes perform RA procedures taking into account the amount of remaining good spare memory. The repair registers store the repair solutions, which are the memory addresses from the BIRA controller. Once the RA procedure is completed, the spare memories are allocated according to the information in the Repair_register.

As Fig. 1 shows, the spare memory is tested before the RAM is tested. Because the RAM cannot be repaired by a faulty spare memory and we do not use the faulty spare memory as an AMT, the address of faulty spare memory should be stored in the Spare_ID_register. When BIST detects a fault from the spare memory, the signal S_Fail is asserted, and the faulty addresses are sent to the Spare_ID_register through the port S_Address. The BIRA_Controller counts the available spare memories and checks faulty spare memory to perform an adequate RA procedure.

After the test for the spare memory is done, the RAM is tested. Because the RAM cannot be repaired by a faulty spare memory and we do not use the faulty spare memory as an AMT, the address of faulty spare memory should be stored in the Spare_ID_register. When BIST detects a fault from the spare memory, the signal S_Fail is asserted, and the faulty addresses are sent to the Spare_ID_register through the port S_Address. The BIRA_Controller counts the available spare memories and checks faulty spare memory to perform an adequate RA procedure.

After the test for the spare memory is done, the RAM is tested. Because the RAM cannot be repaired by a faulty spare memory and we do not use the faulty spare memory as an AMT, the address of faulty spare memory should be stored in the Spare_ID_register. When BIST detects a fault from the spare memory, the signal S_Fail is asserted, and the faulty addresses are sent to the Spare_ID_register through the port S_Address. The BIRA_Controller counts the available spare memories and checks faulty spare memory to perform an adequate RA procedure.

IV. ADDRESS MAPPING TABLE OF THE PROPOSED BIRA

The proposed BIRA focuses on low area overhead and an optimal repair rate even though the time for repairing the faulty memory is increased. To reduce the area overhead for BIRA, the fault information is stored in the spare memory, and the spare memory is used as an AMT.

Fig. 2 shows the AMT structure of the proposed BIRA for 2-D spare memory architecture. The notations are defined as follows: \( R_s \) is the number of spare rows, \( C_s \) is the number of spare columns, \( M \) is the number of rows of a memory, and \( N \) is the number of columns of a memory. The AMT consists of four fields: address descriptor field, row/column address field, mapped row/column address field, and row/column repeat fault counter field. The address descriptor field is used for
classification since the fault addresses, i.e., the row address and column address, are stored in the spare row memories at the same time. The original row or column address of the memory under test is stored in the row/column address field. The range of the original row address is from 0 to \([M-1]\), and the range of the original column address is from 0 to \([N-1]\). The mapped addresses are allocated according to the fault detecting order and are stored the mapped row/column address field. The range of the mapped row address is from 0 to \([Rs(Cs+1)]-1\), and the range of the mapped column address is from 0 to \([Cs(Rs+1)]-1\). The row/column repeat fault counter field shows the number of faults in the line. If the number of faults in a row (column) is greater than \(Cs\) (\(Rs\)), then the faulty row (column) line becomes the must-repair row (column) \([3]\). This is called must-repair analysis because this faulty row (column) line must be repaired with a spare row (column) line in order to have a solution. There remains at least one fault when this faulty row (column) line is repaired with several spare columns (row) since the number of faults in a row (column) is greater than \(Cs\) (\(Rs\)). Therefore, the repeat fault count for the row and column is \(Cs\) and \(Rs\), respectively. The maximum number of different row addresses and column addresses with a must-repair condition is \(Rs(Cs+1)\) and \(Cs(Rs+1)\), respectively \([5]\). It is unnecessary to store faults that have the same address as the must-repair address after the must-repair lines are chosen. If the row address of a newly detected fault is matched to the must-repair row, the column address is not stored in the column address of the AMT, and vice versa.

There are cases in which faults occur on the spare memories, because the spare memories are memory as well. Therefore, we need to ensure that there are enough spare row memories to store the information, since the number of different addresses is dependent on the number of spare memories. If all the spare rows (spare columns) are faulty, the spare allocation is executed whenever a fault is detected by BIST. The worst case is when there is only one spare row and several spare columns, because the spare row memory is used as an AMT. The AMT information is stored in only one spare row in this case. Although there is only one spare row, all the AMT information can be stored in the one spare row because the size of the memory is large. For example, if there is 1 spare row and 17 spare columns when \(M = 1024\) and \(N = 1024\), the spare row is large enough to store the AMTs because the total number of bits is 990 [the number of bits of row AMTs: \(18 \times (1 + 10 + 5 + 5)\) and the number of bits of column AMTs: \(34 \times (1 + 10 + 6 + 1)\)], as illustrated by the structure shown in Fig. 2.

When the fault is detected by BIST, in order to convert the newly faulty address to the mapped address, the new faulty address is compared to stored addresses in the AMT in sequence, and BIST is halted. If the matching occurs, the new faulty address is converted to the mapped address and the row/column repeat fault count in the AMT increases by 1. If there is no matched address, the new faulty address is stored in the AMT and converted to the mapped address. After the original fault address is converted to the mapped address, BIST operation is resumed. The time overhead for comparing the address is analyzed because the comparison procedure is sequentially performed. BIST time occupies most of the test and repair time. The size of spare memory is less than the size of main memory. If there is \(1024 \times 1024\) memory with 10 spare rows, the area overhead of spare row memory is only 1%. Therefore, the increase in time for using spare memory is negligible.

V. PROPOSED BIRA APPROACH

A. Fault Collection

The proposed BIRA uses the spare memory as the AMT. Fig. 3 is an example of a faulty memory that is \(M \times N\) using 2-D spare architecture. The spare row memories are adequate to use the AMT because the spare row memories are more accessible than the spare column memories.

Fig. 4 shows the fault-storing CAM structure of the proposed BIRA for memory using 2-D spare architecture with \(Rs\) spare rows and \(Cs\) spare columns. The length of the fault address of the CAMs does not depend on the size of the memory; instead, it depends on the number of spare memories, since the proposed BIRA requires only short mapped addresses, which are decided by the number of spare memories. Although both the widths of parent CAMs and child CAMs are the same, the parent CAMs and the child CAMs are separated in order to facilitate the understanding of the structure of CAMs, fault collection, and
RA procedure. The maximum number of parent address CAMs is the same as the sum of the total spares (i.e., $Rs + Cs$). The maximum number of child address CAMs is $Rs(Cs - 1) + Cs(Rs - 1)$ (i.e., $2RsCs - Rs - Cs$); the size of the CAMs to restore the child faults has already been introduced in [5], [11], [12], and [15].

When the faults are detected by BIST, the faulty information is sent to BIRA, and BIST is stopped. The proposed BIRA compares the newly detected fault address and all stored fault addresses in the AMT to change memory address to the mapped address and classify into must-repair faulty line, faulty line, and single fault. The memory address is mapped to a short logical address that depends on the amount of spare memory. The mapped address is stored into the parent address CAMs and child address CAMs according to their characteristics; parent faults, like the leading elements introduced in [12] and [15], have different row addresses and column addresses from each other. If the row address or column address is the same as that of any of the other parent faults, then the fault is classified as a child fault.

Fig. 5 shows an example of how the proposed BIRA collects faulty addresses for the same memory that was shown in Fig. 3. The example memory has two spare row lines and two spare column lines with 10 faults. There is a total of four parent address CAMs and child address CAMs in this case. Since the size of the memory is $1024 \times 1024$, the length of the row address and column address is 10 bits each. However, the length of the mapped row address and mapped column address is 3 bits each in the proposed BIRA.

The first fault, cell (1, 4), is detected by BIST and the faulty information is sent to BIRA. The fault address is compared with all of the address in the AMT; however, since this is the first fault, the spare memories are empty, so the row address is mapped to 0 and the column address is also mapped to 0. The mapped address is then stored in the parent address CAMs shown in Fig. 5(a). The second fault, cell (2, 1), does not match any addresses when compared to the AMT, so its row address and column address are mapped to a new address; the row address is mapped to 1 and the column address also is mapped to 1. The mapped address is then stored in the parent address CAMs shown in Fig. 5(b). Since the third fault, cell (2, 4), has the same column address as the first fault and the same row address as the second fault, the row repeat fault count of the second row AMT and the column repeat fault count of the first column AMT are updated to 2, respectively. Then the mapped address is stored into the child address CAMs shown in Fig. 5(c). The fourth fault, cell (4, 0), is similar to the second fault, so, its row address is mapped to 2, its column address is mapped to 2, and the mapped address is stored in the parent address CAMs shown in Fig. 5(d). When the fifth fault, cell (4, 3), is compared to the AMT, the row address of the fifth fault is the same as the row address of the fourth fault. The row repeat fault count of the third row AMT is updated to 2, and the column address of the fifth fault is mapped to 3. Then the mapped address is stored into the...
child address CAMs shown in Fig. 5(e). The row repeat fault count of the third row AMT is updated to 3 because of the row address of the sixth fault, cell (4, 5). Since the number of faults in row 4 is greater than \( Cs \), row 4 (i.e., the mapped row address is 2) becomes the must-repair row. Therefore, the enable flag of the third parent address CAMs and the enable flag of the second child address CAMs become 0, as shown in Fig. 5(f). Because the row address of the seventh fault, cell (4, 7), is the same as the must-repair row 4, the AMT and the fault storing CAMs are not changed, as shown in Fig. 5(g). When the eighth fault, cell (5, 1), is compared with the AMT, its column address is the same as the column address of the second fault. The column repeat fault count of the second column AMT is updated to 2, and the row address of the fifth fault is mapped to 3. Then the mapped address is stored into the child address CAMs shown in Fig. 5(h). The ninth fault, cell (6, 5), is similar to the fourth fault, so, its row address is mapped to 4 and its column address is mapped to 3, and the mapped address is stored in the parent address CAMs shown in Fig. 5(i). The 10th fault is the last one, and is stored in the child address CAMs because of its row address, as shown in Fig. 5(j).

B. Redundancy Analysis

As mentioned in Section I, the RA algorithm complexity for memory repair using 2-D spare cells is NP-complete, and an exhaustive search is a unique method to achieve an optimal repair rate. The RA procedure of a fault-driven exhaustive searching algorithm, such as ISF or SFCC, is based on a binary search tree. Because each branch of a binary search tree can be a repair solution, and these RA algorithms compare a faulty address with every node in a branch, a considerable amount of time is required to find a repair solution. BRANCH analyzer analyzes all nodes in a branch concurrently for combinations of 2-D spares in order to reduce the time needed for finding a repair solution. The number of branches of a binary search tree for a memory with \( Rs \) spare rows and \( Cs \) spare columns is \((Rs + Cs)!/R!^*Cs!)\). For example, if there are two spare rows and two spare columns, the number of solution candidates is 6, and each branch is a set of solution candidates. Each branch has a different order for the row and column address combinations, such as RRCC, RCRC, RCCR, CRRC, CRCR, and CCRR, where the \( R \) depicts the spare row and the \( C \) depicts the spare column.

After the fault collection is completed, the proposed BIRA starts RA in order to find repair solutions. In order to achieve an optimal repair rate, an RA algorithm, which is an exhaustive search method, is proposed. All the solution candidates are applied to the parent faults and child faults according to the number of parent faults. To perform an exhaustive search, if the number of parent faults is smaller than the sum of available spare memories, some child faults should be applied to the solution candidates. If there is one solution at least, then the memory is repairable. If there is no solution after all the solution candidates are applied to the parent faults, then the memory is irreparable. If the number of parent faults is less than the sum of the available spare memories, there may be no solution by applying solution candidates to the row and column combinations of the parent faults. Therefore, a new RA algorithm, which considers the number of parent faults, is proposed.

There are two cases after fault collection. In the first, the number of parent faults is equal to the number of available spare memories, and, in the second, the number of parent faults is less than the number of available spare memories. According to the number of parent faults and the number of available spare memories, there is a different point of finding a repair solution. For the former case, the number of solution candidates from the available spare memories is equal to the number of combinations of the parent faults choosing the available spare rows. It is clear that there are solutions in the solution candidates because all of the parent faults are covered by some solution candidates, and all of the slave faults are covered because of their address dependency as well. If there is no solution, however, the memory is not repairable, since the faults remain after the RA procedure. For the latter case, the number of solution candidates from the available spare memories is greater than the number of combinations of the parent faults. It is possible that there is no solution among the solution candidates even though the memory is repairable, because of the unused spare memories. In this case, the solution candidates are expanded to the slave faults, and so the complexity is increased.

A detailed RA procedure is described, as shown in Fig. 6. The basic strategy of the proposed RA algorithm is to compare

---

| RA, Ca | available row spares and column spares |
| Pa_cnt | number of parent faults |
| Ch_cnt | number of child faults |
| Sol_cnt | set of repair solution candidates |
| N | number of selected repair solution candidates from child address CAMs |
| PCAM | parent address CAMs |
| CCAM | child address CAMs |

**Fig. 6. Proposed RA algorithm.**
all fault addresses to set a of repair solution candidates. Although the RA procedure is simple, an exhaustive search is possible since the set of repair solution candidates contains all repair solution candidates from the available spare memories. Also, since the complexity of finding a repair solution depends on the relation between the number of parent faults and the number of available spare memories, an appropriate set of repair solution candidates is required for each case after the fault collection.

For the case in which the number of parent faults is equal to the number of available spare memories, the number of available spare rows of the parent faults is selected as the row repair solution. Then, the remaining parent faults, after the spare rows are selected, are the column repair solution. These repair solution candidates are stored into the spare register, which is composed of a spare row register and a spare column register. The repair solution candidates that are generated by these procedures are equivalent to the branches of a binary search tree since the number of combinations of available spare rows of the parent faults is equal to the number of branches of available spare rows and available spare columns, \((Ra + Ca)!/Ra!Ca!\). According to the order of the repair solution candidates, the faults in the parent address CAMs and the faults in the child address CAMs are compared with the repair solution candidates in the spare register. If the address of the spare row register is matched to the row address parent address CAMs or to the row address child address CAMs, the enable flag of the parent address CAM or the enable flag of the child address CAM becomes zero. Similarly, if the address of the spare column register is matched to the column address in the parent address CAMs or to the column address in the child address CAMs, then the enable flag of the parent address CAM or the enable flag of the child address CAM becomes zero. After the comparison process is finished, all enabled flags of the parent address CAMs are zero. However, some enable flags of the child address CAMs are still equal to 1. In this case, the next repair solution candidate is picked, and the proposed BIRA repeats the comparison process until all the enable flags of the child address CAMs are zero.

For the case in which the number of parent faults is less than the number of available spare memories, it is possible that there is no solution in the combination of available spare rows of the parent faults. Because the number of combinations of available spare rows of the parent faults is less than the number of branches from spare memories, an exhaustive search is not guaranteed. In this case, the parent faults are selected, and the child faults are selected for the repair solution candidates. The number of selected row addresses from the child faults is increased from 0 to the number of parent faults minus \(Ca\), while the number of selected row addresses from the parent faults is decreased from \(Ra\) to the number of parent faults minus \(Ca\). Because the repair solution candidates are based on the addresses of the parent address, the number of selected row addresses is decreased to the number of parent faults minus \(Ca\). These repair solution candidates are compared with the fault-storing CAMs in order to find the repair solution like the former case in which the number of parent faults is equal to the number of available spare memories.

Since the addresses of the repair solutions through the RA procedure are mapped addresses, the final step is to find a final repair solution. According to the mapping rule of the AMT, the mapped addresses return to the original addresses.

**C. Examples of Proposed BIRA**

An RA example for the case in which the number of parent faults is equal to the number of available spare memories using the proposed BIRA is shown in Fig. 7. The RA process is performed for a faulty memory with two spare rows and two spare columns, as shown in Fig. 1. Fig. 7(a) shows a status of the parent address CAMs and child address CAMs after the fault collection and must-repair process. There are three available spare memories: one spare row and two spare columns, and there are three parent faults after the must-repair process. The set of selected row addresses is \((0, 1, 4)\), and the set of repair solution candidates is \{\((R0, C1, C3), (R1, C0, C3), (R4, C0, C1)\}\). Fig. 7(b)–(d) shows the status of the fault-storing CAMs after the proposed RA is executed by each repair solution candidate. The repair solution candidates in Fig. 7(b) and (c) are not repair solutions because the enable flag of the child address CAMs is 1. On the other hand, the repair solution candidate in Fig. 7(d) is a repair solution because all the enable flags of the parent address CAMs and child address CAMs are 0. The repair solution candidate \((R4, C0, C1)\) is selected as the repair solution. Although the repair solution is selected, the repair solution is not an exact solution. Since the addresses are mapped logical addresses, the final step remains to repair the faulty memory. Fig. 7(e) shows the status of the AMT after the fault collection and must-repair processes. According to the AMT, the repair solution including the must-repair line \((R2, R4, C0, C1)\) is returned to \((R4, R6, C4, C1)\), as shown in Fig. 7(f).

Another RA example for the case in which the number of parent faults is less than the number of available spare memories using the proposed BIRA is shown in Fig. 8.
Fig. 8(a) shows a memory that has seven faults with two spare rows and two spare columns. Fig. 8(b) shows the status of the AMT after the fault collection. Fig. 8(c) shows stored faulty information for the memory of Fig. 8(a) within the parent address CAMs and the child address CAMs. According to the fault detection order shown in Fig. 8(a), the faults are classified as three parent faults and four child faults. The repair solution candidates contain the row address from the child address CAMs since the number of parent faults is less than the available spare memories. There are two cases of the repair solution candidate sets: in the first case, they are two row addresses from the parent address CAMs; in the second case, one is a row address from the parent address CAMs and the other is a row address from the child address CAMs.

If two row addresses are selected from the child address CAMs, the faulty memory is not repaired because the number of parent faults is greater than the number of spare rows. The set of repair solution candidates that is selected from only the parent address CAMs is {(R0, R1, C3), (R0, R3, C1), (R1, R3, C0)}, since the set of selected row addresses is {(0, 1), (0, 3), (1, 3)}. Fig. 8(d) contains the status of the fault-storing CAMs after the proposed RA is executed by the set of repair solution candidates. The repair solution is (R0, R2, C1, C3) since all of the enable flags of the fault storing CAMs are 0. Therefore, according to the AMT, the repair solution (R0, R2, C1, C3) is returned to (R2, R5, C0, C6).

Although the fifth repair solution candidate is the selected repair solution in this example, the total number of repair solution candidates is 15. When the selected row addresses are two addresses from the parent address CAMs, there are three repair solution candidates. When the number of selected row addresses from the parent address CAMs and child address CAMs is 1, the number of repair solution candidates is 12. When all of the parent address CAMs contain valid addresses, the maximum number of branches is \(\frac{(R_s + C_s)!}{R_s! × C_s!}\) for both BRANCH analyzer and the proposed BIRA method.

When some of the statuses of the parent address CAMs are invalid, however, the maximum number of repair solution candidates is increased. For BRANCH analyzer, the maximum number of repair solution candidates is

\[
\frac{(R_s + C_s)!}{R_s! × C_s!} \times \left(1 + \left(\frac{2R_sC_s - R_s - C_s}{\min(R_s, C_s)}\right)\right).
\]

(1)

According to the theorem introduced in [21], the number of parent faults that require two spare memories to repair is \(\min(R_s, C_s)\). The total number of parent faults is \(\max(R_s, C_s)\) in this case because the repairable memory satisfies the
condition that the sum of the parent faults and parent faults that require two spare memories is less than or equal to the sum of the spare memories [i.e., required spare memories are $R_s + C_s$ since $\min(R_s, C_s) + \max(R_s, C_s)$]. The number of invalid parent address CAMs is $\min(R_s, C_s)$. The last term of (1) leads to the maximum number of repair solution candidates. Term 1 of (1) is added because BRANCH analyzer is executed by the first set of repair solution candidates from only the parent address CAMs. If there are two spare rows and two spare columns, for example, the maximum number of repair solution candidates is 42.

For the proposed BIRA, the maximum number of repair solution candidates is

$$
\sum_{i=0}^{R_s} \left( \frac{\max(R_s, C_s)}{R_s - i} \right) \times \left( \frac{2R_sC_s - R_s - C_s}{i} \right). \quad (2)
$$

The number of invalid parent address CAMs is $\min(R_s, C_s)$ in the same manner. Because the proposed BIRA focuses on parent faults and child faults, however, the maximum number of repair solution candidates is dependent on the selected row addresses from the child address CAMs as shown in (2).

If there are two spare rows and two spare columns, for example, the maximum number of repair solution candidates is 15.

Since the proposed BIRA requires a step in which BIST stops to store the fault information, the proposed BIRA tries to reduce the maximum number of repair solution candidates. In addition, the proposed BIRA uses spare memory as an AMT and stores the small mapped addresses into the fault-storing CAMs. For these reasons, the proposed BIRA has a smaller area overhead than BRANCH analyzer.

VI. EXPERIMENTAL RESULT

As mentioned in Section I, the three main performance criteria of BIRA are the area overhead, repair rate, and analysis speed. Furthermore, the spare memory availability is considered in the proposed BIRA. In order to estimate the performance of the proposed BIRA, we developed a simulation tool in C-language called a redundancy analysis simulation (RAS). RAS generates random faulty addresses according to various inputs. After executing the selected BIRA algorithms for these faulty addresses, RAS generates the output data: the sizes of the AMT, storage information, repair rates, CPU time, clock cycles, and fault statistics.

First of all, both the sizes of the AMT and the spare row memories are estimated by the RAS. Since the proposed BIRA uses the spare row memories as an AMT, it is important that the area of the AMT is smaller than the area of the spare row memories. For an $M \times N$ memory using 2-D spare architecture with $R_s$ row spares and $C_s$ column spares, the area of the AMT and the area of the spare row memories are calculated by (3) and (4), respectively. $A_{\text{AMT}}$ of (3) represents the number of bits required as shown in Fig. 2. Because the architecture of the spare memory is the same as that of the main memory, $A_{\text{row, spare memories}}$ of (4) is obtained. Fig. 9 shows a comparison of the area of the AMT to that of the spare row memories with different memory sizes and amount of spare memory. The area of the spare row memories is smallest when $M = 512$ and $N = 2048$. Because all the areas of the AMT are smaller than the smallest number of spare row memories, however, the proposed BIRA uses the spare row memories as an AMT without any constraints

$$
A_{\text{AMT}} = (1 + \log_2 R_s(C_s + 1) + \log_2 C_s) \times R_s
\times (1 + \log_2 C_s(R_s + 1) + \log_2 R_s) \times C_s
\quad (3)
$$

$$
A_{\text{row, spare memories}} = R_s \times M.
\quad (4)
$$

For the next main feature of BIRA, the storage information is estimated by using RAS. It is difficult to compare the area overhead for various BIRAs without realizing the hardware. Nevertheless, previous research has estimated the storage requirements for various BIRAs in order to compare their area overhead [5], [12], [15]. Although the area of the storage requirements is not the same as the area of BIRA, many studies have adopted the storage requirements as the area overhead of BIRA because the storage cells dominate the silicon area of the BIRA circuit. Therefore, the storage requirement is calculated in order to estimate the hardware overhead since the proposed BIRA uses storage cells.

For an $M \times N$ memory using 2-D spare architecture with $R_s$ row spares and $C_s$ column spares, the storage requirements for each BIRA can be calculated as shown in (5)–(13). $A_{\text{spare register}}$ of (5) represents the number of bits required to store the repair solution. $A_{\text{LRM}}$, $A_{\text{ESP}}$, $A_{\text{CRESTA}}$, $A_{\text{ISF}}$, $A_{\text{SFCC}}$, $A_{\text{BRANCH}}$, $A_{\text{PROPOSED}}$, which are shown in (6)–(13), represent the number of bits required for each BIRA. This comparison type for the area overhead among different BIRAs has already been introduced as part of previous studies [5], [12], [15].

$$
A_{\text{spare register}} = (\log_2 M + 1) \times R_s + \log_2 N + 1) \times C_s
\quad (5)
$$

$$
A_{\text{LRM}} = m \times n + \left[(\log_2 M + 1) + \log_2 (n + 1)\right] \times m
\quad (6)
\quad + \left[(\log_2 N + 1) + \log_2 (m + 1)\right] \times n
\quad + A_{\text{spare register}}
$$

where

$$
m = (R_s \times (C_s + 1) + R_s)
$$

$$
n = (C_s \times (R_s + 1) + C_s)
$$

$$
A_{\text{ESP}} = (R_s + C_s) \times \log_2 M + 1 + \log_2 N + 1 + A_{\text{spare register}}
\quad (7)
$$

This article has been accepted for inclusion in a future issue of this journal. Content is final as presented, with the exception of pagination.
Since the proposed BIRA uses spare memories as overhead of BRANCH is smaller than the area overhead of a branch of the binary search tree. Because BRANCH does a hybrid method that analyzes all nodes concurrently within overhead is smaller to that of CRESTA. BRANCH focuses on RsCs search tree, focus on low area overhead. Because they store implemented. ISF and SFCC, which are based on a binary proposed BIRA achieve optimal repair rates. CRESTA requires are not optimal. CRESTA, ISF, SFCC, BRANCH, and the results shown in Table I.

\[ A_{\text{CRESTA}} = A_{\text{spare\_register}} \times (Rs + Cs)/(Rs! \times Cs!) \]  
\[ A_{\text{ISF}} = 2RsCs \times (\log_2 M + \log_2 N + 1) \]  
\[ + 2RsCs \times \left( \log_2 Rs + \log_2 Cs \right) \]  
\[ + A_{\text{spare\_register}} \]  
\[ A_{\text{SFCC}} = (Rs + Cs) \times (\log_2 M + \log_2 N + 1) \]  
\[ + (Rs + Cs) \times (\log_2 Rs + \log_2 Cs) \]  
\[ + (Rs \times (Cs - 1) + Cs \times (Rs - 1)) \]  
\[ \times (\log_2 (\max(M, N)) + \log_2 (Rs + Cs) + 2) \]  
\[ + A_{\text{spare\_register}} \]  
\[ A_{\text{BRANCH}} = (Rs + Cs) \times (\log_2 M + \log_2 N + 3) \]  
\[ + (Rs + Cs) \times (\log_2 Rs + \log_2 Cs) \]  
\[ + (Rs \times (Cs - 1) + Cs \times (Rs - 1)) \]  
\[ \times (\log_2 (\max(M, N)) + \log_2 (Rs + Cs) + 2) \]  
\[ A_{\text{[20]}} = 2RsCs \times (\log_2 M + \log_2 N + 2) \]  
\[ A_{\text{PROPOSED}} = 2RsCs \times (\log_2 Rs(Cs + 1)) \]  
\[ + \log_2 Cs(Rs + 1) + 1 + A_{\text{small\_spare\_register}} \]  
where \[ A_{\text{small\_spare\_register}} = Rs \times \log_2 Rs(Cs + 1) \]  
\[ + Cs \times \log_2 Cs(Rs + 1). \]  

\[ A_{\text{PROPOSED}} \] of (13) can be derived from the fault-storing CAMs structure, as shown in Fig. 3. Although the proposed BIRA uses a spare register like other BIRAs, the area overhead of the spare register that is used by the proposed BIRA is different from that of other BIRAs. Since the proposed BIRA only handles the small addresses in the RA procedure, the area overhead related to the spare register is dependent on the number of spare memories rather than on the size of the memory. An area estimation with different spare memories for various BIRAs with \( M = 1024 \) and \( N = 1024 \) is shown in Fig. 10. The redundancy configurations varied from two row spares and two column spares (2 × 2) to seven row spares and seven column spares (7 × 7). ESP focuses on minimizing the area overhead of storage requirements with a simple RA algorithm. However, the repair rate of ESP is not optimal because of its excessive omission of faulty information. LRM uses a small size failure bitmap to reduce area overhead. In [20], spare memory is used to record the number of faults in that line and requires small fault address buffers to store the faulty addresses. Because both LRM and [20] are based on the RM algorithm, their repair rates are not optimal. CRESTA, ISF, SFCC, BRANCH, and the proposed BIRA achieve optimal repair rates. CRESTA requires the highest area overhead since in parallel subanalyzers as much area overhead as entire cases of possible solutions is implemented. ISF and SFCC, which are based on a binary search tree, focus on low area overhead. Because they store only 2RsCs faults regarding the must-repair line, the area overhead is smaller to that of CRESTA. BRANCH focuses on a hybrid method that analyzes all nodes concurrently within a branch of the binary search tree. Because BRANCH does not require the amount of storage for spare registers, the area overhead of BRANCH is smaller than the area overhead of SFCC. Since the proposed BIRA uses spare memories as an AMT, it stores only short addresses into the fault-storing CAMs. The proposed BIRA requires on average 31.9% smaller storage space in comparison to BRANCH when \( M = 1024, N = 1024, Rs = 2–7, \) and \( Cs = 2–7. \) Therefore, the area overhead of the proposed BIRA is the smallest among BIRAs with an optimal repair rate.

Fig. 11 shows the area estimation with different memory sizes with four spare rows and four spare columns. The memory sizes varied from \( M = 512, N = 512 \) to \( M = 4096, N = 4096. \) The area overhead of most BIRAs, except for the proposed BIRA, is increased when the memory size is increased. Since the area overhead of the proposed BIRA only depends on the number of spare memories, however, it is not changed even when the memory size is varied.

Table I shows a comparison of the area overhead of BRANCH and the proposed BIRA with different memory sizes and different spare configurations in terms of the storage element. The proposed BIRA requires on average 33.9% smaller storage space in comparison to BRANCH, based on the results shown in Table I.

Table I shows a comparison of the area overhead of BRANCH and the proposed BIRA with different memory sizes and different spare configurations in terms of the storage element. The proposed BIRA requires on average 33.9% smaller storage space in comparison to BRANCH, based on the results shown in Table I. Typically, the area of a BIRA circuit is mainly dominated by the storage elements. However, we implemented BRANCH analyzer and the proposed BIRA for a more precise comparison in terms of gate counts. The proposed BIRA was designed at RTL in Verilog HDL and was synthesized at the gate level using Synopsys Design Vision. We designed the proposed BIRA circuit using Synopsys 90-nm generic library.
Table II shows comparisons of the area overhead in terms of gate counts, power consumption, and clock cycle of BRANCH and the proposed BIRA with different memory size. The gate count is defined as the ratio of a NAND gate to BIRA circuit when the NAND gate is the unit gate. The proposed BIRA requires a BIRA circuit that is 38.84%–40.07% smaller in comparison to a BRANCH based on the data shown in Table II. Tables I and II show different experimental results. When \( R_s = 2 \) and \( C_s = 2 \), with various memory sizes, the area overhead with respect to gate counts is larger than the area overhead with respect to storage elements. Because the storage area of the proposed BIRA is not dependent on the memory size, but rather on the number of spare memories, the storage area ratio becomes small for large memory sizes when the number of spare memories is fixed. However, since BIRA circuit is composed of storage elements and non-storage elements (logic circuits), the area of BIRA circuits becomes large when the memory size becomes large. Furthermore, the logic circuits of the proposed BIRA can become large, since the proposed BIRA accesses spare memories and controls the RA procedures. However, the area overhead of the proposed BIRA in terms of the storage element and gate counts is much smaller than that of the BRANCH analyzer, and occupies only up to 61.16% of the area of BRANCH analyzer. Both power consumptions of BRANCH and the proposed BIRA are increased as the memory size becomes large. Because the area overhead of the proposed BIRA is smaller than that of BRANCH, the power consumption of the proposed BIRA is smaller than that of BRANCH. The proposed BIRA consumes 48.06%–62.49% power in comparison to BRANCH power consumption based on the data shown in Table II. Both the proposed BIRA and BRANCH require same delay, which is about 2.5 ns. Therefore, the proposed BIRA achieves low area overhead and power consumption compared to BRANCH without timing penalty.

The next main feature of BIRA is the repair rate. Generally, the repair rate means the ratio of the amount of repaired memory to the amount of tested memory. Most studies including the proposed BIRA, however, adopt a normalized repair rate, which is the ratio of the amount of repaired memory to the amount of repairable memory, as the repair rate. Since the fault-storing CAMs status depends on the fault distributions or the order of the fault detection, RAS randomly generates 100 000 different sets of faulty addresses for each random fault. To check the validity of the repair rates of BIRAs, a sufficient number of different sets of faulty addresses are required.

The repair rates of BIRAs are estimated by RAS when \( M = 1024, N = 1024, R_s = 4, C_s = 4 \), and the number of faults = 1–16, as shown in Fig. 12. When the number of faults is more than 17, the number of repairable memories is nearly 0. The repair rates of all BIRAs with the number of faults from 1 to 8 are optimal repair rates. Because the number of faults is less than the sum of the spare memories, the memories are repaired by any BIRA method regardless of the fault distributions. The repair rates of ISF, SFCC, BRANCH, and the proposed BIRA are optimal repair rates, even though the number of faults ranges from 9 to 17. As described in the previous section, since these BIRAs are based on exhaustive search methods, their repair rates are always optimal repair rates. Because RM, ESP, and [20] are based on heuristic methods, however, they do not guarantee optimal repair rates when the number of faults is greater than the sum of the spare memories.

Before the analysis speed of BIRAs is dealt with, the test time overhead which is caused by stopping BIST is considered. Table III shows the quantitative measurement time overhead to BIST time using the March C-algorithm. If the number of cells of the memory under test is \( n \), for example, BIST time is \( 10n \) using March C-algorithm. According to Table III, the time overhead, i.e., the additional comparison time using spare memory, versus BIST time increases as the size of spare memory increases.
memory increases under the fixed memory size. When the size of memory increases under the fixed spare memory size, the time overhead decreases because the number of memory cells highly affects an increase of BIST time. The proposed BIRA requires on average 0.60% time overhead in comparison to BIST time when \( M = 512–2048, \ N = 512–2048, \ Rs = 2–6, \) and \( Cs = 2–6. \) Therefore, the increase in the time overhead for using spare memory is negligible.

The last main feature of BIRA is the analysis speed. Fig. 13 shows the clock cycles of BIRAs which have optimal repair rates when \( M = 1024, \ N = 1024, \ Rs = 4, \ Cs = 4. \) The average number of clock cycles by RAS for LRM, ISF, SFCC, BRANCH, [20], and the proposed BIRA are 13.76, 300.43, 87.95, 35.30, 15.32, and 55.70 cycles, respectively. Because repair solutions of ESP and CRESTA are derived during BIST operation, there are no additional clock cycles for ESP and CRESTA. LRM has high analysis speed because its operation is greedy and simple. ISF has the largest analysis speed because exhaustive search tree of ISF compares whenever a fault is detected. The analysis speed of the proposed BIRA is higher than that of SFCC, but it is slower than that of BRANCH for the complex cases (i.e., the number of faults is more than 9). Since SFCC adopts a line-based search tree, the search space is reduced and the analysis speed is higher than an existing BIRA based on a search tree, such as ISF. SFCC spends time for checking the line faults and sorting the line faults according to the number of faults in a faulty line. If the number of faulty lines increases, the number of clock cycles of SFCC also increases. The proposed BIRA compares the repair solution candidates to the fault-storing CAM whether or not the faults are located in a faulty line. Because BRANCH analyzes all nodes concurrently within a branch by using the BRANCH analyzer, however, it achieves the highest analysis speed in comparison to other BIRAs including the proposed BIRA.

A different number of faults are injected into each memory at random locations using the Polya–Eggenberger distribution [21], [25]–[27] for more comparisons. Polya–Eggenberger distribution is suitable for modeling integrated circuit yields [25]–[27]. The two distributions represent the cases with cluster faults (\( \lambda = 8 \) and \( \alpha = 2.382 \)) and evenly distributed faults (\( \lambda = 8 \) and \( \alpha = 0.623 \)), respectively. Table IV shows the comparison repair rate and average clock cycles of SFCC, BRANCH, and the proposed BIRA. The repair rates of SFCC, BRANCH, and the proposed BIRA are the optimal repair rates. According to Table IV, the analysis speed for cluster faults (\( \lambda = 8 \) and \( \alpha = 2.382 \)) is larger than that for distributed faults (\( \lambda = 8 \) and \( \alpha = 0.623 \)) because the cluster faults contain many cross line faults and the cross line faults generate complicated cases for RA. In common with the analysis speed for the random faults distribution, we confirm that the analysis speed of the proposed BIRA is higher than that of SFCC, but it is slower than that of BRANCH for Polya–Eggenberger distribution.

Fig. 14 shows the comparison of overall performances, such as the repair rate, RA execution cycles, and storage requirements, for LRM, CRESTA, ISF, SFCC, BRANCH, the proposed BIRA, and ideal BIRA. For an ideal BIRA, the RA execution cycles and the storage requirements are zero, and the repair rate is optimal. LRM does not guarantee the optimal repair rate with large storage requirements, although its RA execution time is fast. CRESTA has the fastest RA execution time and its repair rate is optimal, but it is not used for BIRA due to the largest storage requirements. ISF and SFCC do
not have a fast enough RA execution time with large storage requirements, although its repair rate is optimal. Although BRANDCH has fast RA execution time and its repair rate is optimal, the storage requirements are still large. Although the proposed BIRA needs more RA clock cycles than BRANDCH, it requires the smallest storage among BIRAs with optimal repair rate. Overall, the proposed BIRA is the closest to the ideal BIRA.

VII. CONCLUSION

An outstanding BIRA scheme that uses spare memory as an AMT under the RA procedure has been proposed in this paper. The proposed BIRA scheme greatly reduces the area overhead with an average value of 33.9% less storage in comparison to BRANDCH, since the length of the mapped fault addresses depends only on the number of spare memories. Also, the proposed BIRA scheme guarantees an optimal repair rate by using an exhaustive search method which is based on a combination of the representatives of the faults. Therefore, the proposed BIRA scheme is a solution for embedded memories for systems-on-chip which require optimal repair rates because of their extremely reduced area overhead and repair rates in comparison to other state-of-the-art BIRA schemes.

REFERENCES


Wooheon Kang received the B.S. degree in electrical and electronic engineering from Yonsei University, Seoul, Korea, in 2009, where he is currently pursuing the combined Ph.D. degree from the Department of Electrical and Electronics Engineering. His current research interests include built-in self-repair, built-in self-testing, built-in redundancy analysis, redundancy analysis algorithms, reliability, and VLSI design.

Hyungjun Cho received the B.S., M.S., and Ph.D. degrees from Yonsei University, Seoul, Korea, in 2005, 2007, and 2013, respectively. He has been a Senior Engineer with System LSI AP Development, Samsung, Inc., Gyounggi-do, Korea, since 2013. His current research interests include fault simulation, fault diagnosis, built-in self-repair, built-in self-testing, built-in redundancy analysis, redundancy analysis algorithms, reliability, and VLSI design.

Joohwan Lee received the B.S., M.S., and Ph.D. degrees from Yonsei University, Seoul, Korea, in 2003, 2005, and 2012, respectively. He has been a Senior Engineer with the Test Development Division, Samsung, Inc., Gyounggi-do, Korea, since 2012. His current research interests include fault simulation, fault diagnosis, built-in self-repair, built-in self-testing, built-in redundancy analysis, redundancy analysis algorithms, reliability, and VLSI design.

Sungho Kang (M’87) received the B.S. degree from Seoul National University, Seoul, Korea, and the M.S. and Ph.D. degrees 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., USA and a Senior Staff Engineer with Semiconductor Systems Design Technology, Motorola, Inc., USA. Since 1994, he has been a Professor with the Department of Electrical and Electronic Engineering, Yonsei University, Seoul, Korea. His current research interests include VLSI/SOC design and testing, design for testability, and design for manufacturability.