BUILT IN SELF REPAIR FOR EMBEDDED HIGH DENSITY SRAM

Ilyoung Kim, Yervant Zorian*, Goh Komoriya, Hai Pham,
Frank P. Higgins, Jim L. Lewandowski

Bell Laboratories
Lucent Technologies

* currently at Logic Vision, Inc.

Abstract

As the density of embedded memory increases, manufacturing yields of integrated circuits can reach unacceptable limits. Normal memory testing operations require Built-In Self Test (BIST) to effectively deal with problems such as limited access and "at speed" testing. In this paper we describe a novel methodology that extends the BIST concept to diagnosis and repair utilizing redundant components. We describe an application using redundant columns and accompanying algorithms. It allows for the autonomous repair of defective circuitry without external stimulus (e.g. laser repair). The method has been implemented with negligible timing penalties and reasonable area overhead.

1. Introduction

Continual advances in the manufacturing process of integrated circuits provide designers the ability to create more complex and dense architectures and higher functionality on a chip. But these manufacturing process advances are not without limitations. In particular, embedded high density memories in combination with these process limitations can result in poor overall yields. Depending on the application and design, much of the low yield can be attributed to faults in the memory. In order to circumvent these problems, designers have added redundancy to their circuits.

This paper discusses an approach for reconfiguring defective memory cells using redundant columns. Redundancy and diagnosis in memory designs are not new concepts[1][2][3][4][5]. Many DRAM manufacturers use redundancy to *repair* defective cells by replacement with good cells using lasers. These techniques have also been applied to cache memory in high performance microprocessors. Niggemeyer, *et al*,[6] describe a self-reconfiguration technique for high density DRAM's in a video processor design. Our paper also discusses a technique that can be used for self-repair, in particular, a memory device that incorporates surrounding logic to detect faults and repair them by column replacement. It differs from laser repair in that it is a *soft* repair.

The additional circuitry of our Built In Self Repair (BISR) architecture not only detects defective memory cells, but also maps redundant memory cells, via multiplexors, to functionally replace these defective cells. Care is taken in the design of the circuitry to ensure that the use of replacement devices does not adversely affect memory performance. The architecture is designed so that defects in the replacement devices are treated uniformly with defects in the SRAM array, utilizing the same hardware for both, thus reducing area overhead. The test and repair operation is generally performed at power-up to also accommodate degradation of the memory over time.

There are several problems that must be addressed to implement this technique. Section 2 discusses the main memory diagnostic algorithm. The memory architecture is described in Section 3. Section 4 discusses the Spare Allocation Algorithm implemented in the design.

2. Fault Location/Identification

To ensure the accurate determination of fault locations without aliasing, in this section we will list the adopted fault model and test algorithm, and analyze the ability to diagnose the detected faults for repair purposes.

0-7803-5092-8/98 $10.00

2.1 The fault model

The memory test algorithm is based on the 17N Algorithm B[7]. We have expanded it to a 20N algorithm incorporating retention tests, described elsewhere[8][9]:

↓(w0)
↓(r0,w1,w0,w1)
↓(r1,w0,r0,w1)
↑(r1,w0,w1,w0)
↑(r0,w1,r1,w0)
Hold
↑(r0,w1)
Hold
↑(r1)

This march test has been successfully used to detect faults. However, in this application we must verify that it can also be used for diagnosis of the memory.

The fault model for the memory in use is the following:

- Faults in the cell array
  - — Stuck-at faults (SAF) of memory elements
  - — Transition faults (TF)
  - — Coupling faults (CF)
  - — Linked coupling faults
  - — Linked transition and coupling faults
  - — Retention faults
- Faults in the address decoders
  - — Multiple addresses are accessed
  - — The appropriate address is failed to be accessed
  - — A different address is accessed
- Faults in the read-write logic
  - — Stuck-at faults in input/output and read-write register
- Faults in the additional BIST circuitry
  - — Stuck-at faults in the test pattern generator and output data evaluator blocks.

Not in the fault model, but detected (if harmful) is an address fault together with a CF or TF or SAF. The fault that is not detected is a <↑ or ↓;0> CF in case of an OR function for two selected address lines, or a <↑;1> <↓;1> CF in case of an AND function (also analogous TFs and SAF are not detected, but unharmful).

For the RAM to be properly repaired, it is necessary that the BIST fault flag be monitored continuously and that at the moment the fault flag goes high, the current address and bit are known.

The faults that the test does not always detect, are coupling faults between cells of a (output) word. This has consequences for the walking 1/0 test applied after the 20N test, to be sure the read and write registers are free of coupling faults. For this test a memory word has to be selected. However it is not guaranteed to be free of coupling faults between bits in a word. Thus a fault signal indicates a fault in the read-write logic or in the memory cell. For the sake of yield it would be better to repeat the test with another memory word and declare the chip unrepairable only if the second test has at least one fault pattern that coincides with the first one.

2.2 Overview of fault detection and its usefulness for repairing

All address faults (AFs) are detected and mapped onto the memory array such that AFs are repaired properly (in some cases address lines which work correctly in absence of other AFs will be replaced; see comment later on). The address when the fault flag goes up is the address to be replaced.

All faults in the memory array are detected. The current address of the test as indicator is sufficient to repair the fault.

Faults in the read-write circuitry will cause a burst of errors in the memory cells. The chip will be rejected automatically if it can not replace bits inside the read and write registers.

Also faults in the BIST circuit will cause a burst of errors. This fault also cannot be repaired.

So for all faults of the model the current address pointer indicates the replaced address. Unfortunately a faulty flag does not necessarily mean that the current address, *Ax*, is wrong. For example, a write access to *Ay* which changes the contents of *Ax* will cause a fault flag at *Ax* (fault C). Also fault *Ay* will be detected and repaired, leaving a usable *Ax*. Still *Ax* will be repaired, despite its solved problem.

Another issue is that an addressing fault (multiple cell access) with a stuck-open fault (fault B)

will only be partially repaired; *i.e.*, the stuck-open fault is detected and the cell is replaced, but the address accessing two or more memory cells will not be replaced. In general this will not be a problem because a cell is first written before it is read.

The same problem arises for faults D2 and D3[10] in an AND-implementation. If two or more addresses are accessed, fault D3 will not be detected for address *Ax*. In the OR implementation fault D2 will not be detected for address *Ax*. In both cases the faulty row will not be replaced.

A similar problem is chained CFs (the coupled cell is a coupling cell too). Also for them it is not necessary that all faulty cells are replaced. Only the cells between the head and the tail of the chain need to be replaced. This will not be detected by the march test.

Faults in the read and write circuitry can not be distinguished from each other. This means that either the two registers have to be tested separately, or if a fault is detected in one of them, in both the same bit replaced. SAF in these circuitries can be recognized by the observation that a certain bit position for all words is SA. If this fault will be repaired then the same test has to be done all over again, because of the risk of masking other faults.

Based on the 17N algorithm, all faults in the memory array are detected. The current address and a fault indicator suffice as repair information in case of row redundancy and the faulty bit position together with the current address suffice in case of column redundancy.

Most address faults are detected and projected on the memory array in such a way that they are repaired properly (although in some cases cells with an AF which is disabled by a repair of another fault, will still be repaired). This means that if the faulty address(es) is replaced by a row, the memory is repaired.

Typically as part of our overall BIST algorithm, at the conclusion of the 20N operation, steps are added to test the BIST circuitry. We rearranged our algorithm to perform the BIST circuit test and retention test on all the memory modules in parallel prior to the 17N section of the algorithm. Memory that is part of the BISR circuitry is also tested using the 17N algorithm. Failures of any of these tests results in unrepairable memory.

3. Architecture

In evaluating redundancy approaches, three methods were considered: a) spare rows and columns, b) spare rows only or c) spare columns only.

The spare rows and columns method typically requires a complete fault mapping of the memory before determining replacement with a row or column. The spare allocation procedure would be quite complex.

For spare rows only or columns only, the spare allocation procedure can be simplified and is more viable. In addition, the spare allocation process can operate in parallel with the diagnostic algorithm. To determine the best method to implement, several factors must be considered based on the basic memory design and expected fault distribution. Either of these two methods would effectively replace single cell faults. However, for some classes of faults, row replacement would be much more effective than column replacement and vice versa. For example, spare rows would be effective repairing faults in a word line, a word line driver and a word line decoder. Spare columns would be effective for faults in a bit line, column multiplexer, sense amplifier, data input register, data output register and column line decoding. Because of this greater functional fault coverage, we used an approach based on spare columns.

An initial spare column implementation of Built In Self Repair embedded memory consists of four basic components:

- a standard SRAM array of memory cells,
- one or more replacement memory columns,
- a BIST/BISR Control Unit (BBCU) and
- a memory Reconfiguration Control Unit (RCU).

The basic memory repair architecture is illustrated in Figure 1. It only shows the blocks active in a normal operating mode, after the repair operation.

In addition to the four basic blocks described above, additional circuitry is included to re-route the spare columns to replace memory columns that contain defective cells. This circuitry includes input multiplexors to route the data to the correct functional columns and output multiplexors to route the data to the output data registers.

The BBCU (not shown) has Finite State Machines that control the operation of the BIST and BISR. For the BIST function, the BBCU uses an Address Generator for stepping through the memory, a Data Generator to provide test stimuli to the memory cells and an Output Data Evaluator to detect failures. The BBCU encodes defective address and bit information, storing it in the RCU.

The RCU is basically a memory whose data outputs provide signals to control the input and output multiplexors. In normal operation, the RCU memory addresses are driven from the upper address bits of the main memory. It behaves as a look-up memory, translating the main memory upper address bits (column selectors) into control signals, so its access time is critical. The overall access time of the BISR memory is derived from the access time of the SRAM, the access time of the RCU and the multiplexor decoding time. We used a high speed QRegfile memory for the RCU in our application. In our application this memory is 128 words by 30 bits. This memory is volatile, so the repair process must automatically operate at power on. Using a nonvolatile flash memory would remove this requirement.

Since defective memory cells are repaired by remapping replacement columns for their columns, a simple *greedy algorithm* for replacement is sufficient and straight forward to implement. That is, when a defective cell is detected and the column containing it is identified, the BISR controller can determine if there are any remaining

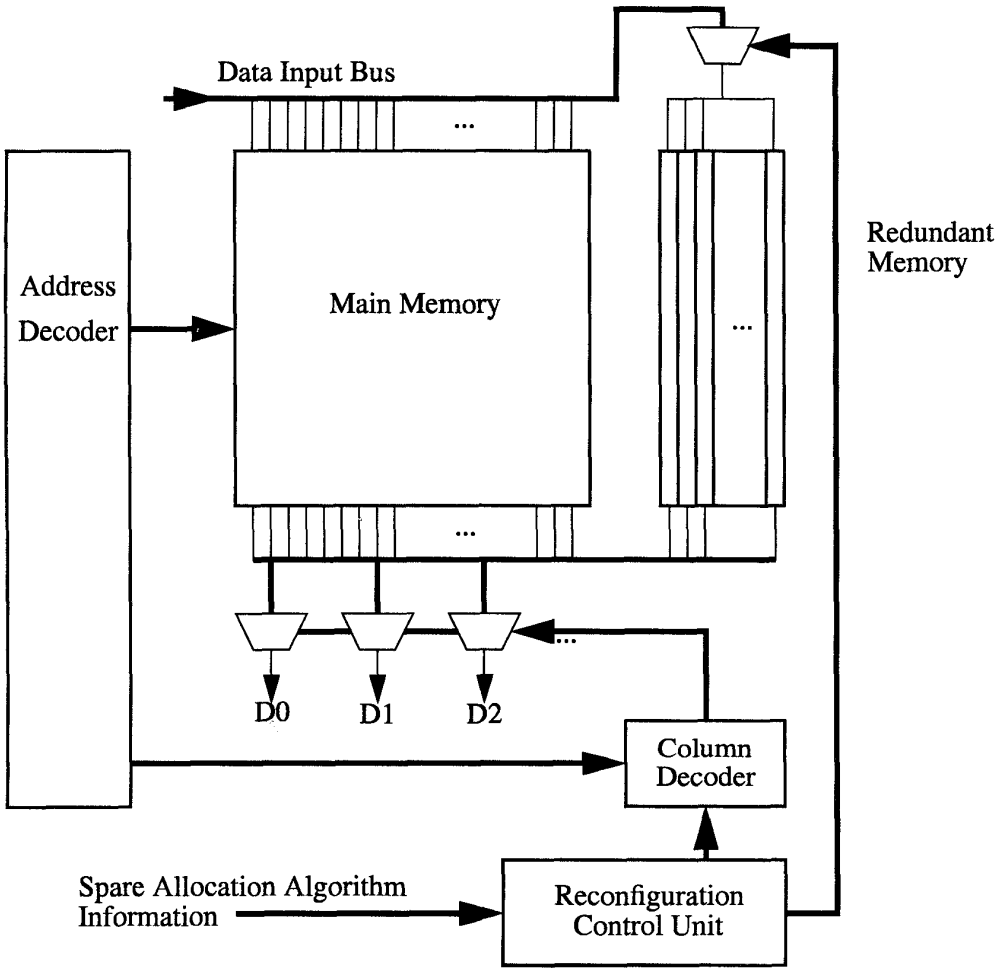

Figure 1. Block Diagram of a Basic Memory Repair Architecture.

spare columns left for repair. If there are, the RCU can be programmed replace the defective cell's column with the replacement column.

In order to minimize the performance impact of large multiplexors, the replacement columns are actually implemented by grouping the spare columns into blocks of columns. As a particular example for a 512 row by 1024 column memory arranged into 8 bit words, there might be two groups of eight spare columns. The Data Inputs are put into an 8:1 multiplexor to select the appropriate replacement bit when writing. For reading, spare 8:1 multiplexors select the column to send to the output multiplexors. The output multiplexors, in addition to being able to select the address decoded columns from the memory block, can also select a repair column when necessary. These multiplexors are driven by the values stored in the RCU Register and sent by the Decoder Control Logic. One implication of this scheme is that at most 2 bits can be corrected in any word (row).

In the current implementation, we classify the defects as two types: repairable defects and non-repairable defects. In particular, if any non-repairable defect occurs, the entire SRAM (and therefore the chip) must be discarded. The repairable type of defects includes the usual defects in the SRAM array as well as defects in the spare blocks. The non-repairable type includes defects in the RCU, data retention defects of all memory blocks, and overflows described in the Spare Allocation Algorithm below. The constraints imposed by the architecture are matters of on-going investigation, and need to be addressed in the future after their impact on final yield is determined.

4. Algorithm Flow

The RCU is first checked for defects using the 17N algorithm. Since defects here are non-repairable, checking this test first minimizes the amount of time wasted on non-usable chips. Then, a complete initialization of the RCU, SRAM, and spare block memory is performed by writing a memory pattern. An optional data retention test is performed by waiting a predetermined amount of time. The data is then verified by reading. During the same verification step the memory is rewritten with the complementary value to allow for checking the memory for the retention of that value. After another wait, the complementary value retention is verified. A walk and other address tests are performed to check for address decoder faults, as well as test the internal BIST circuitry. The RCU is then loaded with a pattern to indicate that the core memory is initially assumed to have no defects. The external circuitry and memory has been tested; we are now ready to proceed with the test and repair operation on the core memory.

The 17N algorithm is now applied to the SRAM array. As a defect is detected, the BBCU directs the flow to the Spare Allocation Algorithm, described below. Only one defect is detected and repaired during any single iteration of applying the 17N algorithm. This is done to minimize the size and complexity of the additional circuitry used in the BBCU and RCU.

During the first pass of the 17N algorithm, the entire core memory is checked regardless of any defects found. In fact, the RCU is disabled until the end of this pass, but its contents are updated as defined by the detection of defects. This first loop is called the *Replace When Done* mode in the Spare Allocation Algorithm. If no defects were detected, the algorithm signals a done condition. If any defects were detected, the test is restarted, now with the RCU enabled. The reason for a new loop is that up to this point the spare columns have only been tested for data retention. The 17N algorithm has not been applied to the spare memory, only the RCU. So any new columns must be tested and a simple way to do this (architecturally) is to rerun the test with the new columns in place.

These secondary loops are called the *Replace Immediately* mode in the Spare Allocation Algorithm. During the second and later iterations, the algorithm marches through successive addresses until a new defect is found and its column identified. If one is detected before the test completes, the Spare Allocation Algorithm is engaged. The RCU is updated to reflect the new failure, and the algorithm restarted without completion. This procedure continues until the array eventually completes the entire test, or the effort to repair is abandoned. The latter could occur if the RCU detects that there are no remaining spare columns, or if a large number of iterations have been made in an attempt to repair the array. The loop ends by signaling via a control signal that the algorithm has successfully repaired the core memory, or that the

device is unrepairable.

The Spare Allocation Algorithm is employed whenever the 17N algorithm finds a core memory fault. A flow chart of the Spare Allocation Algorithm is shown in Figure 2. This algorithm has the following characteristics:

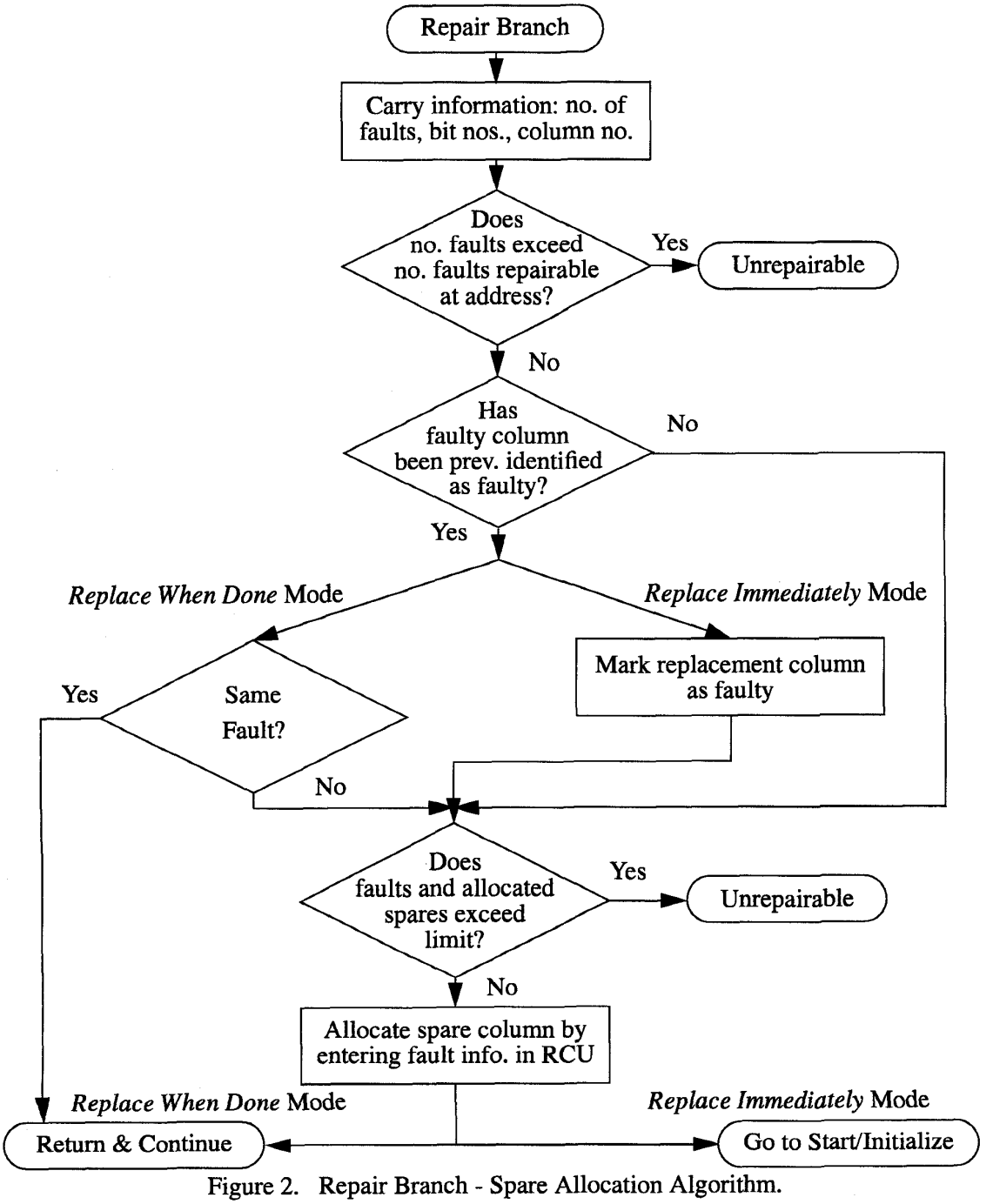

Figure 2. Repair Branch - Spare Allocation Algorithm.

— operates in one of two modes: a) *Replace When Done* - the repair information is stored in the RCU, but the RCU is not *enabled* until done; b) *Replace Immediately* - the repaired column is replaced immediately.

— does not allow duplicate repairs, *e.g.*, a faulty bit line could indict all bits in a column; the algorithm must handle this as a single column replacement.

— determines that the device is *unrepairable* when limits defined by the architecture are reached.

If a fault is found in the 17N algorithm, the following information is carried to the repair branch: number of faults, faulty bits and column number. The repair branch first checks if the number of faults exceed the repairable limit for a given address. It then checks if the faulty column had been previously identified. If so, one of two paths is followed - *Replace When Done* (RWD) or *Replace Immediately* (RI). In the RWD mode, if it is the same as a previously identified bit (column), the algorithm continues by returning control to the 17N algorithm. In the RI mode, the defective column is one that has already been replaced; it is marked as faulty and the algorithm will attempt replacement of another column. If the number of faults and allocated spares do not exceed the limits, a spare column is allocated by writing to the RCU. In the RWD mode, the algorithm returns control to the 17N algorithm. In the RI mode, the 17N algorithm must restart from the beginning.

5. Results

An initial implementation of this BISR architecture has been fabricated at Lucent Microelectronics. Using this BISR circuit, detection and repair of both single and multiple defects has been observed. Investigation is ongoing to determine the trade-offs of yield, performance, area overhead, and architecture for future designs.

As an example, the core memory size of our design is 1 M bit (64K x 16). The BISR area overhead, including routing and the RCU (128 x 30 memory) is < 4% of the 1M bit memory area. The spare columns are < 2% of the 1M bit memory area. This is very favorable when compared to DRAM's and other architectures.[6]

6. Summary

This paper describes a novel method for implementation of *Built-In Self Repair*. We have outlined a diagnostic algorithm, an architecture and repair algorithm that demonstrate an effective solution by straightforward modifications of the memory core and extensions of the BIST hardware.

In the current architecture, defects in the memory array are only covered by the use of additional spare columns. Further experience may show that this function may be better performed by the use of spare rows, or a combination of spare columns and rows. In the latter case, it has been shown that the problem of optimally utilizing the columns and rows to cover defective cells is NP-hard[11]. This implies that, especially in the case of Built In Self Repair, efficient approximation heuristics are needed.

7. Acknowledgments

We would like to acknowledge David Lepejian of Heuristic Physics Laboratories, Inc., who assisted in the analysis of the diagnostic algorithm. We would also like to thank Mike Depaolis at Lucent Microelectronics for providing management support and technical insight to pursue this project.

References

[1] T. Chen, G. Sunada, "Design of a Self-Testing and Self-Repairing Structure for Highly Hierarchical Ultra-Large Capacity Memory Chips", IEEE Trans.on VLSI Systems, Vol. 1, No. 2, June 1993, pp. 88-97.

[2] M. Franklin, K.K. Saluja, "Hypergraph Coloring and Reconfigured RAM testing", IEEE Trans. on Computers, Vol. 43, No. 6, June 1994.

[3] V.G. Hemmady, S.M. Reddy, "On the Repair of Redundant RAMs", 26th ACM/IEEE Design Automation Conference, pp. 710-712, 1989.

[4] R.W. Haddad, A.T. Dahbura, A.B. Sharma, "Increased Throughput for the Testing and Repair of RAM's with Redundancy", IEEE Trans. on Computers, Vol. 40, No. 2, Feb. 1991, pp. 154-166.

[5] R. Treuer, V.K. Agarwal, "Built-In Self-Diagnosis for Repairable Embedded RAMs", IEEE Design & Test of Computers, Vol. 10, No. 2, pp. 24-33, June 1993.

[6] D. Niggemeyer, J. Otterstedt, M. Redeker, "A Defect-Tolerant DRAM employing a Hierarchical Redundancy Scheme, Built-In Self-Test and Self-Reconfiguration", International Workshop on Memory Technology Design and Testing, Aug. 1997, pp. 33-40.

[7] M. Marinescu, "Simple and efficient Algorithms for Functional RAM Testing", 1982 IEEE Test Conference, pp. 236-239

[8] D.R. Aadsen, H.N. Scholz, Y. Zorian, "Automated BIST for Regular Structures Embedded in ASIC Devices", AT&T Technical Journal, Vol. 69, No. 3, pp. 97-109, May/June 1990.

[9] Y. Zorian, (1990) "A Structured Approach to Macrocell Testing Using Built-In Self-Test", Proc. of IEEE Custom Integrated Circuits Conference, 28.3.1-28.3.4.

[10] A.J. van de Goor, (1991). Testing Semiconductor Memories, Theory and Practice. John Wiley & Sons: Chichester, UK.

[11] S.Y. Kuo and W.K. Fuchs, "Efficient Spare Allocation in Reconfigurable Arrays", IEEE Design & Test, vol.4, pp.15-22, Feb. 1987.