An efficient approach for memory repair by reducing the number of spares

Vrezh Sargsyan
Synopsys Armenia
vsargsya@synopsys.com

Valery Vardanian
Synopsys Armenia
vvardani@synopsys.com

Samvel Shoukourian
Synopsys Armenia
samshouk@synopsys.com

Yervant Zorian
Synopsys USA
zorian@syopsys.com

Avetik Yessayan
Synopsys Armenia
avetiky@synopsys.com

Abstract

In this paper, we proposed an approach to reduce the hardware required by the conventional algorithm CRESTA (Comprehensive Real-time Exhaustive Search Test and Analysis) by trying to share the hardware between different sub-analyzers. Experiments with synthesis for the case of three spare rows and three spare columns evidenced for saving at least ***15%*** *of hardware, and for the case of two spare rows and two spare columns – about* ***7%****.*

1. Introduction

The portion of embedded memories in System-on-Chips (SoCs) increases very fast (it was expected to exceed 94% by 2014 [1]), and yield prediction for SoC becomes very difficult and important (the yield of embedded memories determines the yield of SOCs). Too much redundancy is a waste of silicon and an additional possibility for occurrence of a defect/fault, too little redundancy means impossibility to repair a defect/fault leading to a significant loss of yield. The classical way to improve yield is to (see [2]-[9]): 1. Detect and locate memory defects/faults by an efficient test algorithm; 2. Apply a simple and efficient algorithm for allocation of spare (redundant) rows/columns enabling to repair the memory. It was noted in [2] that the greater the fault location "coverage", the higher the repair efficiency, and hence, the obtained yield. In [2], three types of enhancements were proposed for fault localization coverage that requires the knowledge of the memory design and its compilation: 1. Adding dedicated infrastructure IP modes to the memory IP design called test modes (read margin control, stress test, ground and substrate isolation, adjustable setup and hold time, supply voltage operation range, and self-timing clock bypass); 2. Leveraging knowledge of its design information, specifically it's scrambling, to determine topological data background patterns. The infrastructure IP uses background patterns to locate coupling faults between cells and between bit-lines and to detect memory periphery weaknesses; 3. Fault detection and localization algorithms optimization.

To repair the memory efficiently, a phase is needed for determination of the best allocation of redundant elements. Redundancy allocation is simple and straightforward if only one type of spare elements is used (only columns or rows). However, the problem becomes too difficult (known as ***NP***-complete) if the design uses both types of spare elements (columns and rows). As noted in [2], [7], [9], there are two types of redundancy allocation algorithms, primitive and "intelligent". The primitive algorithm predefines a predetermined sequence of allocations of spare elements and is based on failure history. The intelligent algorithm performs analysis at every step before the allocation of a spare element. To select the best algorithm for spare allocation we need to develop a methodology for evaluation of the efficiency of an algorithm.

Recently, due to the emerging deep sub-micron technologies used in the design and development of SOCs, increasing of yield by using spare elements and corresponding simple and "intelligent" spare allocation algorithms with nearly or exactly ***100 %*** repair coverage, gains wide popularity.

In [6], the authors proposed an algorithm, called CRESTA (Comprehensive Real-time Exhaustive Search Test and Analysis), for repair of bit-oriented memories. It makes use of parallel engines (sub-analyzers) allowing to process all possible repair strategies simultaneously. Each repair strategy is processed in one sub-analyzer in parallel and enabling find a solution for a bit-oriented memory repairable with the given spare elements. In [8], the authors generalized the method of CRESTA for word-oriented memories to detect multi-bit faults in a word.

The known Built-In-Redundancy-Analysis (***BIRA***) algorithms have both advantages and disadvantages. CRESTA achieves ***100 %*** repairability among the repairable memories with fast analyzing speed. However, its main disadvantage is its area overhead that increases exponentially with respect to the number of redundancies.

In this paper, first we propose a general approach how the problem of repairing memories with a greater number of redundancies than, e.g. two redundant rows and two redundant columns, that is currently widely accepted for nowadays large enough memories, can be reduced to that of a small number redundancies. However, as it is expected, the portion and size of memories in SoCs is rapidly increasing and it is not excluded that a greater number of redundancies might be used in future memory instances and, thus, the need to develop simple and efficient algorithms to repair memories with more than two redundant rows and more than two redundant columns. We will apply this new approach for development of new algorithms for three (two) redundant rows and three (two) redundant columns and show how to reduce the development of the repair algorithm to a specific case when it is based on the usage of CRESTA with two redundant rows and two redundant columns. This approach seems efficient since the hardware overhead is not too much as compared to CRESTA. In particular, CRESTA was implemented in its conventional form, and then we applied the described method of reduction. According to this notion, we can save hardware in CRESTA by merging some branches that have the same sub-strategy of repair. For example, suppose we have to repair the memory with two spare rows and two spare columns. The ***BIRA*** engine consists of 6 parallel sub-analyzers corresponding respectively to the following repair strategies: ***CCRR, CRCR, RCCR, CRRC, RCRC, RRCC***. Each repair strategy describes how the memory will be repaired. For example, repair strategy CRCR denotes the following repair steps: the first detected fault is repaired by means of a spare column, the second fault that is not covered by the used spare element is covered by a spare row, the third fault not covered by the previously used spare elements will be covered by the second spare column, and the next detected fault that was not covered by the used spare elements will be covered by the second available spare row. The symbol C means the fault will be repaired by an available spare column, R - row. Note that in repair strategies ***CRCR, RCCR, CRRC, RCRC*** we can find similar segments ***(RC, CR)*** the hardware of which can be designed to be shared between different strategies.

2. General method of reduction

We will concentrate our attention on the problem of developing efficient and perfect (with ***100 %*** of repair coverage) algorithms for the repair of memories with at least two redundant rows and at least two redundant columns. For future technologies and very large memories two redundant rows and two redundant columns may appear to be insufficient to repair the memory with sufficiently greater number of faults to obtain sufficiently acceptable yield for SoCs.

Suppose we have a memory of size nxn with ***r (***respectively, ***c)*** redundant rows (columns). Suppose we have algorithms efficiently implemented before for repair of memories with ***r – 1(***respectively, ***c – 1)*** spare rows (respectively, columns).

To repair the memory with its hardware as low as possible, the algorithm should be very simple with very easy implementation. We propose the following. As soon as the first fault is detected we propose to repair it exhaustively, i.e. use two parallel sub- analyzers, one repairing the fault with the first available spare column, and the second one - repairing the fault with the first available spare row (see Fig. 1).

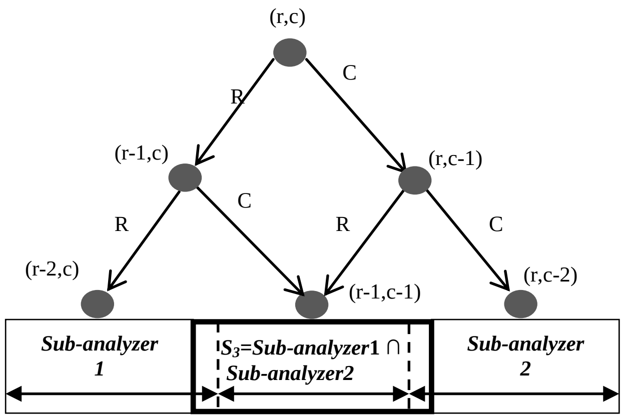

Figure 1. ***The process of reduction***.

Thus, we have two branches for our algorithm. The number of available spare elements after the first repair will be: ***r*** spare rows, ***c-1*** spare columns for the first branch, and ***r-1*** spare rows, ***c*** spare columns for the second branch (see Fig.1). Next, when the second fault is detected needing repair, the first (respectively, second) sub-analyzer of the ***BIRA*** engine tries to repair the memory with an available spare row (respectively, column). Thus, after the second step of the repair, for both sub-analyzers the number of available spare rows and spare columns will be the same, namely, ***r-1*** spare rows, and ***c-1*** spare columns, respectively. Consequently, the problem of repairing the memory with ***r*** spare rows and ***c*** spare columns is now reduced to the problem of repairing the memory with a lower number of spare elements, ***r-1*** spare rows, and ***c-1*** spare columns. If we find similar segments in both sub-analyzers we can share their hardware and save possibly a significant amount of area. Thus, the problem of repairing the memory with ***r*** spare rows and ***c*** spare columns can be reduced to repairing the memory with a smaller number of spares, namely, with ***r– 1*** spare rows and ***c – 1*** spare columns (see sub-circuit $\mathbf{S_3}$ in Fig.1). There are two sub-analyzers working in parallel, and the hardware of these two sub-analyzers may have parts that can be used by both sub-analyzers and save hardware (see Fig.1). We will explain the proposed approach by considering CRESTA [6] that has 6 branches of sub-analyzers processing in parallel and repairing the memory always when it is repairable.

CRESTA is known as efficient only for the cases when the memory has very small number of redundancies, and its hardware grows exponentially since the number of sub-analyzers itself grows exponentially with respect to the number of spare elements. Details of this compaction are shown in the next section. We will show that in parallel branches of CRESTA, it appears that some branches contain similar segments and the hardware corresponding to these segments can be shared thus saving some area.

3. Consumption diagram

Since the notion of Consumption Diagram (***CD***) [7] lies in the basis of the proposed approach prompting the main steps we will cite some details from [7]. For memories with ***m*** spare rows and ***n*** spare columns, the flow of redundancy consumption during a repair process may be depicted as a directed path of length ***m+n*** starting from the top of the following diagram, called in the sequel ***consumption diagram*** (see ***Fig. 2***), and ending at the bottom. Each vertex of the graph has two outgoing edges, one – labeled with symbol "***R***" (***R*** – row) and the second one – labeled with symbol "***C***" (***C*** - column) showing that now any next fault detected by the ***BIST*** engine can be repaired in two ways, either by a row, or column. Each vertex is labeled with a pair of numbers (***i,j***) where ***i*** (respectively, ***j***) shows the number of available spare rows and columns, respectively.

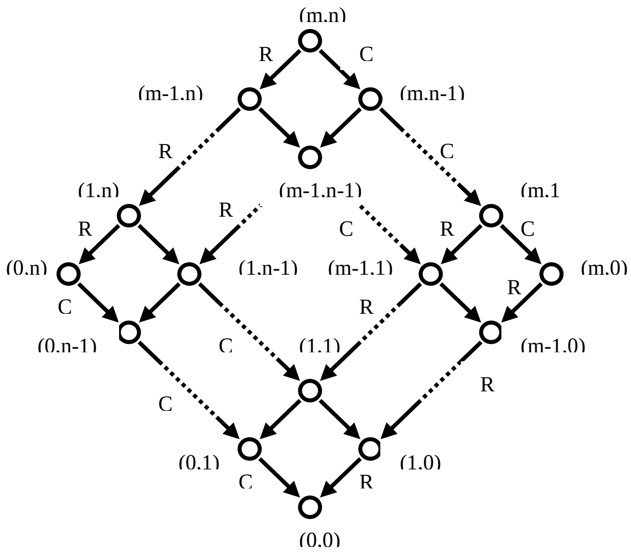

Figure 2. Consumption Diagram

A repair strategy (predetermined sequence of an actual repair by ***m*** spare rows and ***n*** spare columns) corresponds to a single directed path in Fig. 2. Obviously there exists a one-to-one correspondence between all ***C(n+m, m)*** single directed paths of the diagram in Fig. 2 and all possible algorithms of repair that implement a predetermined order of using ***m*** spare rows and ***n*** spare columns.

The repair ability of each sub-analyzer is usually low but altogether they achieve ***100 %*** of fault repair (in the case when the memory is repairable). The algorithm CRESTA, proposed in [6], uses exhaustively all possible ***C(n+m, m)*** paths in parallel. This exhaustive approach is acceptable only for smaller values of ***m*** and ***n***. When ***m*** and ***n*** are not small, the exponential growth of the number of algorithms leads to an unacceptable hardware complexity.

In this paper, we make use of the structure of ***CD*** and take into account the existence of symmetric segments in strategies that was used in the proposed implementation.

4. Implementation

The BIRA scheme consists of 4 base modules (RR,CC,RC,CR). In Fig. 3, the diagram branches with "*" show the common parts, that were shared. There is a signal second_phase indicating that a new fault should be analyzed taking account the address of already covered faults. For example, for the module RR, the signal second_phase takes value of "logic high", when the first two defects are covered with different rows. When the value of second_phase is "logic high", the RR module works as a part of branch CC-RR (RR* in Fig. 3). In Fig. 3 the multiplexer conditionally illustrates the intermediate logic that switches base modules. Each base module is a simple finite state machine that takes as input data the faulty cell row and column addresses, and generates repair signature. In Fig.3, the parts of branches with the same color indicate the possibilities of hardware sharing. For example, the sub-analyzers RC-2 (respectively, CR-3) and CR* (respectively, CR*) are colored in blue (respectively, yellow) can be used for hardware sharing.

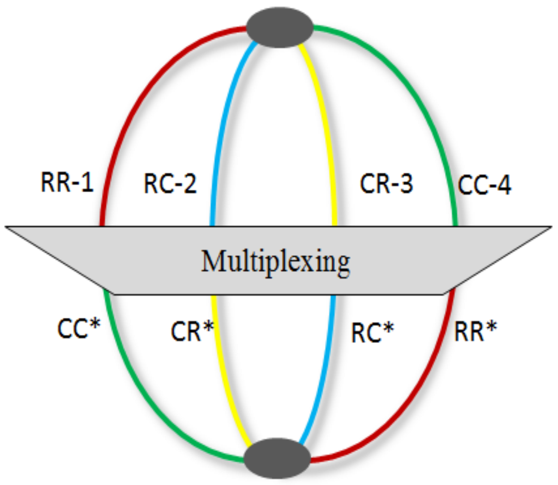

Figure 3. Possibilities of sharing sub-analyzers

5. Experimental results

To evaluate the efficiency of the proposed method, the BIRA circuits were implemented on the basis of the conventional CRESTA algorithm [6] and the proposed algorithm. The circuits were implemented for memories with 3 redundant columns and 3 redundant rows (3X3), and for memories with 2 redundant columns and 2 redundant rows (2X2).

The synthesis results before and after optimization are presented in Table 1. The designs were implemented in a 28-nm technology using nominal settings on all parameters. As a result, we have saved the hardware for more than 15 % for the case 3x3. Moreover, we optimized CRESTA for the case 2x2 as well getting approximately 7 % saving of hardware.

Table 1. Experimental results

Project type	Circuit area (μm)		Improvement Ratio
	before	after	
CRESTA 2X2	3316.87	3093.16	**~6.74%**
CRESTA 3x3	10370.94	8804.88	**~15.1%**

6. Conclusions

In this paper, we proposed an approach to enhance the well-known CRESTA algorithm by sharing the hardware between different sub-analyzers.

For CRESTA 3x3, the hardware was reduced by at least ***15%,*** and for CRESTA 2x2 – by nearly ***7%***.

We believe that for higher number of spare elements the area saving will be too much higher. The approach can be generalized to reduce the problem of repairing memories with higher number of spare elements to that of smaller number of spare elements.

7. References

[1] International Technology Roadmap for Semiconductors, Test and Test Equipment, 2000.J. Clerk Maxwell, pp.68-73.

[2] S. Shoukourian, V. A. Vardanian, Y. Zorian, SoC yield optimization via an embedded memory test and repair infrastructure, IEEE Design & Test of Computers, vol.21, May-June, 2004, pp. 200-207.

[3] R. Rajsuman, “Design and test of large embedded memories: An overview,” IEEE Des. Test Comput., vol. 18, no. 3, pp. 16–27, May 2001.

[4] Y. Zorian and S. Shoukourian, “Embedded-memory test and repair: Infrastructure IP for SoC yield,” IEEE Des. Test Comput., vol. 20, pp.58–66, May–Jun. 2003.

[5] I. Kim, Y. Zorian, G. Komoriya, H. Pham, F. P. Higgins, and J. L. Lewandowski, “Built in self repair for embedded high density SRAM,” in Proc. Int. Test Conf. (ITC), Oct. 1998, pp. 1112–1119.

[6] T. Kawagoe, J. Ohtani, M. Niiro, T. Ooishi, M. Hamada and H. Hidaka, “A Built-in Self-repair Analyzer (CRESTA) for Embedded DRAMs,” Proc. IEEE Int’l Test Conference, 2000, pp.567-574.

[7] S. Shoukourian, V. Vardanian and Y. Zorian, “An Approach for Evaluation of Redundancy Analysis Algorithms”, Proc. IEEE MTDT Workshop, pp. 51-55, San Jose, 2001.

[8] X. Du, S.M. Reddy, W.-T. Cheng, J. Rayhawk, and N. Mukherjee, “At-Speed Built-in Self-Repair Analyzer for Embedded Word-Oriented Memories,” in Proc. of the 17th International conference on VLSI Design, pp. 895-900, 2004.

[9] S. Shoukourian, V. A. Vardanian, Y. Zorian, A methodology for design and evaluation of redundancy allocation algorithms, in Proc. IEEE VLSI Test Symposium, Napa Valley, USA, 2004, pp. 249-255.