International Journal of Advanced Research in

Electrical and Electronics Engineering

Volume: 1 Issue: 2 08-Nov-2013,ISSN_NO: 2321-4775

BIRA for Word Oriented Memories Using Parallel Prefix Algorithm.

C. Bhaskar[1], K. Jeyaprakasam[2]

Assistant Professor, Department of Electronics and Communication Engineering[1], Assistant Professor, Department of Bio Medical Engineering[2], Odaiyappa College of Engineering and Technology, Theni, Tamil Nadu, India[1,2].

bhaskarmemba@gmail.com[1]

kannamanilakshmy.swamysaranam@gmail.com[2]

Abstract- This paper presents a built-in self repair analyzer(BIRA) with the optimal repair rate for ingmemory arrays with redundancy. The proposed method requires only a single test, even in the worst case. By performing the must-repair analysis on the fly during the test, it selectively stores fault addresses, and the final analysis to find a solution is performed on the stored fault addresses. To enumerate all possible solutions, existing techniques use depth first search using a stack and a finite-state machine. Instead, we propose a new algorithm and its combinational circuit implementation. Since our formulation for the circuit allows us to use the parallel prefix algorithm, it can be configured in various ways to meet area and test time requirements. The total area of our infrastructure is dominated by the number of content addressable memory entries to store the fault addresses, and it only grows quadratically with respect to the number of repair elements. The infrastructure is also extended to support various types of word-oriented memories by using pre computation CAM.

Keywords: **Built-in self repair (BISR), memory test, Built in repair analyzer(BIRA).**

1.Introduction:

Today's system-on-chip (SoC) environment requires significant changes in testing methodologies for memory arrays. The failure of embedded memories in a SoC is more expensive 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 the relatively lower yield, if Necessitating yield optimization techniques at present, the area occupied by the embedded memories takes more than half of the total area of a typical SoC, and the ratio is expected to keep increasing in the future. The defects are thus likely to affect the functionality of the memory arrays rather than that of logic. In addition, the aggressive design rules make the memory arrays prone to defects. Therefore, if the overall SoC memory yield is dominated by the other memory yield optimizing

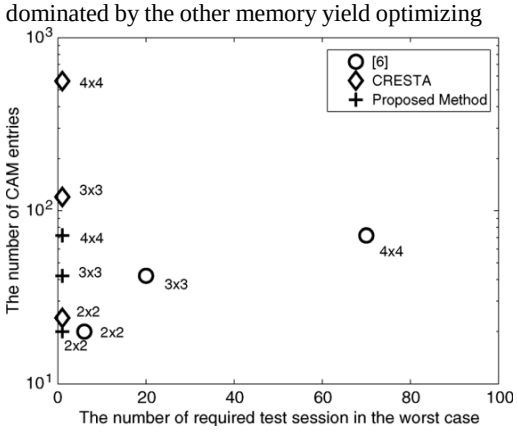

Fig. 1. Required number of CAM entries and worst-case test sessions of each analyzer for several redundancy configurations

the memory yield plays a crucial role in the SoC environment. To improve the yield, memory arrays are usually equipped with spare elements, and external testers have been used to test the memory arrays and configure the spare elements. However, in area for on-chip test infrastructure at lower cost than before, which makes feasible a variety of built-in self test (BIST) and built-in self-repair (BISR) techniques for reducing the test time. For existing optimal analyzers as well as our analyzer, Fig. 1 shows the number of test sessions and CAM entries required for the repair analysis.. Our infrastructure provides the optimal repair rate with a single test as in CRESTA and has the same requirements for the number of CAM entries,Instead of a stack and a finite- state machine (FSM) used to enumerate all possible solutions, we propose a combinational circuit, which can be configured in various ways to meet the requirements for area and test time. For the fastest configuration, it can generate the next solution candidate in a single cycle. Unlike most repair analysis studies, we show that the proposed method can work for word-oriented memories.

2. 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

Volume: 1 Issue: 2 08-Nov-2013,ISSN_NO: 2321-4775

tests a memory array and provides fault addresses whenever detected. 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 fault-list, each CAM has one extra valid bit for each word, and the valid bits are initialized to "0" in the beginning. Since the CAMs assert "1" at the valid bit position for write and match operation, only written entries can be matched. During the test, if the BIST engine detects a fault, it sends the fault address to the MRA on the fly through *BIST_R_DUTAddra*nd *BIST_C_DUTAddr*, and continues the test. The row (column)fault address is compared against row (column) CAM entries, and the number of matched entries is efficiently counted by a parallel counter[1] . If the number of the matched entries equals in the row (column) CAM, the row (column) indicated by the fault address satisfies the must-repair condition and *R_MustRepair (C_MustRepair*) signal is asserted. If the fault address triggers neither the row nor column must-repair condition, MRA writes the row and column address in the row and column CAMs, respectively. if the overflow of the fault-list occurs, the memory array can be determined as un repairable, and the test can be terminated early. If a particular row or column is identified as must-repair, the row or column address must be part of the solution. Thus the MRA writes the row or column address in the solution record. The L registers are used as valid bits for the solution record and also determine the next available CAM entry. Since a must-repair row and a must-repair column can be identified by a fault at the same time, the MRA should be able to write a pair of row and column addresses simultaneously. Once a row or column address is stored as part of solution by the must-repair condition, then all solution candidates considered by the SOLVER include the address, and faults on the address do not affect the final analysis any more. Therefore, such faults do not need to be stored, and we can collect all necessary information for the final analysis during a single test. Once the test is completed (thus the must-repair analysis is done), *BIST_Done* signal is asserted and the final analysis is started. The operation of the SOLVER and the MRA in the final analysis phase is illustrated in Fig.3. In the final analysis, The SOLVER will generate repair strategies one by one and will check whether each repair strategy can fix all the faults captured in the fault-list. If "r " and "c"are mapped to"1" and "0", respectively, then a repair strategy can be represented by a-bit word. The Repair Strategy module comprises a-bit register and stores the repair strategy being tested currently. The first repair strategy is

Volume: 1 Issue: 2 08-Nov-2013,ISSN_NO: 2321-4775

generated depending on the numbers of must-repair rows and columns, or Used Must Repair Rows and Used Must Repair Cols. If one repair row and one repair column are used as must-repair, only the two repair strategies "r " and "c" should be generated After a repair strategy is tested, the state of the MRA should be reverted to that right after the must-repair analysis so the values in the L registers are copied to the L_save registers before the final analysis begins. The SOLVER generates the first repair strategy and the MRA reads each fault address in the fault-list in order until there exists no more fault address or the *RESTART* signal is arrived. The MRA check if each fault is covered by the current solution, stored in the solution record, and asserts *R_Covered* or *C_Covered*. If both signals are low, the fault should be covered by a new repair row or column. The SOLVER determines whether a repair row or column is used for the uncovered fault, and asserts *R_Insert* or *C_Insert*. If *R_Insert* (*C_Insert*) is high, the fault row (column) address is written in the row (column)CAM of the solution record. If the CAM is full, the memory array cannot be repaired by the first repair strategy, and the SOLVER generates the next repair strategy and asserts the *RESTART*signal. When the *RESTART* signal becomes high, the MRA restores the initial state, and the next repair strategy starts being evaluated. In this way, the SOLVER explores the solution space and can find a optimal solution, for the cost function. In our implementation, the cost is defined as the number of used spare elements. The SOLVER has a register to store the cost of the current repair strategy, or Used Repair Elements. The SOLVER also has registers to store the repair strategy with minimum cost so far and the minimum cost, or Repair Strategy Opt and Used Repair ElOpt. The current cost is compared against the minimum cost so far, which generates the Better signal. If the Better signal goes down during the evaluation of the current repair strategy, the SOLVER immediately asserts the *RESTART* signal and moves on to the next repair strategy. If the Better signal stays at "1" until the end of the evaluation, the SOLVER saves the current repair strategy and its cost. Since the SOLVER continues to search for a better solution even after finding a solution, the MRA may not have the optimal solution after the last repair strategy is evaluated. To reduce area, we have stored the optimal repair strategy instead of the optimal solution. If the solution is directly stored, the size of the solution record should be doubled.

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. In type A memories, there are spare column-groups of

columns each. A group of columns associated with a word is replaced with a space-column group. In other words, the column replacement is performed on a column group basis. For example, if the first bit line in group 0 is faulty, and it is replaced.

The first spare column in group 0, then the first bit lines in the other groups are also replaced with the associated first spare columns, respectively. The spare allocation problem in this type can be reduced to the conventional spare allocation problem for bit-oriented memories. Most repair analyzers in the literature are developed for bit-oriented memories. In type B memories, a faulty column is replaced with a spare column, but among a group of columns associated with a word, only one column can be replaced. Where the restriction comes from. A word-

3. 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 (R ,C , S), where 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[2] .

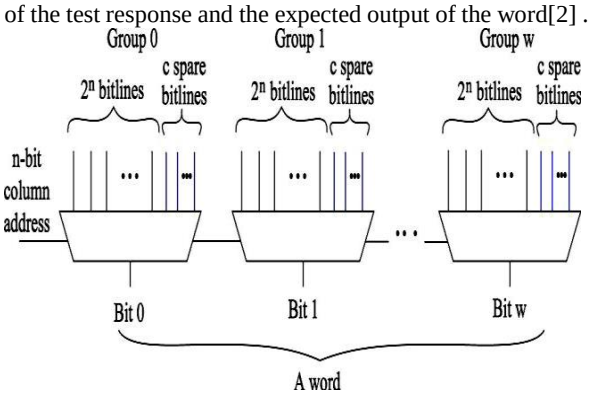

Fig 2. Column circuitry of a word-oriented memory of type A.

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.

Type B MEMORY

For a word-oriented memory of type B, our repair analyzer is modified as follows. Let be the word size of the

oriented memory of type B has only spare columns un like that of type A. Each spare column is selected when a programmed column address is accessed. Up to faulty columns can be replaced, but columns that constitute a word cannot be replaced together. An efficient implementation of a word-oriented memory of this type is proposed. memory of type C, where any faulty column can be replaced with an available spare column without any restriction. Various implementations In order to generalize the constraint that arises in type B, wide fine a new term. In a memory, if up to columns out of those
associated with a word can be replaced with spare columns, the memory is *column-per-word replaceable*. Thus, memories of type B are 1 column-per-word replaceable.ig.3.
Solver details and MRA operation in the final analysis phase
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. Thus, if this case is handled separately, one triplet will generate only one virtual column address. The pair of the incoming row address and the virtual column address is fed into the proposed infrastructure, which will work with the word- oriented memory of type B.

Let us extend this scheme for a column-per-word replaceable memory where . In this memory, a triplet can generate virtual column addresses. It is common to perform memory BIST at-speed for higher test quality, which means that our infrastructure may receive a triplet at every cycle so the virtual column addresses may need to be handled in one cycle[3]. They can be placed in pipeline, but this does not prevent the BIST from being stopped.

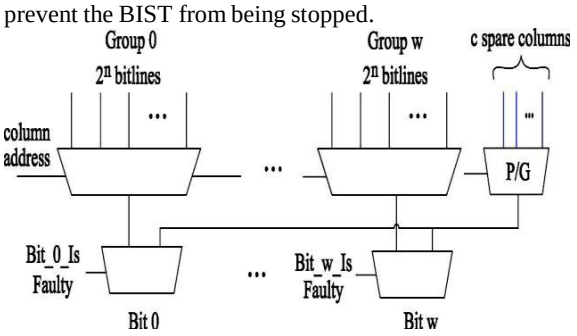

Fig. 3. Column circuitry of a word-oriented memory of type B.

Thus, in order to enable at-speed BIST together with BISR, it is necessary to handle this memory in a different way

from that of type B. Note that -bit-word-oriented memories of type C are column-per-word replaceable. If we can deal with type C, any column-per-word replaceable memory with can be also handled easily.

Type C MEMORY

We modify the MRA to support word-oriented memories of type C. To begin with, we define several terms. In word-orient memories, bits (columns) have the same address. However, in order to repair such memories on a column basis, we need to distinguish each column anyway, so we define the *extended column address* as a pair of a column address and a word of bits, each of which corresponds to one among the columns indicated by the column address [4]. The extended column address can indicate multiple columns within a word. Also, we call a pair of a row address and an extended column address the *extended fault address*.Multiple extended fault addresses
dicating

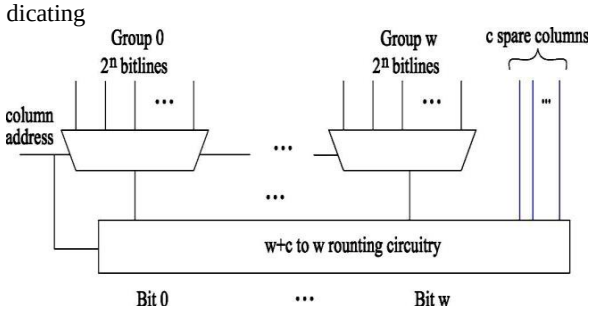

Fig.4. Column circuitry of a word-oriented memory of type C.

4. Experimental Results

We implemented the proposed infrastructure in 130-nm technology for a memory array with four repair rows and four repair columns, and the operating frequency is 400 MHz. We custom-designed CAMs and synthesized the other logics using Synopsys Design Compiler except an 8-bit -subset enumerator fault within a word can be combined into a single extended fault address. Major evaluation factors of BIRA performance include analysis time, area, and repair rate. 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. 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. the number of possible solution candidates increases linearly in the number of repair elements, and the spare allocation problem becomes relatively easy[5]. The method reduce the number of required CAM entries at the cost of the analysis time. The *Basic Solve* in 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 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.

	CRESTA [4]		[6]		Proposed Method						
							CAM size				
							Fault-list		Solution record		
r / c	# test	CAM size	# test	CAM size	# test	Extra time (cycles)	Row	Column	Row	Column	Total
2/2	1	24	6	20	1	34	8	8	2	2	20
3/3	1	120	20	42	1	146	18	18	3	3	42
4/1	1	25	5	21	1	35	8	8	4	1	21
4/4	1	560	70	72	1	638	32	32	4	4	72
5/5	1	2520	252	110	1	2782	50	50	5	5	110

Table 1 Numbers of test sessions in the worst case and the numbers of cam entries

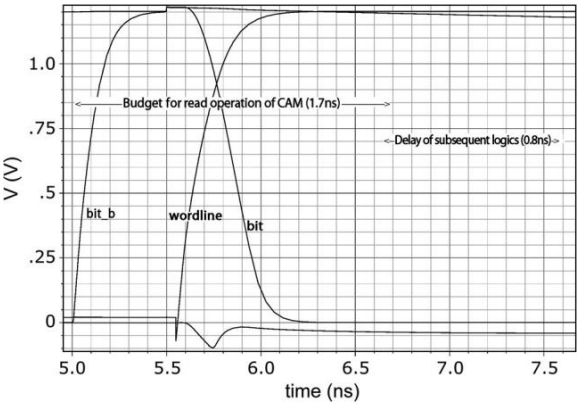

Fig. 7 Read operation of the row CAM in the fault-list.

Our implementation of the proposed method uses 94 flip flops and the number of flip-flops grows at most linearly with respect to the number of repair elements. Also, it uses about 800 combinational cells including an 8-bit -subset enumerator with Kogge-Stone style configuration, The -subset enumerator has much slack in timing, so if it is replaced by a slower configuration, we can implement it using a smaller number of cells. As mentioned earlier, the number of cycles to generate repair strategies varies in the stack-based implementation unlike the -subset enumerator, which could lengthen the final analysis time. However, this may be marginal compared to the long test time [6]. Rather, the -subset enumerator is beneficial for its simplicity as well as smaller area. Due to the varying generation time, the stack-based implementation complicates control logics, while our proposed enumerator does not require them and it is easy to verify once implemented.

5. 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 repair columns. Most built-in repair analyzers are developed for bit-oriented memories, whereas our repair analyzer also aims at 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, May 2001.*

[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. Workshop Memory Technol., 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. 2003.*

[4] 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.*

Biography

C.Bhaskar, Assistant professor in ECE Department in Odaiyappa College of Engineering and Technology, Theni, Tamil Nadu, India. He has more than 09 years of teaching experience with expertise in VLSI. He has completed post graduate in VLSI Design.

K.Jeyaprakasam, Assistant professor in Department of Bio Medical Engineering in Odaiyappa college of Engineering and Technology. Theni, Tamil Nadu, India. He has more than 09 years of teaching experience.