RAM TESTING TECHNIQUES

Aly M. Hassan
Department of Computer Engineering and Science, Faculty of Electronic Engineering,
Menoufia University, Menouf, Egypt.

ABSTRACT

This paper presents a review and comparison for the work done in RAM testing methods. Types of faults and fault models are first presented, then the different methods of testing are explained. These methods were used to test a one megabit RAM (128 KByte) of an IBM-PC computer memory space. Results have shown that many methods are inadequate and takes a very long computer time. Due to the very high capacity of RAMs available today, this comparison is important for selecting the suitable test method for certain RAM according to its size.

Keywords: Digital testing, Functional testing, RAM testing, Diagnostic test algorithms.

1. INTRODUCTION

The memory is a subsystem of particular importance in most digital systems. However, apart from input/output, it is one of the less-reliable components within the basic digital system structure (control, memory, ALU, buses, general registers, etc.) [1]. In the early days of memory design, test procedures were developed in an ad hoc manner. Although memory size was limited to the order of a few tenth of kilobytes, the fault coverage of these ad hoc test procedures were also limited and often indeterminable.

A significant amount of work has been done in recent years to obtain fast and very large memory systems. As a result, the density of semiconductor memory chips has increased dramatically. With the increasing complexity, the efficient testing of such memories has been recognized as a difficult problem. Because the complexity of memories is quadrupling every two or three years, even a linear increase in testing time becomes inefficient for testing large memories. A multimegabit random access memory requires excessive time just to test only all cells stuck-at faults. To overcome this problem, researchers have sought to develop innovative test generation algorithms [6]-[13] and on-chip built-in test methods [14]-[18].

Unfortunately, RAMs are susceptible to many different forms of failure. The main problem in a test generation algorithm is how to provide a set of test vectors that are not excessively long and yet cover all the possible failures. Possible failure modes for RAMs include:

(i) incorrect addressing (decoder malfunction).
(ii) multiple writing (usually caused by capacitive coupling)
(iii) pattern sensitivity.
(iv) stuck-at 1/0 faults

In this context, the article starts by discussing how RAM chips are organized, various RAM fault models are reviewed and the related physical faults are highlighted. Different test algorithms and their fault coverage are presented. The address range from 016550 h to 03654F h (1 Mbits) in an IBM-PC memory space was used as an experimental test area for comparing the efficiency of each algorithm and the results are compared. Finally the built-in self-testing technique is presented and its different approaches are discussed.

2. RAM FAULT MODELS

Before discussing the important failure models, let us consider how RAM chips are organized in a digital system. A RAM chip consists of an array of

memory cells, an address decoder, address and data registers, and read/write logic. This basic architecture is generally known as the four quadrant architecture, Figure (1). Generally, an M-word * k-bit RAM is organized as k identical partitions. Each M-bit partition may itself be organized as p two dimensional arrays of m * n cells, such that M=p * m * n, where (p>1).

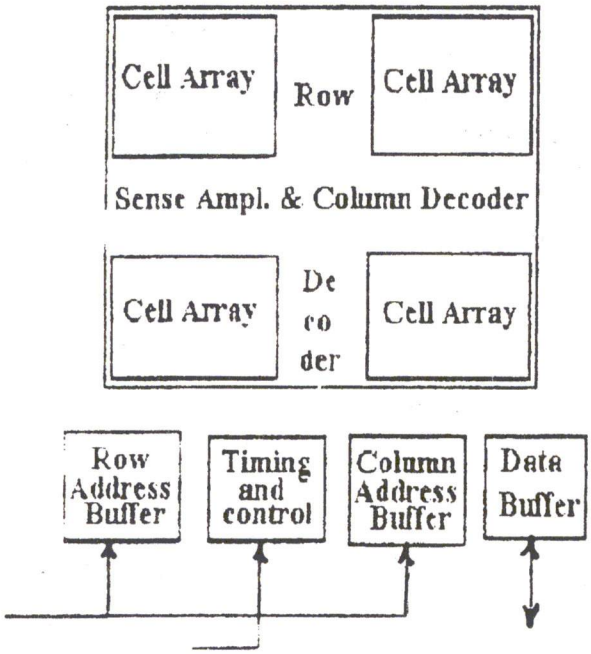


Figure 1. Four quadrant organization of a typical RAM.

A wide variety of physical failures can occur in memory array, address decoder , and read/write logic, causing various failures in the memory function. Their causes depend on some factors such as component density, circuit layout, and manufacturing method. A number of fault models have been developed to capture the effects of physical failures in RAMs. This section describes the important fault models relevant for the functional testing of RAMs. Faults not covered include soft faults such as transient faults and intermittent faults.

Generally, an assumption have been used in the development of all fault models and test algorithms, the Single fault assumption.

This assumption is justified by the frequent testing strategy , which states that we should test a system often enough so that the probability of more than one fault developing between two consecutive testing experiments is sufficiently small [2]. There are situations, however, in which frequent testing is not sufficient to avoid the occurrence of multiple faults. For example , in high- density circuits, like RAMs, many physical faults can affect an area containing several components and appears as a multiple fault.

A multiple fault means simultaneous presence of many faults in a circuit. To highlight the difficulty of the multiple fault testing problem, consider a circuit with n lines, while the number of single stuck-at faults is only 2n, the total number of all possible multiple and single stuck-at faults in the same circuit is 3n -1. It means, for a simple circuit with only 10 lines, that the number of single stuck-at faults is about 20, while the number of multiple and single faults is about 59000. In addition to very long test time needed, the detection of multiple faults is again complicated due to the masking effect. Fortunately, it was found that, in most cases, tests that detect all single faults often detect most multiple faults. This justifies the single fault assumption.

2.1 The Stuck-at fault model:

The simplest fault model is to consider that any line or memory cell may have fault which causes its contents to remain permanently either at logic 1 or at logic 0 (stuck-at 1 or stuck-at 0). Stuck-at faults are also useful for modeling faults in other parts of the memory system, such as the decoder [4].

To test a stuck-at fault on a line or memory cell, it should be sensitized by the logic value opposite to the stuck-at value. If no input combination exists which sensitizes the fault, or the fault effect can not propagate to an output, the fault is called redundant fault. The usefulness of stuck-at fault model results from the following characteristics:

1- It represents many different physical faults.
2- It is independent of technology.
3- Tests that detect this fault detect many other faults as well.

2.2 Coupling fault model [2]:

A pair of memory cells is said to be coupled if a transition in one of them changes the contents of the

other cell from one state to another . Consider the situation shown in Figure (2), cell i stores a logic 1 (right-most transistor Qi acting as a short circuit) and cell i+1 stores a logic 0 (left-most transistor Qi+1 acting as an open-circuit). If the state of cell i is changed, i.e. Qi drain goes to Vdd , then it may be possible for the change of Qi drain to couple through to Qi+1 via stray capacitance, and hence cause cell i+1 to change state erroneously.

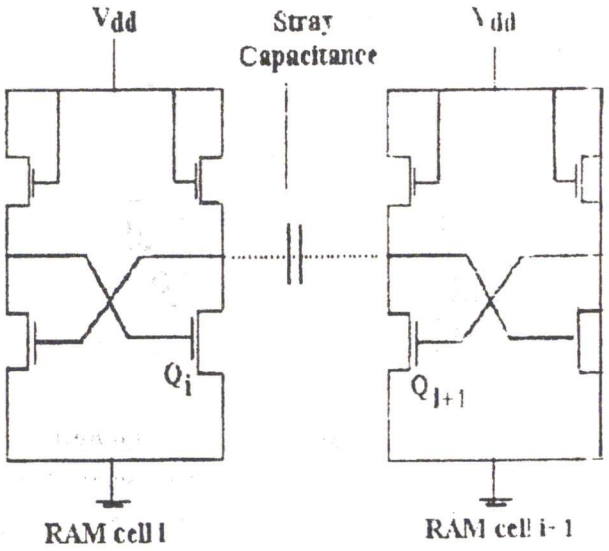


Figure 2. Coupling fault in RAM cells.

Coupling faults can occur in two ways, either the transition in one cell forces the contents of other cell to a certain value (1 or 0), or the transition causes an inversion in the contents of the second cell. Coupling faults could also exist between more than two cells. In [3], two test strategies for detecting RAM coupling faults were presented. In both strategies, the input test data is randomly selected and the output response is compressed in a signature analyzer. One test strategy uses random selection of both the address lines and the read/write control, while the other sequentially cycles through the address space with deterministic setting of the read/write control. The relative merit of the two strategies was measured by the average number of accesses per address needed to meet a standard test quality level. Results have shown that the deterministic strategy offers a better performance and is quite easy to implement.

2.3 Pattern-sensitive fault model:

A useful way to model faults in high density LSI, VLSI circuits, specially RAMs, is to consider the interactions logical signals that are adjacent in space or time [9-11]. Such a fault occurs when a signal X causes an adjacent signal Y to assume an incorrect value. Faults of this type are called pattern-sensitive faults [PSFs]. The high component and connection densities of RAMs aggravate this fault. Another instance of pattern sensitivity is the failure of a device to recognize a single 0 (or 1) that follows a long sequence of 1's (or 0's) on a particular line.

A memory cell i is said to have a pattern sensitive fault if its state is altered by a pattern of 0's and 1's, 0 -> 1 transition, 1 -> 0 transition, or both 0 -> 1 and 1 -> 0 transition in a group of other memory cells j,k,.... The effect of pattern sensitivity appears as a cell stuck- at fault or cell state transition fault for small duration or as long as locations j,k,.... contain the previous specific data or data transitions. The reason of such an influence was reported as a special case of state coupling.

To simplify testing, a simpler fault model known as neighborhood cell pattern sensitive fault model, is often used. In this model, only the data in the neighborhood cells (physical neighborhood) may affect the state of a cell. Common neighborhood models are the five-cell and nine-cell physical neighborhood shown in Figure (3).

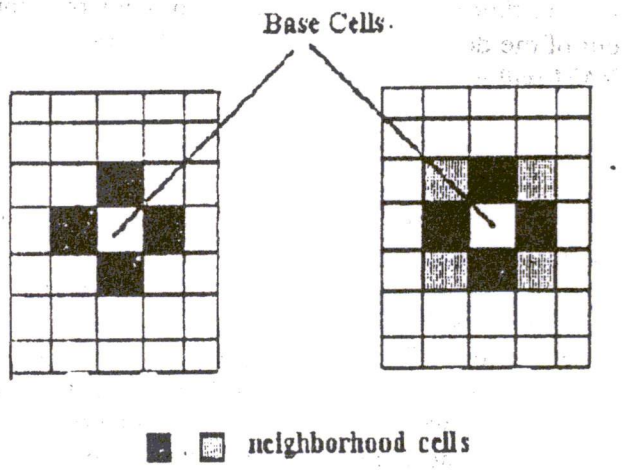


Figure 3. Five & Nine cell neighborhood.

3. RAM TESTING TECHNIQUES

All test algorithms consist of a sequence of writes and reads applied to the cells in the memory array. Over the years, several algorithms of different complexities have been developed to test RAMs. These algorithms can be categorized into two classes. In the first class, pattern- sensitive faults are the target, so the whole memory is read after changing the value of one or more cells. In the second class, stuck-at faults are the target, so only the cell of changed value is read.

It is important to note that most faults occurring in the address decoder and the read/write logic can be mapped to faults in the memory cell array, so decoder faults will be detected by tests for the memory cell array.

3.1 *Simple read/write test:*

This is a simple test procedure with very small fault coverage developed in an ad hoc manner [2]. The strategy is as follows:

step 1: write a 1 to each location in the RAM.
step 2: read each location and check that it is 1.
step 3: write a 0 to each location .
step 4: read each location and check that it is 0.

All this test will prove is that at least one of the cells in the memory works. This is because a fault in the address decoder may cause the same cell to be referenced each time. For example, if all address lines (either external address lines or lines that come out of the decoder) were bridged to 0 volts, then the RAM will still pass the test.

3.2 *The Galloping pattern algorithm:*

There are many variations for the galloping pattern algorithm, known generally as GALPAT [5]. Normally galloping-one followed by galloping-zero (or vice versa) sequences are used to achieve a complete test. The prime objective of the GALPAT sequence is to identify read-write disturbance problems between a given cell and all other cells. Effectively, therefore, the application of both forms of GALPAT (1s and 0s) should detect any pattern sensitivity problems.

The algorithm first initializes all memory locations to 0. Then for each cell it writes 1, reads all cells, then writes 0 back to the cell, this is the galloping 1. The procedure is repeated with galloping 0. To make the process clear we consider a simple memory array of only 9 cells. The galloping 1 part of the algorithm is shown in Figure (4). Each cell is read at least once, when it has a value of 1 as well as when it has a value of 0. Thus , all cell stuck-at-1/0 faults are detected. The switching of state also detects all state transition faults. As only one cell contains 1 at any time , address decoder faults are also detected. Any two arbitrary cells under states 00,01,10 are read, which covers a majority of state coupling faults [5]. The procedure looks like a 1 shifting its position from cell to cell throughout the whole memory.

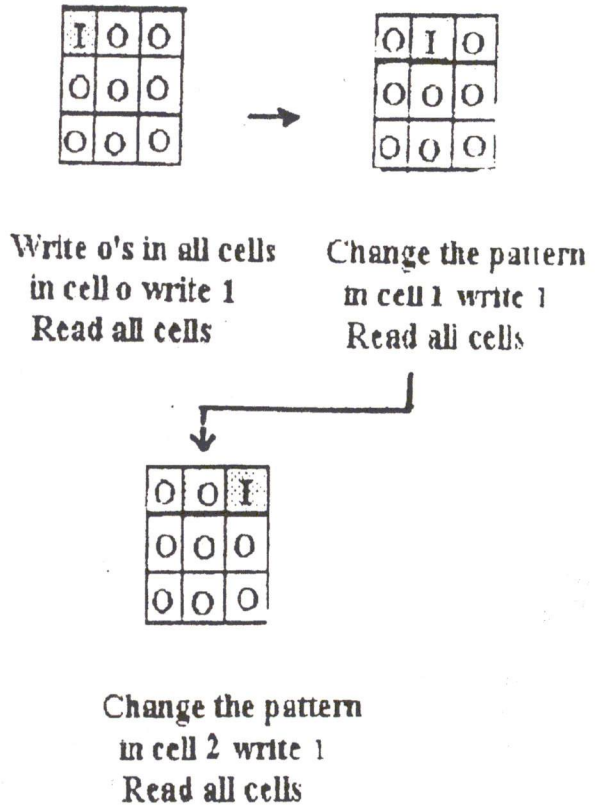


Figure 4. Galloping 1 pattern in Galpat test.

3.3 Check pattern test

This is a simple algorithm developed in an ad hoc manner [6]. The algorithm starts by filling the memory array with checkerboard pattern by writing

0's and 1's in alternate cells. the 9 cells memory array is shown in Figure (5.a), with first pattern. The whole memory is read, then cells are complemented as shown in Figure (5.b), and read again. The fault coverage of this test procedure is rather low. Cells stuck-at faults and about 50% of state transition faults are the only detected faults. Address decoder faults and state coupling faults are not covered. It is considered as a fast algorithm, but fault coverage is not high.

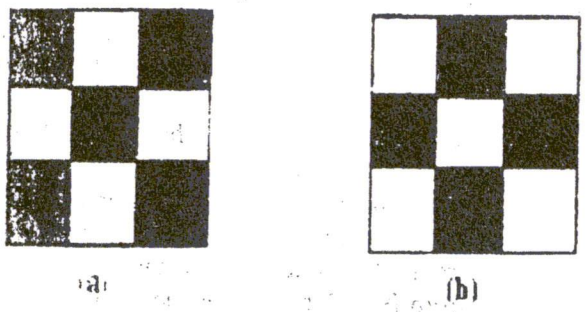


Figure 5. Check pattern test.

3.4 *Diagonal pattern test*

The strategy of the diagonal pattern is shown in Figure (6). Instead of shifting a 1 through memory cells, a complete diagonal of 1s is shifted , or a complete row or a complete column [5]. Although this reduces the required number of read/write operations, state coupling faults are not detected. The sequence will verify that the address decoders are functional. If multiple cells are selected (due to capacitive coupling between cell address lines) then this will be detected as a 1 read back from a cell off the diagonal. Similarly, if a cell is totally inaccessible, i.e. always wrongly addressed, then this will appear as either a 0 in the background of 1's or as a 1 in the background of 0's.

A variation of the diagonal pattern is to progress a single 1 through the memory array rather than a diagonal of 1's, reading the full array after each progression. The variation is called 'walking pattern' or WALKPAT. Obviously WALKPAT will take longer, but it is more effective [2].

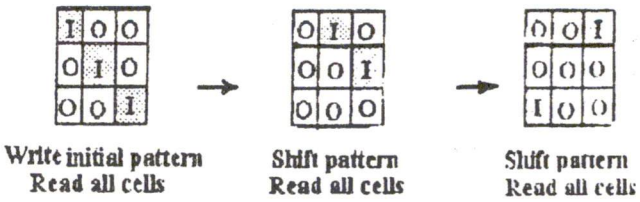


Figure 6. Diagonal pattern.

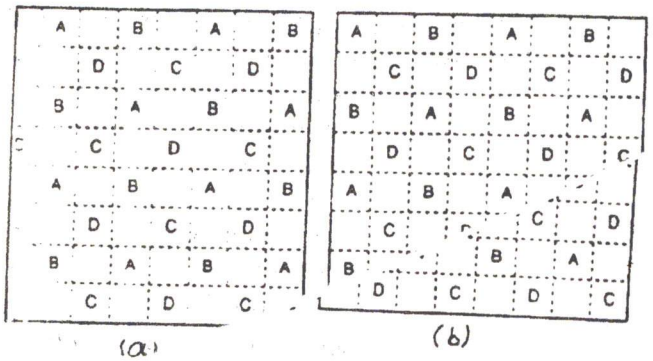
3.5 *Static pattern-sensitive-faults test (SPSF Test):*

Within the context of pattern sensitivity, different fault models have been proposed, based on the type of interaction between the cells[9-11]. In the SPSF model, a cell is said to be faulty if its contents change when a certain pattern of 0's and 1's exist in the neighborhood cells. In [11] an algorithm is proposed to detect five- cell-neighborhood SPSF. The algorithm is based on tiling the memory array, and was used later to implement a BIST arrangement. Figure (7) shows the tiling arrangements used by this algorithm. The unmarked cells are the base cells. Each base cell is surrounded by four characters (A, B, C, D). In the first phase of the test, which uses the tiling arrangement of Figure (7.a), the base cells are kept fixed at logic 0. The five-cell-neighborhood patterns are applied to the base cells using all four-tuples (16 patterns), consisting of variables A, B, C, and D. The base cells are read after the application of each pattern. The second phase uses the tiling arrangement of Figure (7.b), and the above process is repeated. Then both phases are repeated with the base cells at logic 1. The algorithm is considered near-optimal in fault detection of SPSF.

3.6 *Marching 1/0 test*

Perhaps this is the most widely used test due to its simplicity coupled with a moderate fault coverage [7]. The algorithm starts by initializing all memory cells to 0, then scans the memory cells in ascending and descending orders. For each cell, scanning involves reading the cell for the expected value, writing the complement value, and reading it again. The algorithm of the test is given below:

```
' total number of memory cells is N
for i=1 to N
  write 0 in cell (i)
continue
'.....Ascending loop ......'
for i=1 to N
  read cell (i)
    if contents=0 continue else Error
  write 1 to cell (i)
  read cell (i)
    if contents=1 continue else Error
continue
                '..... Descending loop ....'
for i=N to 1
  read cell (i)
    if contents=1 continue else Error
  write 0 to cell (I)
  read cell (i)
    if contents=0 continue else Error
  continue
End
```



(a) (b)

Figure 7. Tiling a memory array for the static-pattern-sensitive fault test
a- phase one b- phase two.

The idea of the algorithm is that, while scanning the memory in ascending order, any direct coupling between the current cell and a higher address is detected when reading the latter. Also any error in the higher address cell due to decoder faults will also be detected. In the same way, scanning memory in descending order detects all the effects on lower address cells.

3.7 *Modified Marching 1/0 test*

Dekker et al. [12],[13]have presented a modified version of the previous algorithm to cover 100% of the possible RAM faults. The algorithm covers all stuck-at faults, state transition faults, state coupling faults and decoder faults.

Each memory cell is initialized to 0 then read back to detect stuck-at 1 faults, and state 00 coupling for any two cells. Two cycles are then performed, one for each half of the total memory array. For each half, a 1 is written in each memory cell and read back to detect any cell stuck-at 0 fault, and state 11 coupling for any two cells. Also state transition from 0 -> 1 is detected. The operation is repeated in a second loop to detect any state transition faults from 1 -> 0.

The procedure is represented by the following algorithm which contains two loops. In each loop, two cycles exist, one for each half of memory array. Variable I is used to represent the cells in the first half from 1 to N/2, variable J for the cells in the second half from N/2+1 to N, it is clear that the two variables are related with j =N+1-i in both loops.

```
        '. total number of memory cells is N'
for i=1 to N
  write 0 in cell (i)
  read cell (i)
    if contents=0 continue else Error
  continue
                '..... First Loop .....'
for i=1 to (N/2)
  j =N+1-i
  write 1 in cell (i)
  read cell (i)
    if contents=1 continue else Error
    write 1 in cell (J)
    read cell (J)
    if contents=1 continue else Error
continue
        'Second Loop with Opposite Values'
for i= (N/2) to 1
  j =N+1-i
  write 0 in cell (I)
  read cell (i)
    if contents=0 continue else Error
    write 0 to cell (J)
    read cell (J)
    if contents=0 continue else Error
  continue
End
```

4. COMPARATIVE STUDY

Any RAM can be tested with any of the previous algorithms, irrespective of its type static or dynamic. Although some algorithms were applied on certain memory type, static or dynamic, this does not limit the generality of the algorithm [5]. It should be clear that not all RAMs exhibit all the previous mentioned faults. The test programmer should have a good understanding of the possible failures for a particular RAM. Unfortunately, in a user environment, test algorithm for certain RAM have to be developed with little, or none, knowledge of the internal organization of the memory array.

To help the test programmer in choosing a test which is suitable for his needs and secures certain target fault coverage, I present Table (1) which compares the previous test algorithms from point of view of their fault coverage.

Table 1. Comparison of fault coverage between RAM testing methods.

Method of Test	Stuck fault	Decoder fault	State Transition Fault	Pattern sensitive faults	State Coupling
SimpleR/W	yes	-	yes	-	-
GALPAT	yes	yes	yes	yes	yes
Check Pattern	yes	-	yes*	-	-
Diagonal	yes	yes	yes*	-	-
SPSF Test	yes	-	--	yes	-
Marching 1/0	yes	yes	yes	-	-
Mod. Marching 1/0	yes	yes	yes	yes	yes

yes ..... 100% fault detection
yes* .... 50% fault detection
- ..... fault undetectable

To give the test programmer the feeling of the time needed by each test, table (2) shows the analytical formula that represents the complexity of each test, where n represents the number of memory cells. Assuming that we have a tester running at 10 Mhz, the test time for each algorithm is also given.

Table 2. Test complexity and test time for the different test methods.

Method of Test	Complexity	1 Mbits Test Time
Simple R/W	4n	0.4 s
GALPAT	$4n^2$	11 hours
Diagonal	4n	0.4 s
SPSF Test	$4n^{3/2}$	0.4 s
Marching 1/0	4n	0.6 s
Mod. Marching 1/0	6n	0.7 s
	7n	

I have applied the various test methods presented earlier on 128 KByte RAM. The memory used was a part of an IBM compatible computer memory space. According to the memory map, the chosen target test area lies in the address range from 1635:0200 to 2655:FFFF which is a part of the user area. The used computer had an operating speed 33MHz, CPU 386DX, and 10 ns RAM access time. The different algorithms were written in assembly language using IBM Microsoft Assembler. Test times were found using DOS INT21 function 2C which delivers the time in hundredths of seconds. The test times obtained are as follows:

Simple R/W	GALPAT	Check Pattern	Diagonal	SPSF Test	Marching 1/0	Mod. Marching 1/0
0.4 S	> 1000 S	0.5 S	370 S	0.7 S	0.9 S	1.3 S

5. BUILT-IN SELF TEST (BIST) TECHNIQUES

For logic circuits, a random test pattern is often applied and the output response is compacted using linear- feedback shift register. For RAMs, however, random patterns are not as effective, this is due to the pattern sensitivity characteristic of RAMs [18-19], so few random testing approaches have been developed for RAMs. Yamada [14] performed an analysis about the probability of fault detection when using random test patterns applied to RAMs. His results have shown that, for the same fault coverage, the test length for RAMs using random patterns is 20 times longer than the marching test. In [20], a comparison was done between the efficiency of random pattern versus deterministic pattern testing of RAMs. The most interesting results was obtained when comparing test times for PSFs involving influential cells anywhere. Deterministic algorithms require (n+ 32n log n) time. On the contrary, a random testing requires a test time which remains linear as a function of n : 447n. Practically the fault model of influential cells anywhere is complicated, the five-cell- neighborhood is the most common used fault model in PSFs analysis. In this model, random test length is 1200n while that of deterministic test is 195n [20].

Previously, BIST logic design was driven in an ad hoc manner, it means, each BIST design is applicable only for a specific test algorithm which is related to a specific functional circuitry. New BIST design should be performed for each new algorithm realized. Much work was devoted to implementing features common to all test algorithms.

5.1 *BIST memory architecture*

Memory design engineers, usually, are restricted with design rules that require the maximization of the number of cells in a chip while minimizing the memory access time. This imposes hard constraints on the BIST logic designer. For example, the BIST designer tries to minimize the area occupied by the BIST hardware, the number of additional pins required, and the required test time. A generalized BIST memory architecture is given in Figure (8), it can be viewed as consisting of three modules [6] :

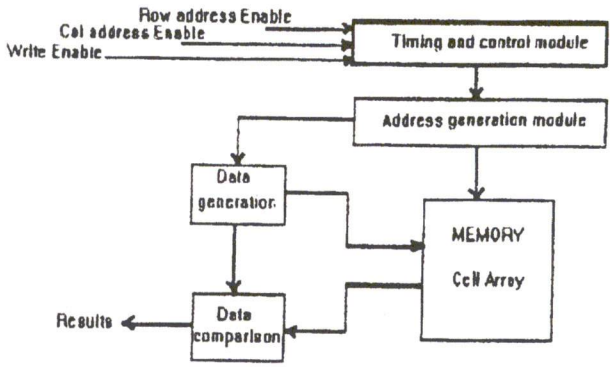


Figure 8. General block diagram of BIST logic for RAMs.

The control module:

This part starts and stops testing and supervises the control flow of the test algorithm. It can be implemented either using micro code or random logic. Although random logic design offers a high speed, recent designs has preferred micro code specially in large RAMs. This is mainly due to the less area overhead.

A secondary job for the control module is to set the BIST logic inactive during the normal mode of operation. The address generation module:

In most designs, the address generation is done using linear-feedback shift registers, counters or registers. Response verification module:

This unit produces the test patterns to be written in the cells. This module varies according to the test architecture, and different strategies can be used for data generation and response verification. Linear-feedback shift registers or counters can be used to generate test data. The correctness of the read values can be verified either by comparing them against the expected values or by signature analysis.

Although direct comparison is more accurate and can locate stuck-at faults, signature analysis is simpler in its hardware realization.

5.2 *BIST methods:*

Several BIST implementations have been proposed by researchers in the field [15]-[19]. Here only some implementation features are presented and compared to highlight the principle, a lot more can be found in literature.

5.2.1 *Self testing of dynamic RAMs:*

You and Hayes [15] suggested a BIST design using on- chip logic for test generation and response evaluation. In this design, the whole memory is divided into several blocks. In test mode, all memory cells in a block are connected to form a circular shift register. The test data, generated from the test generation logic, follows the marching 1/0 test algorithm. This test data is supplied to all memory blocks simultaneously. The on-chip logic scans all the blocks in parallel by concurrently shifting the data in the different blocks. After that, the scanned data and the responses from the different blocks are compared using on-chip comparison logic to detect a fault. The fault coverage of this approach is the same as that of the marching 1/0 test algorithm given in table(1). Since many blocks are tested in parallel and multiple bits of a block are accessed simultaneously, this scheme detects also a class of pattern sensitive faults in which a write operation becomes faulty in the presence of a few specific patterns in the cell's adjacent cells. The hardware overhead in this design is relatively high. For 1-Mbit RAM, this overhead is about 5%. Figure (9) shows an architecture for two memory blocks in this implementation.

5.2.2 *Parallel test using signature analyzer*

Sridhar [16], proposed a scheme that uses a parallel signature analyzer (PSA)to access bit lines from different memory arrays simultaneously. The analyzer operates in three modes: the scan mode, the write mode, and the signature read mode. The scan mode is used to load the analyzer with specific test pattern serially, it is also used to scan out the signature at the end of the test. In the write mode, the value stored in the analyzer is written to line bits in parallel. In the signature mode, memory cells contents are read and a new signature is generated which determines if an error has happened.

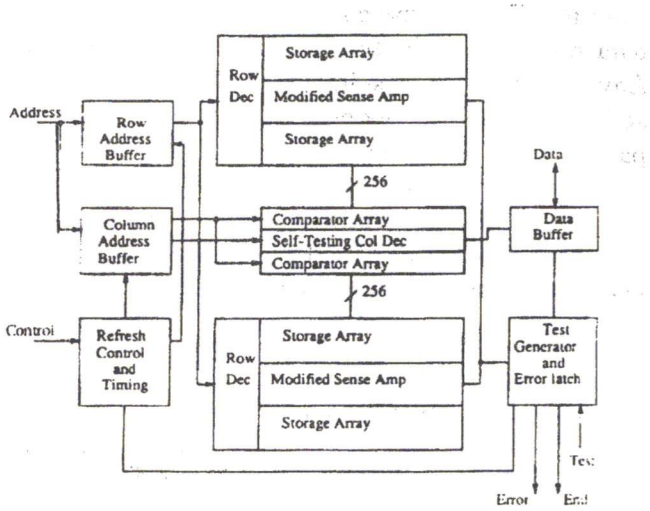


Figure 9. BIST of dynamic RAMs.

The major problem in the previous scheme is that it requires an external tester to scan in the data, which makes it not a fully BIST scheme. Another problem is the low fault coverage offered by the use of the marching test algorithm. Estimated area overhead for a 256-kilobit DRAM is 1 to 2.2%, while for 64-kilobit SRAM, it is 1.8 to 2.9%.

5.2.3 *Memory partitioning method:*

Another BIST scheme was proposed by Jarwala and Pradhan [17]. They partitioned the memory into small blocks and accessed them in parallel using a data bus. The partitioned memory blocks realize an H-tree through a set of comparators and switches between the blocks. The design was called the " TRAM".

In the TRAM architecture a RAM of size M=2m (m is the number of address lines) is divided into modules, which appears as a leaf node in a binary tree, Figure (10). Two types of nodes exist : memory nodes and switch nodes. The memory nodes have the cell array based on the traditional four-quadrant organization . Each switch node is a simple 1-out-of-2 decoder with buffers. Any test algorithm

can be used with the proposed method, and according to its choice, the fault coverage and the test patterns are determined. The testing procedure starts by writing the test patterns to all RAM nodes in parallel. During the read/verify part of the algorithm, the tester performs a read. All the nodes send their data to the comparators. This enables the data output of neighboring nodes to be compared. Any disagreement results in the FAIL signal being activated. If all the nodes agree on the outcome of a particular test, then they are all fault free. Unfortunately, the hardware overhead in this implementation is significant. For 1-Mbit RAM the hardware overhead is 15% when memory is partitioned into 8 blocks each of 128-kbits.

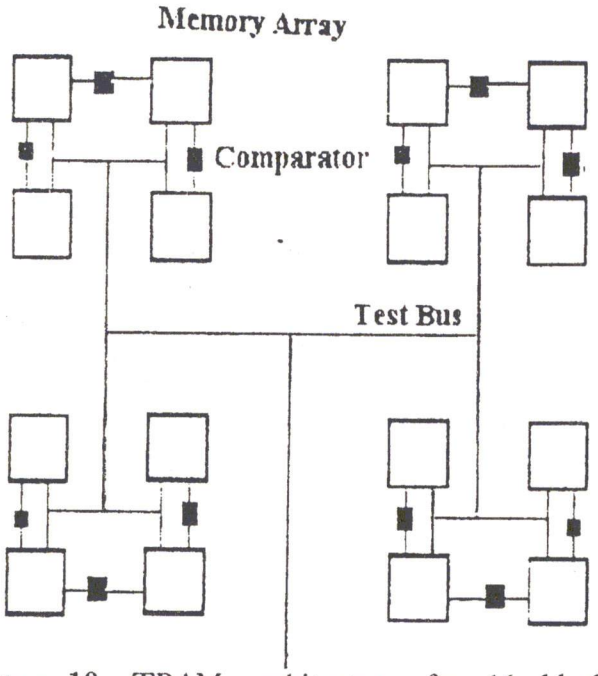


Figure 10. TRAM architecture for 16 blocks partitions.

5.2.4 *Comparison between BIST methods:*

The most negative aspect of all BIST methods is the extra hardware required. Generally, overhead is expressed as a percentage in comparison to the total memory area. We should be careful because area overhead can not be considered a sharp evidence about certain method efficiency. For example 10% overhead in 16-kbit memory can be acceptable, while the same overhead percent is not acceptable for 16-Mbit memory. Table (3) compares the previous three implementations.

Another drawback in BIST designs is the degradation in performance. The extra hardware increases parasitic capacitance which cause a large delay during normal operation. In some designs, sense amplifiers and address decoders are modified. This also increases the access time during normal operation.

Since SRAMs and DRAMs have a difference in their basic cell structure, some BIST arrangements were developed to work only with SRAMs while others to work with DRAMs and takes into account the additional refresh circuitry, also other arrangements work with both RAM types Table (3).

Table 3. Comparison of BIST implementations.

Bist implementation	Type of RAM	Area Overahead
Self testing of DRAMS	DRAM	High, for 1-Mbit RAM it is 5%
Parallel test using SA	DRAM. SRAM	High for several Mbits RAMs
Partitioning method	DRAM	Very high, for 1-Mbit RAM it is 15%

6. CONCLUSION

RAM testing is extremely time consuming process. For large RAMs, testing time should be considered the critical factor for estimating test algorithm efficiency. Here I have reviewed the common test methods and I have tried to highlight the points of strength and weakness in each. According to tables (1),(2),(3) the user is advised to choose the suitable test method for his fault model and also estimate accurately the test time according to the test complexity. Various design methods to enhance testability are also presented. The basic idea in all BIST techniques is the divide and conquer strategy, it means, memory is partitioned into smaller blocks which are tested separately. Each approach has its own advantages and drawbacks, but it seems that the most important comparison criteria is the area overhead.

REFERENCES

[1] E. Luque et al., "Fault-tolerant memory with content- recovery capability", *IEEE Proc.*, Vol. 128, Pt. E, No. 1, Jan. 1981, pp. 7-12.

[2] R.G. Bennetts, "Techniques for testing microprocessor boards", *IEEE Proc.*, Vol. 128, Pt. A, No. 7, Oct. 1981, pp. 673-691.

[3] J. Savir, W.H. McAnney and S.R. Vecchio, "Testing for coupled cells in Random Access Memories", *IEEE Trans. Comp.*, Vol. 40, No. 10, pp.1177-1180, Oct. 1991.

[4] M. Abramovici, M. Breuer and A.D. Friedman, *Digital systems testing and testable design*, Computer Science Press, 1990.

[5] R. Rajsuman, *Digital hardware testing: Transistor- level fault, modeling and testing*, Artech House Inc., 1992.

[6] M. Franklin and K.K. Saluja, "Built-in self-testing of random access memories", *Computer*, pp.45- 56, oct. 1990.

[7] A.J. van de Goor and C.A. Verruijt, "An Overview of Deterministic Functional RAM Chip Testing", *ACM Computing Surveys*, Vol. 22, No. 1, pp. 5-33, Mar. 1990.

[8] R. Nair, S.M. Thatte and J.A. Abraham, "Efficient Algorithms for Testing Semiconductor Random Access Memories", *IEEE Trans. Comp.*, 27(6), pp. 572-576, June 1978.

[9] J.P. Hayes, "Detection of Pattern Sensitive Faults in Random Access Memories", *IEEE Trans. Comp.*, 24(2), pp. 150-157, Feb. 1975.

[10] J.P.Hayes, "Testing Memories for Single Cell Pattern Sensitive Faults", *IEEE Trans. Comp.*, 29(3), pp. 249-254, March 1980.

[11] K.K. Saluja and K. Kinoshita, "Test pattern generation for API faults in RAM", *IEEE Trans. Comp.*, Vol. c-34, NO.3 pp. 284-287, March 1985.

[12] R. Dekker, F. Beenker and L. Thijssen, "A Realistic Fault Model and Test Algorithms for Static Random Access Memories", *IEEE Trans. CAD*, 9(6), pp. 567-572, June 1990.

[13] R. Dekker, F. Beenker. "Fault Modeling and Test Algorithm Development for Static Random Access Memories", *Proc. Int. Test Conf.*, pp. 343-352, 1988.

[14] T. Yamada, M. Saito and Y. Kasuya, "Test generation method for highly sequential circuits", *Proc. Compcon*, pp. 104-107, 1979.

[15] Y. You and J.P. Hayes, "A self-testing dynamic RAM chip", *IEEE Journal of Solid-state circuits*, 20(1), pp. 428-435, Feb. 1985.

[16] T. Sridhar, "A new parallel test approach for large memories", *IEEE design and Test*, pp. 15-22, Aug. 1986.

[17] N.T. Jarwala and D.K. Pradham, "TRAM: a design methodology for high performance, easily testable multimegabit RAMs", *IEEE Trans. Comp.*, 37(10), pp. 1235- 1250, Oct. 1988.

[18] E.J. McCluskey, "Built-in self-test techniques", *IEEE Design & Test of Computers*, pp. 21-27, April 1985.

[19] H. Fujiwara et al., "Test research in Japan", *IEEE Design & Test of Computers*, pp. 60-76, Oct. 1988.

[20] R. David, A. Fuentes B. Courtois, "Random pattern testing versus deterministic testing of RAM's", *IEEE Trans. Comp.*, Vol. 38, No. 5, pp. 637-650, May 1989.