A BIO-INSPIRED TECHNIQUES FOR BUILT-IN SELF TESTING WITH OPTIMAL REPAIR CAPABILITY IN WORD ORIENTED MEMORIES

S.Vithya[*]

V.Gopi[**]

Abstract—

This paper presents a built-in self repair analyzer with the optimal repair rate for memory arrays with redundancy. The proposed method requires only a single test, even in the worst case. The must-repair analysis begins by comparing the number of faulty cells in a row or a column to the number of available redundant columns or rows. If the number of faulty cells in a row or column is greater than the number of available rows or columns, then the line becomes a must-repair line, and a redundant row or column line is used to repair that line. This is the must-repair rule. If the must-repair method cannot find a solution, the information of the faulty cells is collected, and each fault is classified as part of a single-fault group or a multi-fault group. The final-repair algorithm finds all solutions in each fault group and generates an optimal solution. In the case of embedded memories for systems-on-a-chip (SOC), built-in redundancy analysis (BIRA) is widely used as a solution to solve quality and yield issues by replacing faulty cells with extra good cells. However, previous BIRA approaches focused mainly on embedded memories rather than commodity memories. The infrastructure is also extended to support varioustypes of word-oriented memories.

[*] PG Student, Applied Electronics, Department of Electronics and Communication Engineering, PSN College of Engineering and Technology, Melathediyoor

[**] HOD, Department of Electronics and Communication Engineering, PSN College of Engineering and Technology, Melathediyoor

I. INTRODUCTION

As the density of the memory in a device increases, the number of faults also increases. Therefore, to increase the memory yield, many manufacturers use incorporated redundancy to replace faulty cells. In this redundancy technology, the implementation of an effective redundancy algorithm is essential. The failure of embedded memories in a SoC is more ex-pensive than that of commodity memories because a relatively large die is wasted. Due to the large die size and the complex fabrication process for combining memories and logic, SoCs suffer from relatively lower yield, nessitating yield optimization techniques [1]. In accordance with this trend,built-in redundancy allocation(BIRA) approaches have been proposed as part of BISR. In [4], Kawagoe et al. propose a pioneering BIRA approach, CRESTA. They use parallel sub-analyzers, each of which evaluates a solution candidate. In [5], the authors provide a formal basis for design of BIRA algorithms from prime repair algorithms, which correspond to the sub-analyzers. They evaluate particular combinations of the sub-analyzers and show that the area overhead can be reduced with slight degradation of the repair rate. Recently, a memory testing methodology using built-in self-repair (BISR) was suggested [8]-[15]. However, due to the large hardware overhead associated with BISR, a lot of memory is fabricated. For this reason, ATE with RA algorithms is extensively employed to test and repair memory. The simultaneous testing and repair of a lot of memory for mass production requires a significant amount of time. Therefore, the analysis speed is very important in RA algorithms for ATE, and manufacturers should minimize the time for testing/repairing.In this paper, the very efficient RA (VERA) algorithm, which has a 100% normalized repair rate, is proposed. In the analysis, a bitmap that grouped addresses is utilized. Because the fault groups are independent of each other, the time needed to find solutions can be greatly reduced. Therefore, the proposed algorithm determines an optimal solution for repairing memory faults in a short amount of time. It is also the fastest and easiest algorithm to develop or implement in ATE tests.

II. PRELIMINARIES

In the classical spare allocation problem, we consider a bit-oriented memory array with spare (repair) rows and spare (repair) columns. Any fault row (column) can be replaced with a spare row (column). A repair solution is a set of at most row addresses and at most column addresses that cover all faults (all faults are on the addresses). If a repair solution exists for a memory array, the memory array is repairable. A repair strategy is a string of the alphabet such that occurs times and occurs times. Thus there are repair strategies .For example, if and the set of all possible repair strategies. This is a contradiction. Therefore, for at least a repair row or column, the must repair condition is satisfied.

Corollary 1: If the number of faults on a memory array is greater than and the must-repair conditions for all repair rows and columns are not satisfied, the memory array is not repairable [12].

Corollary 2: If a memory array is repairable, the number of faults captured in the (unbounded) fault-list is at most Proof: Let and be the number of must-repair rows and columns in the memory array, respectively. All faults which are neither on the must-repair rows nor columns should be covered by at most repair rows and at most repair columns.

The number of the faults are at most by the same argument as Lemma 1. The fault-list has at most faults for the must-repair rows and columns. There are various types of word-oriented repairable memories, and they impose different constraints on the spare allocation problem. Since it is difficult to capture all the different types of repairable memories into a generalized model and to design an universal repair analyzer, we categorize them into three types, which will be called type A, type B, and type C, respectively. Typically, a faulty row is replaced with a spare row, but the way to replace a faulty column varies, based on which they are classified. Fig. 3 illustrates the column circuitry of a word-oriented memory of type A. In word-oriented memories, the data in a word is usually not placed in adjacent locations due to several issues such as the coupling effect, and the columns associated with the same bit position are clustered together.

III. EXISTING METHOD

The failure of embedded memories in a soc more expensive than that of commodity memories because large die is wasted. In addition aggressive design rules make the memory array prone to defect. BIST was used to defects the faulty memory. In accordance with this trend, built in redundancy allocation (BIRA) approaches have been proposed as part of BISR. Performance was very low. More time conceptive work. Can't self repair capability. Kuo and Fuchs proposed the branch-and-bound algorithm, which is a heuristic method based on a binary search tree [3]. Although the branch-and-bound algorithm [2] is faster than the exhaustive binary search algorithm, it does not significantly reduce the time or memory space needed to search for a solution. As such, various early-termination methods for reducing the analysis time have been proposed [3]-[5]. However, early-termination methods for reducing the analysis time cannot be considered to be absolute solutions in cases of large amounts of memory.

IV. PROPOSED INFRASTRUCTURE

In this section, we propose an on-chip infrastructure for bit-oriented memories. This infrastructure will be extended for word-oriented memories later. Our repair analyzer requires only a single test and provides the optimal repair rate. Our infrastructure does not depend on BIST engines, and we assume

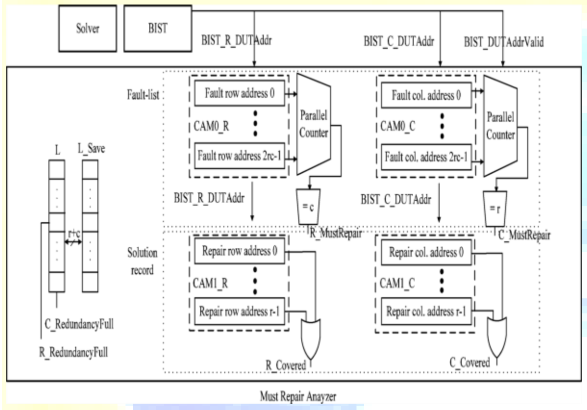
that an arbitrary BIST engine tests a memory array and provides fault addresses whenever detected. Our infrastructure adopts the framework of the Kuo-Fuchs algorithm [11] and consists of must-repair analsysis and final analysis. The must-repair analysis identifies must-repair rows and columns, and the final analysis searches a repair solution. The must-repair analysis is performed concurrently with the test, while the final analysis is done after the test is completed. The must repair analyzer (MRA) is shown in Fig. 2. The MRA consists of a pair of CAMs for fault addresses, called the fault-list, and a pair of CAMs for a repair solution, called the solution record. In the first step, our approach is similar to that of previously proposed must-repair analysis [2], [3]. The must-repair analysis begins by comparing the number of faulty cells in a row or a column to the number of available redundant columns or rows. If the number of faulty cells in a row or column is greater than the number of available rows or columns, then the line becomes a must-repair line, and a redundant row or column line is used to repair that line. This is the must-repair rule. If the must-repair method cannot find a solution, the information of the faulty cells is collected, and each fault is classified as part of a single-fault group or a multi-fault group.

For the next step, the number of groups and remaining redundant cells are analyzed to see if they satisfy Property 3. If the number of fault groups is greater than the number of redundant cells (RS + CS), the faulty memory cannot be repaired. Therefore, the use of this early-termination solution allows irreparable memory to be terminated prior to the performance of an RA algorithm. If Property 3 is not satisfied, the faulty memory can be repaired, and the next step, which is performing the FAST algorithm to determine the solution in the shortest amount of time, is conducted. To improve the yield, memory arrays are usually equipped with spare elements. External testers have been used to test the memory arrays and configure the spare elements. Built in self repair techniques for reducing the test time In accordance with this trend, built in redundancy allocation (BIRA) approaches have been proposed as part of BISR. Design of BIRA algorithms from prime repair algorithms, which correspond to the sub analyzers. Such serial implementations may increase the overall test time, but the number of possible solutions is reduced using the must-repair analysis. In array memory some bits are not considered while processing. In future, those bits are also considered for increasing the repairing capability.

Algorithm Final-Repair

```
/* ordering */
begin queue = fault groups;
while (fault groups left and the queue not empty)
 {
 i = # of faults in maximum group from the queue;
 j = # of faults in first group from the queue; group tmp = group j;
 group j = group i;
 group i = group tmp
; remove the first group from the queue;
 }
 end
/*Finding an optimal solution */
 begin queue1= fault groups;
 while (fault groups left and the queue1 not empty) {
 queue2 = solutions;
 While (solutions left and the queue2 not empty) {
 if (RS >= R + # of used spare row of this queue2)
R = R + # of used spare row of this queue2;
 If (CS >= C + # of used spare column of this queue2)
 C = C + # of used spare column of this queue2;
 Combined solutions are stored;
 remove the first record from the queue2;
 }
 Remove the first record from the queue1;
}
 if (the queue1 is empty and RS >= R and CS >= C)
 an optimal solution is selected ;
else fail;
end
```

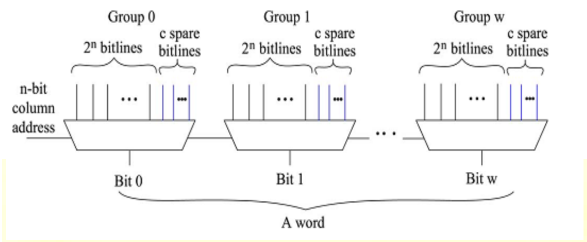
Proposed on-chip infrastructure



V. EXTENSION FOR WORD-ORIENTED-MEMORIES

In this section, we will extend the proposed infrastructure for word-oriented memories, which is more common than bit-oriented memories in practice. Unlike the bit-oriented memory case, from the BIST engine, our infrastructure takes as input a triplet(Rx,Cx)where (Rx;Cx);is the row(column)address, and is the failure syndrome, which is the exclusive OR of the test response and the expected output of the word at(Rx,Cx) . For word-oriented memories of type A, we can discard the failure syndrome and can input only the row and column addresses to the proposed infrastructure. Then without any modification, it will perform repair analysis for type A word-oriented memories. For a word-oriented memory of type B, our repair analyzer is modified as follows. Let be the word size of the device under test (DUT). We will map the word-oriented memory to a bit-oriented memory. Since every bit should be addressable in the bit-oriented memory, we expand the width of the column address by to distinguish each column within a word .We call the extended address the virtual column address. In this case, a triplet can generate up to virtual column address for the bit-oriented memory. However, in the case that the number of “1”s in is greater than 1, it is obvious that the row being tested is a must-repair row since the DUT is 1 column-per-word replaceable.

A word oriented memory



VI. EXPERIMENTAL RESULTS

We implemented the proposed infrastructure in 130-nm technology for a memory array with four repair rows and four re-pair columns, and the operating frequency is 400MHz.We custom-designed CAMs and synthesized the other logics using Synopsys Design Compiler except an 8-bit subset enumerator. Major evaluation factors of BIRA performance include analysis time, area, and repair rate. Table VI compares our method to CRESTA and the intelligent Solve First proposed in [6]. Since all these methods provide the optimal repair rate, the repair rates are not presented. The number of test and the number of CAM entries dominate the analysis time and the area, respectively. Thus, the test time in the worst case and the area can be estimated using Table VI. As mentioned earlier, CRESTA performs repair analysis in parallel with the test and evaluate all solution candidates simultaneously using the multiple sub-analyzers, requiring only one test irrespective of the number of repair elements. The test and repair analysis finish at the same time, and one of the sub-analyzers contains the optimal solution. Since no extra cycle after the test is required, the analysis time equals the test time. In the case that or as in the third row in Table VI, the number of possible solution candidates increases linearly in the number of repair elements, and the spare allocation problem becomes relatively easy. However, it is known that the repair rate of and is worse than that of and [4]. The methods proposed in [6] reduce the number of required CAM entries at the cost of the analysis time. The Basic Solve in [6] performs the exhaustive search and requires tests (or, restarts) if the optimal solution is necessary in terms of the number of repair elements used. If the number of repair elements used does not matter, it may find a solution with a few tests [6] but the worst case bound is still The required number of tests by the intelligent Solve and the intelligent Solver First in the worst case seems to be much less in simulation, but it is not proven theoretically. In our proposed method, the restart of the test does not happen in any case. This comes at the cost of a few extra cycles after the single test for the final analysis.

BIST

Memory Location

00000000

00000001

00000010

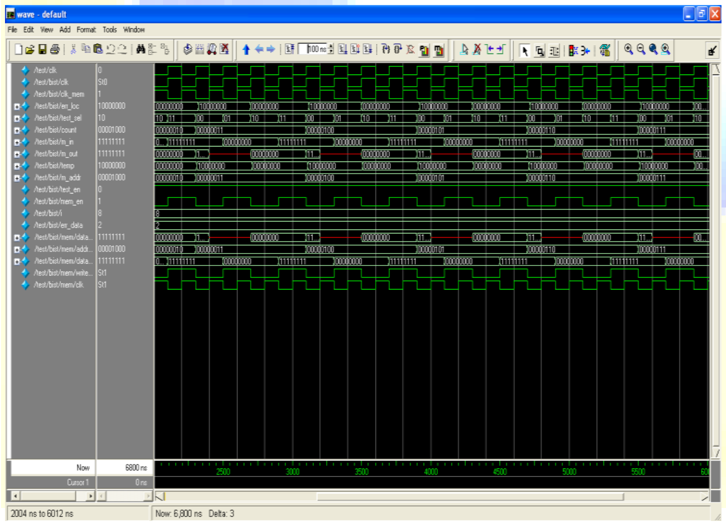
00000011

00000100

00000101

00000110

00000111



```
/test/clk                0
/test/bist/clk           St0
/test/bist/clk_mem       1
/test/bist/err_loc       10000000
/test/bist/test_sel      10
/test/bist/count         00001000
/test/bist/m_in          11111111
/test/bist/m_out         11111111
/test/bist/temp          10000000
/test/bist/m_addr        00001000
/test/bist/test_en       0
/test/bist/mem_en        1
/test/bist/i             8
/test/bist/err_data      2
/test/bist/mem/data...   11111111
/test/bist/mem/addr...   00001000
/test/bist/mem/data...   11111111
/test/bist/mem/write...  St1
/test/bist/mem/clk       St1

Now          6800 ns
Cursor 1     0 ns
2004 ns to 6012 ns      Now: 6,800 ns  Delta: 3
```

Error Location

10000000

10000000

10000000

10000000

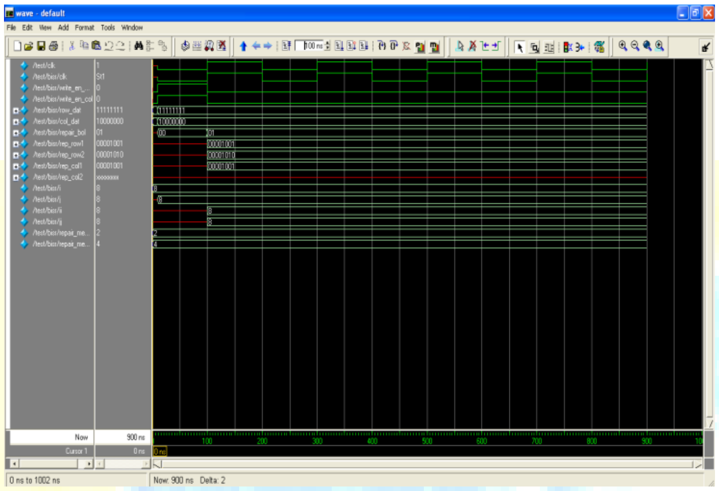
10000000

10000000

10000000

10000000

BISR



```
/test/clk                 1
/test/bisr/clk            St1
/test/bisr/write_en_...   0
/test/bisr/write_en_col   0
/test/bisr/row_dat        11111111
/test/bisr/col_dat        10000000
/test/bisr/repair_bol     01
/test/bisr/rep_row1       00001001
/test/bisr/rep_row2       00001010
/test/bisr/rep_col1       00001001
/test/bisr/rep_col2       xxxxxxxx
/test/bisr/i              8
/test/bisr/j              8
/test/bisr/ii             8
/test/bisr/jj             8
/test/bisr/repair_me...   2
/test/bisr/repair_me...   4

Now        900 ns
Cursor 1   0 ns
0 ns to 1002 ns      Now: 900 ns  Delta: 2
```

Row Correction

00001001

00001010

Column Correction

00000001

VII. CONCLUSION

In this paper, we have proposed an on-chip infrastructure for repair analysis with the optimal repair rate. Our infrastructure requires a single test and a few extra cycles, which is about 600 cycles in a memory array with four repair rows and four re-pair columns. Most built-in repair analyzers are developed for bit-oriented memories, whereas our repair analyzer also aim sat various types of word-oriented memories. To achieve this, we have extensively studied existing word-oriented repairable memories and have classified them into three types. For each type, we have showed how the bit-oriented version can be extended. As part of our repair analyzer, we have also developed a novel combinatorial circuit for enumerating constant-weight vectors.

REFERENCES

[1] R. Rajsuman, "Design and test of large embedded memories: An overview," IEEE Design Test Comput., vol. 18, no. 3, pp. 16-23, May2001.

[2] S. Hamdioui, G. Gaydadjiev, and A. van de Goor, "The state-of-art and future trends in testing embedded memories," in Proc. Records Int.WorkshopMemoryTechnol.,Design,Test.,2004,pp.54-59.

[3]Y. Zorian and S. Shoukourian, "Embedded-memory test and repair:Infrastructure IP for SOC yield," IEEE Design Test Comput., vol. 20,no. 3, pp. 58-66, May/Jun. 20034]T. Kawagoe, J. Ohtani, M. Niiro, and T. Ooishi, "A built-in self-repair analyzer (cresta) for embedded drams," in Proc. Int. Test Conf., 2000,pp. 567-574.

[5]S. Shoukourian, V. A. Vardanian, and Y. Zorian, "A methodology for design and evaluation of redundancy allocation algorithms," in Proc.VLSI Test Symp., 2004, pp. 249-255.

[6]P. Oehler, S. Hellebrand, and H.-H. Wunderlich, "An integrated built-in test and repair approach for memories with 2D redundancy,"in Proc. Eur. Test Symp., 2007, pp. 91-96.

[7]P. Oehler, S. Hellebrand, and H.-J. Wunderlich, "Analyzing test and repair times for 2D integrated memory built-in test and repair," inProc. Design Diag. Electron. Circuits Syst., 2007, pp. 1-6.

[8]P. Oehler, A. Bosio, G. D. Natale, and S. Hellebrand, "A modular memory BIST for optimized memory repair," in Proc. Int. On-Line Test. Symp., 2008, pp. 171-172.

[9]W. Jeong, I. Kang, K. Jin, and S. Kang, "A fast built-in redundancy analysis for memories with optimal repair rate using a line-based search tree," IEEE Trans. Very Large Scale Integr. (VLSI) Syst., vol. 17, no.12, pp. 1665-1678, Dec. 2009.

[10] W. Jeong, J. Lee, T. Han, K. Lee, and S. Kang, "An advanced BIRA for memories with an optimal repair rate and fast analysis speed by using a branch analyzer," IEEE Trans. Comput.-Aided Design Integr. CircuitsSyst., vol. 29, no. 12, pp. 2014-2026, Dec. 2010.

[11] S.-Y. Kuo and W. Fuchs, "Efficient spare allocation for reconfigurable arrays," IEEE Design Test Comput., vol. 4, no. 1, pp. 24-31, Feb. 1987. [12] C.-T. Huang, C.-F. Wu, J.-F. Li, and C.-W. Wu"Built-in redundancy analysis for memory yield improvement," IEEE Trans. Reliab., vol. 52, no. 4, pp. 386-399, Dec. 2003.

[13] A. Sehgal, A. Dubey, E. Marinissen, C. Wouters, H. Vranken, and K. Chakrabarty, "Redundancy modelling and array yield analysis for repairable embedded memories," IEE Proc. Comput. Digit. Techn., vol.152, no. 1, pp. 97-106, 2005