for most circuits. Applying grouping of POs and using at most five incremental steps (column IFC 5-Step) has been shown to be an efficient tradeoff. The average propagation path length decreases only slightly compared to the n-step approach. At the same time, the run time decreases drastically. The 5-step approach is in most cases even faster than the traditional approach while generating tests of higher quality.

VII. CONCLUSION

In this paper, we presented a set of incremental solving techniques for improving SAT-based ATPG with respect to run time and quality. A run time analysis provided a valuable insight into a state-of-the-art SAT-based ATPG tool and gave the motivation for this paper. We figured out that, firstly, the SAT instance generation often dominates the overall run time and, secondly, testable faults are usually much harder to classify than untestable faults. By incrementally solving only partial instances, both the instance generation time and the instance solving time can be significantly reduced. We additionally proposed an enhancement that allows to generate test pattern for transition faults that have long propagation paths.

REFERENCES


EOF: Efficient Built-In Redundancy Analysis
Methodology With Optimal Repair Rate

Myung-Hoon Yang, Hyungjun Cho, Wooheon Kang, and Sungho Kang, Members, IEEE

Abstract—Faulty cell repair with redundancy can improve memory yield. In particular, built-in redundancy analysis (BIRA) is widely used to enhance the yield of embedded memories. We propose an efficient BIRA algorithm to achieve the optimal repair rate with a very short analysis time and low hardware overhead. The proposed algorithm can significantly reduce the number of backtracks in the exhaustive search algorithm; it uses early termination based on the number of orthogonal faulty cells and fault classification in fault collection. Experimental results show that the proposed BIRA methodology can achieve optimal repair rate with low hardware overhead and short analysis time, as compared to previous BIRA methods.

Manuscript received May 18, 2009; revised October 1, 2009 and January 22, 2010. Date of current version June 18, 2010. This paper was recommended by Associate Editor W. Kuo.

M.-H. Yang is with Flash Development Division, Hynix Semiconductor Inc., Gyeongju-Do 467-781, Korea (e-mail: myunghyang9@hynix.com).
H. Cho, W. Kang, and S. Kang are with the Department of Electrical and Electronic Engineering, Yonsei University, Seoul 120-749, Korea (e-mail: ch0937@soc.yonsei.ac.kr; sudal@soc.yonsei.ac.kr; shkang@yonsei.ac.kr).

Digital Object Identifier 10.1109/TCAD.2010.2044846
I. INTRODUCTION

Cell size of memory devices is rapidly shrinking, while the number of cells integrated in a single memory chip continues to drastically increase. The size of embedded memory cores increases and the yield of systems-on-a-chip (SoC) is strongly dependent on the yield of embedded memory cores. In spite of advanced fabrication technologies, the high densities of memory devices and embedded memories introduce various faults that deteriorate device yield [1]. Faulty cell repair [2]–[4] is a widely used repair scheme for improving the memory yield. Faulty cell repair adds spare rows and columns to the memory to create 2-D redundancy. Most embedded memories of SoCs are tested using built-in self-test (BIST) and repaired using built-in redundancy analysis (BIRA) instead of external automatic test equipment (ATE).

The effectiveness of BIRA methods can be estimated using the repair rate and the normalized repair rate. The repair rate is defined as the ratio of good memories after redundancy analysis on all repairable memories [13]. The repair rate is strongly dependent on the number of unrepairable memories which are included in the total tested memories. Therefore, the normalized repair rate is more appropriate for estimating the performance of BIRA algorithms. For memories with 2-D redundancy, the optimal redundancy allocation problem is nondeterministic polynomial (NP) time complete [5], [6]. Thus, BIRA may require substantial analysis time and a high hardware overhead in order to achieve a high repair rate. Consequently, the normalized repair rate, the analysis time and the hardware overhead, as well as possible trade-offs, should be used to estimate the performance of BIRA methods.

This paper addresses an efficient BIRA method for large embedded memories with 2-D redundancy. The rest of this paper is organized as follows. In Section II, previous work is given. The detailed end of file (EOF) algorithm is described in Section III. The hardware structure and the experimental results are given in Sections IV and V. Finally, Section V concludes this paper.

II. PREVIOUS WORKS

Built-in self repair (BISR) is indispensable for enhancing the yield of SoCs including many embedded memories. In general, BISR executes a test and a diagnosis (fault location) using BIST, and a repair using BIRA. Various BIRA algorithms for memory blocks with 2-D redundancy have been proposed [8]–[21]. The repair analyzer of the comprehensive real-time exhaustive search test and analysis (CRESTA) scheme provides plural sub-analyzers [11]. Each sub-analyzer performs redundancy analysis concurrently in a predetermined spare allocation order using spare rows and columns, as the memory test proceeds. Since CRESTA tries all possible spare allocation orders, it can guarantee the optimal repair rate. Most

III. EOF BIRA METHODOLOGY

This section explains the proposed algorithm, called EOF for Early termination by the number of Orthogonal Faulty cells. The goal of the EOF algorithm is to achieve the optimal repair rate with simple hardware and a short analysis time. The EOF algorithm is based on an exhaustive search of the binary tree. This basic concept was introduced in [22], and the IntelligentSolve algorithm is based on this concept, too. Both algorithms can guarantee the optimal repair rate. While the IntelligentSolve algorithm uses the dynamic must-repair processes to reduce backtracks, the EOF algorithm uses the number of orthogonal faulty cell to limit the analysis time. The following condition can be derived for a memory with \( r \) spare rows and \( c \) spare columns, where \( F_o \) denotes the number of orthogonal faulty cells [13].

**Condition:** If \( F_o \) is greater than the number of spare lines \( (r + c) \), the memory is unrepairable.

The proposed EOF algorithm postpones the spare allocation for the orthogonal faulty cells, repairing them only after all other faulty cells are repaired. The number of backtracks is strongly dependent on the order in which faulty cells are detected. Furthermore, solutions, which are not optimal with respect to the used spare elements, can still achieve the optimal repair rate. For example, while the IntelligentSolve algorithm can find the optimal solution, the IntelligentSolveFirst algorithm can achieve the optimal repair rate. The EOF algorithm exploits the number of orthogonal faulty cells to find the optimal solution with few backtracks, and in addition to that information about detected faulty cells is used to find the first solution faster. The first guideline described in [6] can be used to reduce the number of backtracks for the first solution. Nonorthogonal faulty cells can be further classified into three types as follows.

1) Faulty cells on a faulty row are row faulty cells.
2) Faulty cells on a faulty column are column faulty cells.
3) Faulty cells on both a faulty row and a faulty column are cross faulty cells.

In the proposed method, the EOF algorithm searches the tree with a row-first strategy. Therefore, if row faulty cells are dealt with first, the first solution can be found more quickly. Fig. 1 shows an example for a memory with six faulty cells. In Fig. 1, the "Type" field represents the type of faulty cells: "R" ("C") represents a row (column) faulty cell, "O" represents an orthogonal faulty cell, and "X" represents a cross faulty cell. If spare lines are allocated using a row-first strategy without fault classification, many backtracks are needed to find the first solution. Here the row faulty cell (5, 5) and the column faulty cell (1, 1) should be repaired with a spare row and a spare column, respectively. If the faults 1, 2, ..., 6 are dealt with in their given order first, this solution cannot be reached without backtracking. However, if row faulty cells are repaired before column faulty cells, Fig. 1 can be solved without backtracking. Consequently, faulty cell classification is necessary.

Faulty cell classification can be easily performed on the fault registers. The proposed fault collection is based on that of ESP [6]. However, since the fault collection of ESP finds the essential pivots and the orthogonal faulty cells, it must be modified to classify the faulty cell types. In the proposed fault collection, the fault type is encoded with two bits: "00" represents an orthogonal faulty cell, "10" for row faulty cell, "01" for column faulty cell, and "11" for cross faulty cell, respectively. The most significant bit (MSB) of the faulty cell type field represents whether there are faulty cell sharing the row address. Similarly, the least significant bit (LSB) of the faulty cell type field represents whether there are faulty cells sharing the column address. If no faulty cell shares the row or column address of the corresponding faulty cell, the faulty cell type is "00" (orthogonal faulty cell). If another faulty cell shares the row address, the column address, or both, the faulty cell type is a row faulty cell, a column faulty cell, or a cross faulty cell, depending on the addresses shared with other faulty cells. The faulty cell type fields are initialized with the "00" value which represents an orthogonal faulty cell type.

Whenever a faulty cell is detected, its row address and column address are compared to those of the existing faulty cells in the fault registers. If the detected faulty cell shares a row (column) address with an existing faulty cell, the MSBs (LSBs) of the cell type fields are set in both cells.

After the fault collection, redundancy analysis procedures are performed; the faulty cell types are used to determine which cell to allocate first. First, the EOF algorithm latches the first row faulty cell in the fault register and checks whether it can be repaired by any of the currently allocated spare lines. If the faulty cell is repaired by one or several allocated spare lines, the next faulty cell is latched in the fault register. Otherwise, the faulty cell is queued for repair by pushing a new node with a row-first strategy and the number of available spare row and column lines onto the stack. Before the next faulty cell is latched, the EOF algorithm compares the number of available spare rows and columns to the number of orthogonal faulty cells, as explained in the previous section. If the repair is impossible according to the comparison results, the EOF algorithm backtracks to a node which has an unsearched branch. After all row faulty cells are repaired, similar procedures are performed for the cross faulty cells and the column faulty cells. The EOF algorithm allocates spare rows or spare columns to the remaining orthogonal faulty cells after finishing allocations for all nonorthogonal faulty cells.

This procedure can be explained using the example in Fig. 2. Fig. 2(a) shows the fault registers after the fault collection. Fig. 2(b) shows the spare allocation procedure using the EOF algorithm with fault classification. In the beginning, the first row faulty cell is (4, 3), so a spare row is allocated to this faulty cell. Since there is another row faulty cell (6, 7), a remaining spare row is allocated to this faulty cell. Since there are no more nonorthogonal faulty cells and two orthogonal faulty cells (1, 1) and (2, 2) left, these faulty cells can be repaired with two remaining spare columns, leaving all faulty cells covered. In this example, the EOF algorithm found the first solution without backtracking. Fig. 2(c) shows the spare allocation procedure using the EOF without fault classification. In order to find a solution, the last faulty cell (6, 7), which is part of a faulty row, should be repaired using a spare row, as shown in Fig. 2(b). However, since the results of the fault classification are not applied, spare rows are first allocated to other faulty cells, and the solution can be reduced. Although, the IntelligentSolve algorithm can find the first solution after two backtracks, as shown in Fig. 2(d). After a spare row is allocated to row 1 to repair the first faulty cell (1, 1), the number of available spare rows is decremented by one. Since two faulty cells on Column 5, (4, 5) and (6, 5) are repaired by the dynamic must-repair, the search space can be reduced. Although, the IntelligentSolve algorithm can find the first solution after two backtracks, it performs many dynamic must-repair operations after the search step for every node. In conclusion, the analysis time of the redundancy analysis algorithms including the IntelligentSolve algorithm is strongly dependent on the sequence of the detected faulty cells. Therefore, the proposed fault collection to classify the faulty cell types is very efficient for reducing the analysis time.

Fig. 3 shows a pseudo code for the proposed EOF algorithm. The function EOF.SA() allocates spare lines to repair the faulty cells collected by the fault collection. As previously explained, first, the spare allocation for the row faulty cell is performed. The search_tree() function allocates a spare line
IV. Hardware Structure of Faulty Cell Registers

The BIST applies the test patterns to the memory cell array and detects faulty cells. Whenever the BIST detects a faulty cell, it transfers the faulty cell address to the BIRA. The BIRA performs the fault collection operations to store the faulty address in the faulty cell registers. When the BIST finishes all test procedures, the BIRA starts the redundancy analysis with the information stored in the faulty cell registers. After the RA procedure, the BIRA communicates the RA results (including the repairability of the tested memory cell array and the repair solution) to the external ATE. As previously mentioned, in order to reduce the size of the BIRA circuits, the number of storage cells of the BIRA should be reduced. This section presents an efficient storage structure. The proposed structure can support the proposed EOF algorithm while reducing the storage cells needed.

Fig. 4 shows the structure of the proposed faulty cell registers. In order to achieve the optimal repair rate, 2^rc faulty cell should be stored. The minimum size of the storage cells has already been proven in [6]. Therefore, the IntelligentSolve algorithm [7] selects 2^rc as the content addressable memory (CAM) size for both the row and the column CAMs. Although it is true that 2^rc faulty cells should be stored to achieve the optimal repair rate, it is not necessary to store the whole row and column addresses of the faulty cells; instead pointers are used. The faulty row (column) addresses are stored in the faulty row (column) address registers. By Theorem 2 of [9], the size of the faulty row and column address list registers is r+c. If both the faulty row and column addresses are full, and a new faulty cell not covered by must-repair rule is detected, the memory cell array cannot be repaired.

In the faulty cell list registers, a valid bit represents whether the corresponding faulty cell is valid (1) or not (0). The faulty cell type is encoded by two bits, as explained in the previous section (00: orthogonal faulty cell, 10: row faulty cell, 01: column faulty cell, 11: cross faulty cell). The row and column address pointers represent the pointers for the addresses of the faulty row and column address registers where a row address and a column address of the corresponding faulty cell are stored. Whenever a faulty cell is detected, a valid
bit of a vacant entry of the faulty cell list registers is set to 1 and the row and column addresses are compared to the row and column address registers. If the row (column) address of the incoming faulty cell equals one of the row (column) addresses in the row (column) address registers, MSB (LSB) is set to 1 and the fail count is incremented. Otherwise, the row (column) address of incoming faulty cell is stored in one of vacant entry of the row (column) address registers and the fail count is set to 1.

When the fail count of the row (column) address registers is equal to $c$, the corresponding row (column) is a must-repair line. Therefore, the fail count is set to 0 and the row (column) address is stored in the spare register. Also, the fail counts of the column (row) address registers for the faulty cells on the must-repaired row (column) address are decremented. If the fail count is decremented to 0, that entry can be used to store another faulty address. In the proposed cell registers, the fault collection need not store the whole faulty cells, as opposed to the previous scheme, which stored the individual row and column addresses of each faulty cell. Therefore, the proposed faulty cell registers can reduce the storage cells needed, as compared to previous methods.

V. EXPERIMENTAL RESULTS

This section discusses the experimental results of the proposed EOF algorithm. In order to estimate its performance, the EOF algorithm has been simulated for a $1024 \times 1024$ memory cell array. Each experiment was repeated 10,000 times using a number of randomly injected faults. Since faulty cells are scattered on large memory cell array with uniform distribution, only a few cells can be repaired. The typical fault is a single fault which shares neither row nor column address with other faulty cells. Therefore, injected faulty cells are scattered along the Gaussian distribution to introduce various fault types including single faulty cells, faulty rows, faulty columns, and cluster faults.

As previously mentioned, nonoptimal solutions may achieve the optimal repair rate. In other words, a single solution is the most efficient way to achieve the optimal repair rate. Therefore, if the EOF algorithm stops at the first solution, a solution can be found with very few backtracks. This algorithm is referred to as the EOF stopping at the first solution (EOFSF).

Fig. 5 shows the average backtracks of the EOF and EOFSF algorithm compared to the IntelligentSolve and the IntelligentSolveFirst algorithm. As shown in the curves in Fig. 5, the proposed EOF algorithm needs fewer backtracks.

When there are many faulty cells (from 12 to 20 faulty cells), the EOF algorithm requires fewer backtracks than the IntelligentSolveFirst algorithm does. Although all algorithms require a similar number of backtracks when there are more than 20 faulty cells, most of these cases are unrepairable.

Fig. 6 compares the number of clock cycles of the EOF and the previous methods. Although the analysis speed of the EOF is slower than that of the SFCC, it is much faster than IntelligentSolve. However, the EOF requires small storage elements when compared to the SFCC because the EOF requires only simple fault registers.

As previously mentioned, we used the number of storage cells to estimate the hardware overhead. The hardware overheads of ESP, IntelligentSolve, and SFCC are given by equations in [11]. For the EOF BIRA circuits, hardware overheads (i.e., storage cells) are estimated using the number of bits of the spare registers and the faulty cell registers.

Fig. 7 compares the hardware overhead of the proposed EOF scheme and the previous schemes, based on a $1024 \times 1024$ bit memory cell array, with $(r, c)$ ranging from (2, 2) to (5, 5). Although the hardware overhead of the proposed method is larger than that of the ESP algorithm, it is smaller than those of IntelligentSolve and SFCC. Also, since the operation of the EOF algorithm is much simpler than those of previous
methods, the hardware except the storage elements requires simple logic circuits as compared to previous method.

VI. CONCLUSION

The EOF algorithm can reduce the analysis time in two ways. Using early termination with the number of the orthogonal faulty cells, EOF significantly reduces the search space for the exhaustive search. Furthermore, the fault classification of the EOF algorithm can eliminate unnecessary branches in the search procedure. Since the fault classification can also reduce the number of backtracks for the first solution, the EOF algorithm is very useful for a BIRA solution targeting the optimal repair rate. Experimental results show that the EOF algorithm can achieve an optimal rate within a very short analysis time and that it can be implemented with less hardware overhead than previous methods.

REFERENCES


On Clustering of Undetectable Single Stuck-At Faults and Test Quality in Full-Scan Circuits

Irrth Pomeranz and Sudhakar M. Reddy

Abstract—We demonstrate that undetectable single stuck-at faults in full-scan benchmark circuits tend to cluster in certain areas. This implies that certain areas may remain uncovered by a test set for single stuck-at faults. We describe an extension to the set of target faults aimed at providing a better coverage of the circuit in the presence of undetectable single stuck-at faults. The extended set of target faults consists of double stuck-at faults that include an undetectable fault as one of their components. The other component is a detectable fault adjacent to the undetectable fault. We present experimental results of fault simulation and test generation for the extended set of target faults.

Index Terms—Full-scan, stuck-at faults, test generation, test quality, undetectable faults.

I. INTRODUCTION

Fault models used as targets for test generation are expected to guide the generation of tests that will be effective in detecting defects. Thus, a test set generated for single stuck-at faults in a full-scan circuit [1]-[7] is expected to detect defects associated with the sites of stuck-at faults. Test sets that contain several different tests for each fault (or detection test sets) are expected to increase the likelihood of detecting defects associated with the sites of target faults [8]-[16]. When a single stuck-at fault is undetectable, it leaves an uncovered site in the circuit. As we demonstrate later, in benchmark circuits, undetectable single stuck-at faults tend to cluster in certain areas. This implies that certain areas of the Manuscript received November 15, 2009; revised January 28, 2010. Date of current version June 18, 2010. The work of I. Pomeranz and S. M. Reddy was supported in part by the Semiconductor Research Corporation under Grant 2007-TJ-1643 and 2007-TJ-1642, respectively. This paper was recommended by Associate Editor, F. Lombardi.

I. Pomeranz is with the School of Electrical and Computer Engineering, Purdue University, West Lafayette, IN 47907 USA (e-mail: pomeranz@ecn.purdue.edu).

S. M. Reddy is with the Department of Electrical and Computer Engineering, University of Iowa, Iowa City, IA 52242 USA (e-mail: reddy@engineering.uiowa.edu).

Digital Object Identifier 10.1109/TCAD.2010.2046448

Authorized licensed use limited to: Yonsei University. Downloaded on June 23,2010 at 07:51:03 UTC from IEEE Xplore. Restrictions apply.