Repair Analysis for Embedded Memories Using Block-Based Redundancy Architecture

Štefan Krištofík, Elena Gramatová, *Member, IAENG*

***Abstract*— Capacity and density of embedded memories have rapidly increased therefore they have higher probability of faults. As a result, yield of system-on-a-chip designs with embedded memories drops. Built-in self-repair is widely used to improve manufacturing yield by replacing faulty memory cells with redundant elements. Most approaches perform reconfiguration on the row/column level. Block-based redundancy architectures divide memory and redundancies into blocks, performing reconfiguration on the block level and offering more efficient usage of the redundant elements. However, existing block-based approaches implement simple redundancy analysis algorithms which lead to non optimal repair rates. This paper proposes a new approach for block-based redundancy architectures, improving repair rates of previous approaches by utilizing a fast hybrid redundancy algorithm with low area overhead and optimal repair rate.**

***Index Terms*—embedded memory, yield, built-in self-repair, built-in redundancy analysis**

I. Introduction

THE density of modern system-on-a-chip (SoC) designs is growing rapidly, so is the capacity and density of memories embedded within them. As a consequence, embedded memories have higher probability of faults and their manufacturing yield drops. Since embedded memories are occupying the majority of nowadays SoCs area (90 % according to [1]), they are the main source of faults in SoCs and they also dominate the overall SoC yield.

To improve reliability and manufacturing yield, the most widely used approach is to add some redundancy to the memories. Faulty memory cells are replaced by redundant elements. In the case of SoC, memory testing and repair are provided in the chip itself (built-in self-repair, BISR), because it is more cost effective than using external test equipment.

The BISR approach has three main functions. First, memories are tested for various types of faults by built-in self-test (BIST). Based on the fault information provided by BIST, memories are analyzed by the redundancy analysis (RA) algorithms which generate repair solutions for memories. Repair solutions consist of information on which redundant elements are to be addressed instead of every single faulty cell. Execution of RA algorithms is controlled by built-in repair analysis (BIRA). Repair solutions provided by BIRA are applied to memories by address reconfiguration (AR) which ensures that respective redundant elements are addressed instead of faulty memory cells.

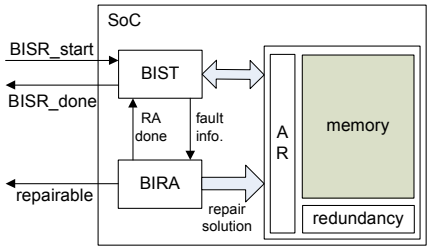

Fig 1. Built-in self-repair architecture.

Fig. 1 shows the BISR architecture and its three main blocks: BIST, BIRA and RA. BISR function is started by activating *BISR_start* signal. When BISR function is finished, *BISR_done* signal is activated. If memory can be repaired, *repairable* signal is active. In case memory is un-repairable, *repairable* signal is not active. This may be due to insufficient number of redundant elements when fault density is high or due to usage of RA algorithm with non-optimal repair rate.

Fault information from BIST is processed in BIRA. This information is provided in the form of fault locations in the memory. Three main features of BIRA are area overhead, repair rate and analysis speed [2]. Smaller area overhead reduces chip production cost. Low repair rate impacts yield negatively and speed affects the cost of repair. Repair rate represents the ability of an RA algorithm to find a repair solution for the memory and is defined as follows:

$$\text{repair rate} = \frac{\text{good memories after BIRA}}{\text{\# of total memories}}$$

The number of total memories includes both repairable and un-repairable memories. Un-repairable memories can be produced by various factors [2] and this may negatively influence the repair rate of RA algorithms that are evaluated using this value. Normalized repair rate was introduced in [2] and is defined as follows:

$$\text{normalized repair rate} = \frac{\text{good memories after BIRA}}{\text{\# of repairable memories}}$$

It is not dependent on the aforementioned factors, and therefore is more appropriate to evaluate the RA algorithms.

Manuscript received March 19, 2012; revised April 15, 2012. This work was supported in part by the Slovak Science Grant Agency (VEGA 1/1008/12).

Š. Krištofík and E. Gramatová are with the Faculty of Informatics and Information Technologies, Institute of Computer Systems and Networks, Slovak Technical University, Bratislava 842 16, Slovakia (e-mail: kristofik@fiit.stuba.sk, gramatova@fiit.stuba.sk).

ISBN: 978-988-19252-1-3
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

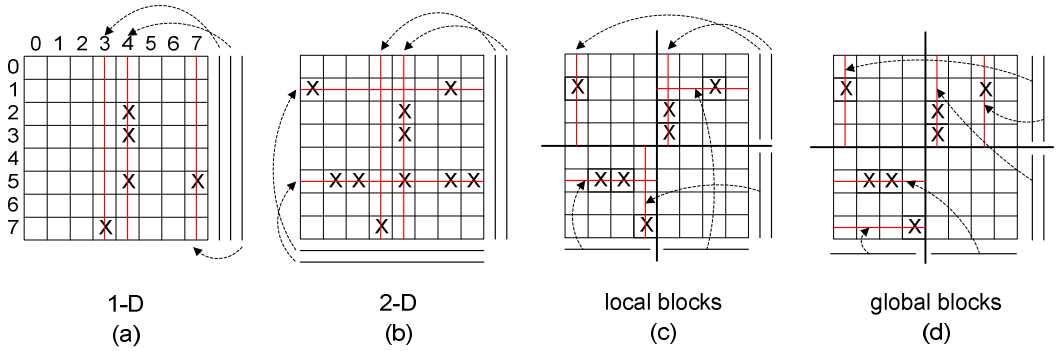

Fig 3. Redundancy architectures.

Optimal repair rate is achieved when the normalized repair rate is 100 % [2]. Ideal BIRA has optimal repair rate with zero area overhead and analysis time. Finding a repair solution is NP-complete problem [3] and various BIRA approaches have been proposed that have tried to balance these three main factors [2], [4] - [6]. Some general observations on how to obtain optimal repair rate with BIRA are listed in [2].

To be able to analyze fault information, BIRA needs to store it in some way. Approaches that utilize fault bitmaps of various sizes to store fault information have been shown to have negative impact on repair rate since some fault information is often omitted. Better repair rates were achieved by using of storage registers and content-addressable memories (CAM) [2], [5], [6].

Two important tasks of BIRA are fault collection (from BIST) and redundancy analysis of redundant elements. Both tasks are performed by RA algorithm. Based on the time of execution of these two tasks, we distinguish three types of RA algorithms [2]. Fig. 2 shows a comparison of RA algorithms. Static RA algorithms perform RA after all fault information has been collected and stored in fault bitmap. This results in increased time it takes the algorithm to finish (slower analysis speed) and high area overhead. Static RA algorithms are neither suitable nor used in built-in solutions. Dynamic RA algorithms perform RA in parallel with fault collection. When BIST is finished, RA is also finished. Time to finish is shorter and area overhead is lower than in static RA algorithms, but repair rates are not optimal. Hybrid RA algorithms perform RA concurrently with fault collection, but after BIST finishes, RA continues for some time. Using hybrid RA algorithms, optimal repair rates can be achieved at the cost of slight increase in time and area overhead compared do dynamic RA algorithms.

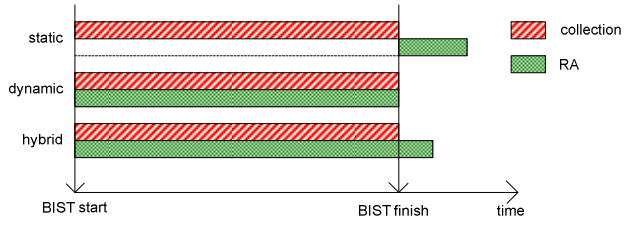

Fig 2. Classification of RA algorithms.

The types of redundant elements used in memories can be classified into four types as shown in Fig. 3 on memories of size 8x8. Faulty cells are denoted by the "X" symbol. Redundant elements are represented as solid lines at the sides of memories and repair solutions are denoted by strike-through lines in memory arrays. 1-D redundancy architectures incorporate only one type of redundant elements (rows or columns) into memories. Fig. 3 (a) shows an example where only redundant columns are used. While this approach is easy to implement, RA algorithms are simple and area overhead is low, it suffers from non-optimal repair rates in larger memories. 2-D redundancy architectures are the most widely used and most BIRA approaches are based on it. Both redundant rows and columns are added to the memory as shown in Fig. 3 (b). With 2-D redundancy, optimal repair rates can be achieved at the cost of enlarging area overhead needed to implement more complex RA algorithms. Local and global block-based redundancy architectures were proposed recently. They are based on divided word line (DWL) and divided bit line (DBL) architectures [5], [6]. Memory and redundancies are divided into number of blocks (divided blocks) and reconfiguration is performed on the block level instead of row/column level as in 1-D and 2-D redundancy architectures. For example, memories are divided into 4 blocks and redundancies are divided into 2 blocks, in Fig. 3 (c) and (d). If local blocks are used, blocks of redundancies are restricted to be used only in their respective memory blocks [5]. Global blocks can be used in any memory block therefore providing better repair rates than local blocks [6].

Among various RA algorithms for 2-D redundancy architectures, selected fail count comparison (SFCC) showed the best performance in matters of repair rate, area overhead and analysis speed [2]. SFCC is a hybrid RA algorithm that builds a fault line-based searching tree that searches through repair solution space faster than previous approaches based on cell-based searching trees thus improving analysis speed. Its fault storing structure is based on CAMs and it focuses on reducing storage requirements by discarding some overlapping fault addresses. Among a few known RA algorithms for block-based architectures, modified essential spare pivoting (MESP) showed the highest but not optimal repair rate [6]. MESP is based on essential spare pivoting (ESP) algorithm [3], which focuses on low area overhead and fast analysis speed, but cannot guarantee optimal repair rates in 2-D architecture. It is a dynamic RA algorithm that

ISBN: 978-988-19252-1-3
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

builds a repair solutions based on identification of pivots (first faults found in a row or column in divided blocks) often omitting some fault information which leads to non-optimal repair rates.

RA algorithms used in known block-based redundancy architectures achieved non-optimal repair rates [5], [6]. In this paper, we propose a block-based redundancy architecture with global redundancy, which uses modified SFCC (MSFCC) algorithm based on SFCC to improve repair rates of previous block-based approaches. Table 1 clarifies the idea proposed in this paper.

II. Proposed Built-In Repair Analysis Approach

We use global block-based redundancy architecture as in MESP and classification of memory faults based on SFCC. Single fault does not share either row or column address with any other fault in the divided memory block. Sparse fault shares its row (column) address with at least one other fault in the divided memory block, but not more than C (R) faults, respectively. C and R denotes the number of available redundant column and row blocks in the memory, respectively. Must-repair fault shares its row (column) address with more than C (R) faults in the divided memory block.

TABLE I
Comparison of RA Algorithms

RA	In 2-D architecture	In block-based architecture
ESP	low repair rate low area overhead	-
MESP	presumably same as ESP	non-optimal repair rate low area overhead
SFCC	optimal repair rate higher area overhead	-
MSFCC	presumably same as SFCC	proposed

Proposed approach is based on SFCC. It uses a group of CAMs for storing fault information. When fault information collection is finished, all faults are classified into three types. Must-repair faults are repaired first using respective available redundant elements. After must-repair faults have been repaired, remaining redundant elements are used to repair sparse faults by utilizing an auxiliary buffer structure. After sparse faults have been repaired, remaining single faults are repaired using remaining redundant elements randomly.

Proposed fault collection structure is based on SFCC and is shown in Fig. 4. Information on must-repair faults is stored in MR/MC CAMs in Fig. 4 (a). Information on faults that are found first in their respective row or column (i.e. pivots or parents) is stored in PA CAMs in Fig. 4 (b). Information on faults that share row or column address with parent faults (named child faults) is stored in CH CAMs in Fig. 4 (c). Maximum number of must-repair CAMs for rows (MR) and columns (MC) is R and C, respectively. Maximum number of parent CAMs is R + C. Maximum number of child CAMs is R.(C – 1) + C.(R – 1). Fault collection is finished when BIST is finished.

In Fig. 4, all enable flags are set to 1 if corresponding CAM is used to store fault information, otherwise it is 0. Block row and block col fields denote the divided block row

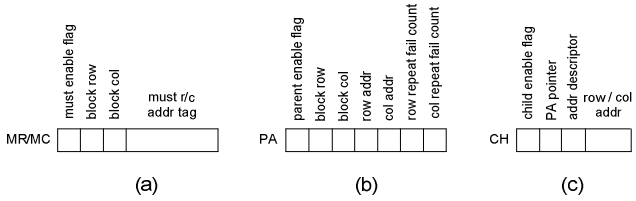

Fig 4. Fault collection structure of MSFCC.

and column address in which the fault is localized, respectively. The idea of how these values are derived is shown in Fig. 5 (b), which depicts the same type of memory as shown in Fig. 3 (d). Must r/c addr tag identifies the row or column address of the must-repair faults stored either in MR or MC CAMs, respectively. Row addr and col addr fields denote the row and column address of parent faults in divided blocks, repsectively. Row repeat fail count and col repeat fail count fields store the number of how many child faults share the same row or column address with their parent faults, respectively. PA pointer points to a respective PA CAM in which the parent fault information is stored. Addr descriptor is set to 0 if the child fault shares the column address with its parent fault, otherwise it is 1. Row / col addr field denotes the row address of child fault if add descriptor is 0, otherwise it denotes the column address.

Proposed approach uses auxiliary buffer structure based on sparse faulty line buffer [2] and is shown in Fig. 5 (a). Maximum number of lines in auxiliary buffer (denoted as L) is 2.(R + C) if R + C is even, otherwise 2.(R + C – 1) + 1. In Fig. 5(a), enable flag, block row and block col fields have the same purpose as in Fig. 4. R/C flag is set to 1 if the sparse faults share the same column address, otherwise it is 0. R/C sparse addr field denotes the row or column address of the sparse faults if R/C flag is 0 and 1, respectively. Line fault count field stores the information on how many sparse faults are sharing the same row or column address if R/C flag is set to 0 and 1, respectively. The contents of a L-bit intersection flag field are set by analyzing the intersected faults (a fault, which shares both row and column address with at least one other sparse fault). The number of buffer line which stores the information about the row and column of the sparse fault by is denoted by i and j, respectively. Then the j-th bit of this field in i-th buffer line and the i-th bit in j-th buffer line are both set to 1.

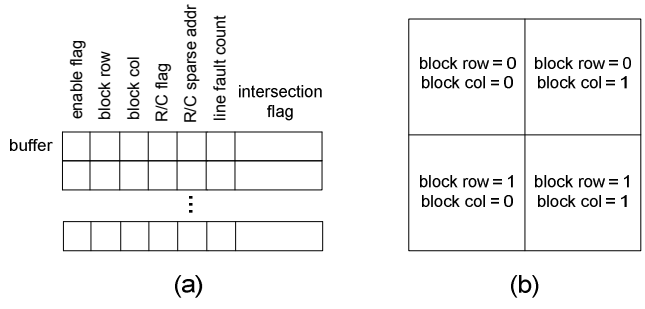

Fig 5. Auxiliary buffer structure of MSFCC.

ISBN: 978-988-19252-1-3
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

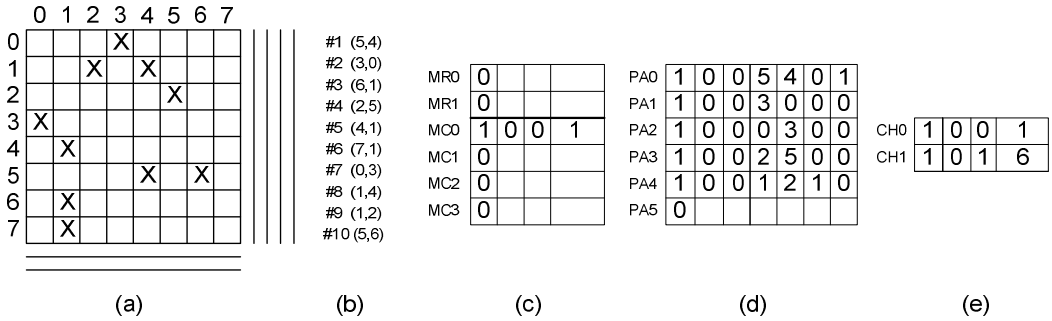

Fig 6. Example of MSFCC algorithm.

Repair solution for sparse faults is derived by analyzing the information in auxiliary buffer. MSFCC searches the solution space by counting the number of faults covered (NFC) for each line combination from auxiliary buffer. For a combination of lines to be the correct repair condition, the value of NFC has to satisfy the following condition [2]:

$$NFC \geq TF - (R + C - S) \qquad (1)$$

where TF is total number of faults remaining in the memory after the must-repair faults have been repaired and S is the number of lines in the selected combination.

III. Example

We show the function of proposed RA algorithm on the example in Fig. 6. Fig. 6 (a) depicts one of four divided memory blocks of the same memory type and with same number of redundant elements (R = 2, C = 4) as in Fig. 3 (d). This block has row and column addresses both equal to 0 (Fig. 5 (b)). Other three blocks are fault free and are not pictured. Total size of the memory in this example is therefore 16x16, but all faults are located only in one of its divided blocks.

The numbers of CAMs needed are as follows: 2x MR, 4x MC, 6x PA and 10x CH CAM. Faults in the memory are detected in the order as shown in Fig. 6 (b). After detection of the last fault, the contents of must-repair, parent and some child CAMs are shown in Fig. 3 (c), (d) and (e), respectively. Column with address 1 (column 1) is repaired first, as it contains three must-repair faults (#3, #5 and #6 in Fig. 6 (b)), therefore the value of C changes from 4 to 3.

There now remains a total of 7 faults (4 sparse and 3 single). To repair sparse faults (#1, #8, #9 and #10), auxiliary buffer structure is introduced. The contents of auxiliary buffer structure after repairing of must-repair faults are shown in Fig. 7. The number of buffer lines needed is 12, but only the first 4 are shown.

buffer							
	1	0	0	1	4	2	0...0110
	1	0	0	0	1	2	0...0001
	1	0	0	0	5	2	0...0001
	0						

Fig 7. Example of usage of auxiliary buffer structure.

In Fig. 7, the first line stores the information about column 4, the second line about row with address 1 (row 1) and the third line about row 5. All of them does have the same number of 2 sparse faults in them. Intersected faults are #1 (row 5, column 4) and #8 (row 1, column 4). Therefore the intersection flags are set to indicate the intersection between first and third buffer line (for fault #1) by setting the third bit in first buffer line and the first bit in the third buffer line to 1 and likewise between the first and the second line (for fault #8).

S	solution	NFC
1	col 4	2
1	row 1	2
1	row 5	2
2	col 4 row 1	3
2	col 4 row 5	3
2	row 1 row 5	4

Fig 8. Determining repair solution in MSFCC.

Now, MSFCC counts the values of NFC for all possible combinations of repair solutions for sparse faults. The results are shown in Fig. 8.

In this example, the values of TF, R and C are 7, 2 and 3, respectively. For a combination to be the correct repair solution, the value of NFC, according to (1) has to be more or equal to 3 for S = 1. In Fig. 8, no solutions for S = 1 satisfy this condition. For S = 2, the value of NFC has to be more or equal to 4. In Fig. 8, only one solution satisfies this condition, so this is selected by MSFCC as the correct repair solution for this example. Sparse faults are repaired using 2 redundant row blocks to replace rows 1 and 5. The value of R is changed from 2 to 0. The value of C remains 3.

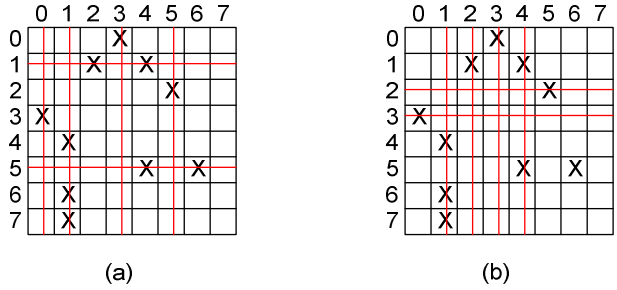

Fig 9. Comparison of MSFCC and MESP algorithms.

After the must-repair and sparse faults have been repaired, MSFCC repairs the remaining single faults (#2, #4 and #7) by remaining three redundant column blocks. The final repair solution by MSFCC is shown in Fig. 9 (a). For

ISBN: 978-988-19252-1-3
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

comparison, the solution for this example found by the MESP algorithm is shown in Fig. 9 (b). As shown in Fig. 9 (b), MESP is not able to find a repair solution for this example as one fault is left un-repaired and marks this memory as un-repairable. This shows the potential of MSFCC to improve the repair rates of previous RA algorithms used in block-based redundancy architectures.

IV. Conclusion

A new block-based redundancy architecture for built-in self-repairing of embedded memories is proposed in this paper. It is based on DWL and DBL techniques and can be used in modern SOC designs to improve manufacturing yield.

The proposed redundancy analysis approach is based on modified SFCC algorithm (MSFCC) which is supposed to have better repair rate for memories than that of previous RA algorithms used in block-based redundancy architectures. Area overhead of the proposed approach is supposed to be higher than that of previous block-based architectures since more complex RA algorithm is used. However, further experiments on this are yet to be performed.

References

[1] International Technology Roadmap for Semiconductors (ITRS), Test and Test Equipment, 2009, Available: http://www.itrs.net/Links/2009ITRS/2009Chapters_2009Tables/2009_Test.pdf

[2] W. Jeong, I. Kang, K. Jin, S. Kang, "A Fast Built-in Redundancy Analysis for Memories With Optimal Repair Rate Using a Line-Based Search Tree", IEEE Transactions on VLSI systems, vol. 17, no. 12, pp. 1665-1678, 2009.

[3] M. Fischerová, E. Gramatová, “Memory Testing and Self-Repair”. R. Ubar, J. Raik, H. T. Vierhaus, *“Design and Test Technology for Dependable Systems-on-Chip*”, Hershey, Pennsylvania: IGI Global, 578 p. ISBN 978-1-60960-212-3, pp. 155-174, 2010.

[4] O. Novák, E. Gramatová, R. Ubar, *"Handbook of Testing Electronic Systems"*, České vysoké učení technické v Praze, 395 p. ISBN 80-01-03318-X, 2005.

[5] S.-K. Lu et al., "Efficient Built-In Redundancy Analysis for Embedded Memories With 2-D Redundancy", IEEE Transactions on VLSI systems, vol. 14, no. 1, pp. 34-42, 2006.

[6] S.-K. Lu, C.-L. Yang et al., "Efficient BISR Techniques for Embedded Memories Considering Cluster Faults", IEEE Transactions on VLSI systems, vol. 18, no. 2, pp. 184-193, 2009.

ISBN: 978-988-19252-1-3
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)