

---

# Built-in Self-Testing of Random-Access Memories

Manoj Franklin and Kewal K. Saluja  
University of Wisconsin-Madison

**B**uilt-in self-test techniques are gaining ground in the testing of logic circuits because they offer a cost-effective way to test high-density digital devices. The basic philosophy behind the BIST technique is "let the hardware test itself" — that is, enhance the functionality of a logic circuit to test itself. The BIST concept was first proposed for combinational circuits, but it later found a quick application in the testing of such regular structures as random-access memories, read-only memories, and programmable logic arrays.

In the early days of memory design, test procedures were developed in an ad hoc manner. The fault coverages of these ad hoc test procedures were limited and often indeterminable. This shortcoming, acknowledged by most researchers, motivated the introduction of such fault models as stuck-at faults, decoder faults, coupling faults, and pattern-sensitive faults. By and large, the fault models have been simple. Until recently, researchers did not develop models covering complex cell interactions, because they believed that long tests would be required to detect such faults. In conventional testing environments with external testers, the only tests thought practical for large RAMs were those having a linear relationship with the number of bits,  $N$ , in the RAM. Ironically, the larger the RAM, the more complex the fault model required

---

**An examination of BIST schemes indicates that approaches based on test architectures rather than on test algorithms are more versatile and will likely predominate in the future.**

---

to effectively model the variety of physical failures that could occur because of interference between closely packed cells.

In a BIST environment, relatively inexpensive testers can perform functional test for testing RAMs. A BIST tester need only power up a chip, initiate the test signal, and read the chip's status. Therefore, much longer tests providing higher fault coverage without excessive cost can be applied.

The test time estimates in Table 1 (assuming a 10-megahertz clock) show that for large RAMs  $O(N^2)$ -length tests may be unacceptable (an  $O(N^2)$  test means a test of length  $CN^2$ , where  $C$  is a constant). Nevertheless,  $O(N^{3/2})$ -length tests can be practical for memories of 4 megabits or larger, depending on the extent to which the memory's internal organization can be exploited by the BIST logic.

Judging by the current trends, memory size will continue to increase. As memories become larger, BIST becomes more of a necessity because of the high costs incurred by off-line testers for even  $O(N)$  tests. Furthermore, even if the order of test length is moderately high, BIST techniques can bring down the effective test time by using such techniques as parallel testing and line-mode testing.

Another motivation for BIST is that the BIST logic incorporated in a chip can be used for both manufacture testing and in-circuit testing. If the implemented algorithm's test length is sufficiently small, the same BIST logic can even be used for testing RAMs during computer power-on, as part of the CPU's self-test procedures. Current BIST implementations cannot, however, be used for testing when the chip contains useful data.

Although BIST may still be a high-overhead concept (about 20 to 30 percent) for general integrated-circuit designs, it requires

**Table I. Test time for different test lengths and memory sizes (with a 10-megahertz clock).**

Test Length	Memory Size		
	1 Mbit	4 Mbits	16 Mbits
$N$	0.1 sec.	0.4 sec.	1.6 sec.
$N \log N$	2.1 sec.	9.2 sec.	40.3 sec.
$N^{3/2}$	1.8 min.	14.3 min.	1.9 hr.
$N^2$	30.5 hr.	20.3 days	325.8 days

little overhead (less than 1 percent) for application in large RAMs, as a later section shows. Furthermore, recent developments in very large scale integrated circuitry allow moderately complex test algorithms to be built within a chip.

In this context, the article has four major purposes:

- to demonstrate that BIST is a viable solution to the problem of testing large memories,
- to introduce the notion of generic test architectures suitable for implementing a wide variety of test algorithms,
- to provide a taxonomy for test architectures and use this taxonomy to categorize BIST implementations, and
- to survey the important BIST implementations reported by universities and industry.

A fair treatment of these four issues requires a discussion of the fault models and the test algorithms on which the BIST implementations are based.

## Fault models for RAMs

Before discussing the important fault models, let's consider how RAM chips are organized. A RAM chip consists of an array of memory cells, an address decoder, address and data registers, and a read/write logic. An  $N$ -bit RAM may be organized either as a single-bit output RAM ( $N$ -word  $\times$  1-bit RAM) or as a  $k$ -bit output RAM ( $M$ -word  $\times$   $k$ -bit RAM). Generally, an  $M$ -word  $\times$   $k$ -bit RAM is organized as  $k$  identical partitions. Each  $M$ -bit partition may itself be organized as  $l$  ( $l \geq 1$ ) two-dimensional arrays of  $m \times n$  cells, such that  $M = lmn$ . Cells and their contents in each of these arrays are independent of the cells in other arrays. By assuming that no interaction can take place between cells of different arrays, we can model faults considering only a two-dimensional array. Therefore, we need only

test each of these arrays completely, as opposed to testing the RAM as a single unit. The arrays can be tested sequentially or in parallel if the memory organization permits. The only restriction is that each array must be tested independently of the remaining arrays.

A wide variety of physical failures can occur in the memory array, address decoder, and read/write logic, causing various failures in the memory function. Their causes depend on such factors 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. In this section, we describe the important fault models relevant for the functional testing of RAMs using BIST. Faults not covered include soft faults such as transient faults and intermittent faults. A recent survey paper on fault models and functional testing techniques for RAMs provides more detailed descriptions.<sup>1</sup>

Invariably, two assumptions have been used in the development of all fault models and test algorithms: the single-fault assumption and the nondestructive or fault-free read operations assumption.

The *single-fault assumption* reduces the complexity of test procedures, which become unwieldy for most fault models if the test is designed to detect multiple faults. Tests that detect all single faults often detect most multiple faults. This justifies the use of the single-fault assumption.

The *fault-free reads assumption* has also been used for practical reasons. Test procedures for RAMs often have some form of embedded *checking experiment*, that is, the application of a sequence of writes to bring the memory to a known state and the verification of this state by reading the memory cells. The test procedure becomes extremely complex — and sometimes impossible — if the read operations are assumed to be faulty or destructive. In reality, however, most faults in the read/

write logic are easily detected because they result in catastrophic failures.<sup>1</sup> Simple tests can be derived and applied by an external tester in the final testing stages to detect noncatastrophic faults in the read/write logic.

**Stuck-at fault model.** A memory cell is said to be stuck-at-1 (stuck-at-0) if its contents remain fixed at logic 1 (0), irrespective of what is written into it. Stuck-at faults are also useful for modeling faults in other parts of the memory system, such as the decoder.

**Coupling fault model.** 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 0 to 1 or 1 to 0. Coupling faults are of two types. An *idempotent coupling fault* is one in which a transition in one cell forces the contents of another cell to a certain value (either 0 or 1), whereas an *inversion coupling fault* is one in which the transition causes an inversion in the contents of the second cell. Coupling faults could also exist between three or more cells.

**Pattern-sensitive fault model.** A memory cell is said to have a pattern-sensitive fault if its state is altered by a pattern of 0's and 1's,  $0 \rightarrow 1$  transitions,  $1 \rightarrow 0$  transitions, or both  $0 \rightarrow 1$  and  $1 \rightarrow 0$  transitions in a group of other memory cells. The group of cells that influences the base cell's behavior is called the *neighborhood* of the base cell. The problem of pattern sensitivity arises primarily from the high component densities of RAMs and the related effect of unwanted interacting signals. As RAM density increases, the cells become physically closer, and pattern-sensitive faults become the predominant faults. Moreover, other fault classes that affect the memory cells — shorts, stuck-at faults, and coupling faults — can be regarded as special types of pattern-sensitive faults.

Testing a RAM for unrestricted pattern-sensitive faults is impractical, as it requires an  $O(2^N)$  test.<sup>1</sup> This fact has led researchers to consider restricted pattern-sensitive fault models in which the neighborhood size is small. Another restriction is on the positions in the array that a neighborhood is allowed to take. Often the neighborhood is allowed to take only the position that physically surrounds the base cell. Traditionally, the restricted neighborhoods considered are the five-cell and nine-cell physical neighborhoods. Figure 1a shows the five-cell physical neighborhood of a memory cell.

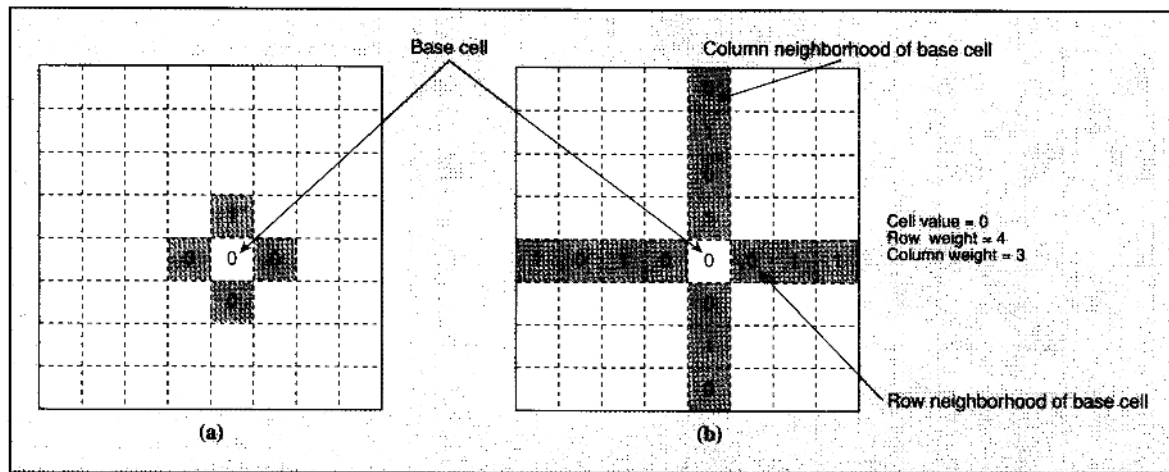


Figure 1. Different types of neighborhood: (a) a five-cell neighborhood; (b) a row/column neighborhood.

Within the context of pattern sensitivity, different fault models have been proposed, based on the type of interaction between the cells. In the *static-pattern-sensitive fault* model, a cell is said to be faulty if its contents change when a certain pattern of 0's and 1's exists in the neighborhood cells (that is, the pattern to which the cell is sensitive is static). A *dynamic-pattern-sensitive fault* is said to occur if the state of a cell changes because of a change in its neighborhood pattern. Researchers have also studied variations of these fault models, for example, those using active and passive neighborhoods.<sup>1</sup>

**Row/column weight-sensitive fault model.** The neighborhoods discussed so far for restricted pattern-sensitive faults are the physical neighborhoods of a cell. The row/column weight-sensitive fault model is based on the broader *row/column neighborhood*.<sup>2</sup> The row (column) neighborhood of a cell consists of all the cells in the same row (column) but excluding the cell. It is related to the electrical neighborhood of a cell because cells of the same row share a common word line, and cells of the same column share a common bit line. The row/column neighborhood is much larger than the conventional five-cell and nine-cell physical neighborhoods described earlier. Row weight of a cell is the number of 1's in its row neighborhood; column weight is the number of 1's in its column neighborhood.

Figure 1b shows the row/column neighborhood, row weight, and column weight of a memory cell. The row/column weight-sensitive fault model is based on the obser-

vation that the contents of a cell can be affected by the contents of cells in its row and column neighborhood. Interference could occur between cells of the same column or row, since these cells are electrically connected and share common addressing and refresh circuitry. In the row/column weight-sensitive fault model, a memory cell is said to be faulty if its content is sensitive to any combination of row and column weights.

Besides considering a larger neighborhood than the conventional five-cell and nine-cell neighborhoods, this fault model has an additional advantage: Tests that detect row/column weight-sensitive faults also detect most of the faults modeled by other fault models. Furthermore, weight-sensitive fault tests are also applicable for reconfigurable memory chips, whereas the five-cell-neighborhood pattern-sensitive fault tests are not.

**Faults in the decoder and read/write logic.** Most faults occurring in the address decoder and the read/write logic can be mapped to faults in the memory cell array; that is, during tests of the memory cell array, they will behave as faults in the memory cell array.<sup>1</sup> A stuck-at fault in the read/write logic will appear as a large group of memory cells with a stuck-at fault. Thus, an algorithm that detects stuck-at faults in the memory array can easily detect this fault. The same arguments are valid for coupling faults. Similarly, faults in the address decoder can be modeled by faults in the memory array, so the decoder faults will be detected by tests for the memory cell array.

## Test algorithms and their fault coverages

Over the years, several algorithms of different complexities have been developed to test RAMs. The early algorithms were ad hoc; the later algorithms were specifically designed to detect faults from various fault models. Recently, random pattern testing has also been proposed for RAMs. In this article, we discuss only those test algorithms (ad hoc or specific) that have been applied (in original or modified form) for BIST implementation in either universities or industry.

All test algorithms consist of a sequence of writes and reads applied to the cells in the memory array. In our discussion,  $W_i \leftarrow v$  denotes the operation "Write value  $v$  into cell  $i$ ." Similarly,  $R_i (= v)$  denotes the operation "Read cell  $i$ , with  $v$  as the expected value."

**Mscan test.** Memory scan is a trivial test procedure developed in an ad hoc manner. The Mscan test writes each cell, first with a 0 and then with a 1. Each value is verified by reading it before a new value is written. The formal algorithm is as follows:

```

For  $i = 0, 1, \dots, n - 1$ 
 $W_i \leftarrow 0$ 
 $R_i (= 0)$ 
 $W_i \leftarrow 1$ 
 $R_i (= 1)$ 

```

The deterministic fault coverage of this test procedure is rather low. All that is known at the end is that there is at least one

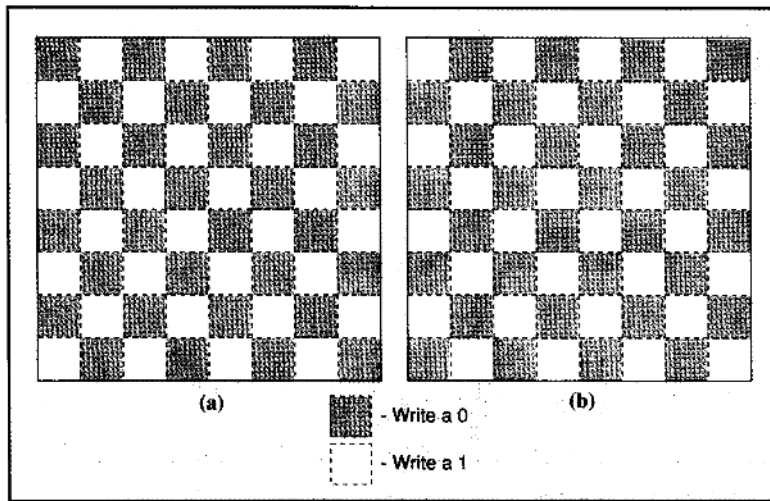


Figure 2. Tiling a memory array for the checkerboard test: (a) pattern one; (b) pattern two.

cell in the RAM that can be set to 0 and 1. This is because a fault in the decoder may cause the same cell to be referenced each time. Since the test performs four operations on each cell, its length is  $4N$ .

**Marching test.** Perhaps the most widely used test algorithm in the industry is the marching test. A reason for its popularity is its simplicity, coupled with a moderate fault coverage. The marching-test algorithm initializes the memory array to all 0's, and 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 idea behind this algorithm is that, while scanning the memory in ascending order, any direct coupling between the current cell and a higher address cell is detected when reading the latter. Along with this, any error in the higher address cell due to decoder faults will also be detected. Similarly, scanning the memory in the descending order detects all the effects on lower address cells.

The formal algorithm is given below; the details can be found elsewhere.<sup>1</sup>

Step 1.  $W_i \leftarrow 0$  for  $i = 0, 1, \dots, n-1$

Step 2. For  $i = 0, 1, \dots, n-1$   
 $R_i (= 0)$   
 $W_i \leftarrow 1$   
 $R_i (= 1)$

Step 3. For  $i = n-1, n-2, \dots, 0$   
 $R_i (= 1)$   
 $W_i \leftarrow 0$   
 $R_i (= 0)$

Step 4. Repeat steps 1 through 3, interchanging 0's and 1's.

The marching test detects all stuck-at faults and decoder faults. However, it does not detect all single coupling faults. Different variations of the marching test, all of length  $O(N)$ , have been suggested in the literature.<sup>1</sup>

**Checkerboard test.** This simple algorithm, developed in an ad hoc manner, is designed for two-dimensional memory architectures. The algorithm fills the memory array with a checkerboard pattern by writing 0's and 1's in alternate cells. Two patterns, as shown in Figures 2a and 2b, are written. The cells are read after the application of each checkerboard pattern.

Step 1.  $W_{(i,j)} \leftarrow 0$  for  $i+j = \text{even}$   
 $W_{(i,j)} \leftarrow 1$  for  $i+j = \text{odd}$

Step 2.  $R_{(i,j)} (= 0)$  for  $i+j = \text{even}$   
 $R_{(i,j)} (= 1)$  for  $i+j = \text{odd}$

Step 3. Repeat steps 1 and 2, interchanging 0's and 1's.

The deterministic fault coverage of this test procedure is rather low. As with the

Mscan test, a decoder fault may cause only four cells at most to be referenced. Therefore, all that is known at the end of this procedure is that at least four cells in the RAM can be set to 0 and 1.

**Five-cell-neighborhood static-pattern-sensitive fault test.** Many algorithms have been proposed to detect five-cell-neighborhood pattern-sensitive faults. All these algorithms are based on tiling the memory array. We briefly explain an algorithm reported by Kinoshita and Saluja, because this algorithm was later implemented as a built-in self-test.<sup>3</sup> Figures 3a and 3b show the tiling arrangements used by this algorithm for the static-pattern-sensitive fault test. The unmarked cells are the base cells. Each base cell is surrounded by four characters (A, B, C, D). The first phase of the test uses the tiling arrangement shown in Figure 3a. During this phase, 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 3b, and the above process is repeated. Then both phases are repeated with the base cells at logic 1.

**Row/column weight-sensitive fault test.** Different test algorithms of varying test lengths have been proposed for testing RAMs for row/column weight-sensitive faults.<sup>2</sup> All the tests are of length  $O(N^{3/2})$  and use divide and conquer by recursive partitioning as the basic strategy. First the border cells of an array are tested. Then the two middle rows and columns are tested, thereby effectively partitioning the array into four, as Figure 4 shows. Partitioning continues recursively until all the cells of the array are tested. Testing the border cells involves scanning the memory array as in the marching test, and this helps to detect decoder faults. The row/column weight-sensitive fault test also detects five-cell-neighborhood pattern-sensitive faults.

**Fault coverage.** All the algorithms discussed so far have been implemented as built-in self-tests. Some have also been implemented for embedded RAMs in application-specific integrated circuits. Table 2 lists the complexities and fault-detection capabilities of the algorithms. Blank entries indicate that those classes of faults are either not detected or detected only to a small extent. The entries marked "unidirectional" mean that a cell may be sensitive

to one or more patterns or transitions, but all of them change the cell's state from either 0 to 1 or 1 to 0. The table shows that the fault coverages offered by the Mscan test, marching test, and checkerboard test are rather poor.

## Test architectures

Generally, test algorithms for RAMs are developed assuming no knowledge of the internal organization of the memory array. This makes sense, because a test algorithm should be generic to be applicable to memories with different internal organizations. Quite often, the internal details of a RAM chip are not released by the vendor and therefore are not available to customers. The BIST logic designer, on the other hand, knows the internal organization and could use this knowledge to reduce the test time and area overhead with possible modifications to the test algorithm that do not sacrifice the fault coverage. For example, for reads and writes the BIST logic may be able to access multiple bits of an array in parallel if the technology and test algorithm allow, instead of accessing the bits serially. Similarly, the internal organization might permit the testing of multiple arrays in parallel. The modifications made to test algorithms to suit memory's internal structure are analogous to the modifications made to high-level-language computer programs by optimizing and vectorizing compilers that take advantage of the computer's internal organization.

So far, BIST logic design has been driven in an ad hoc manner by the desire to implement specific test algorithms. That

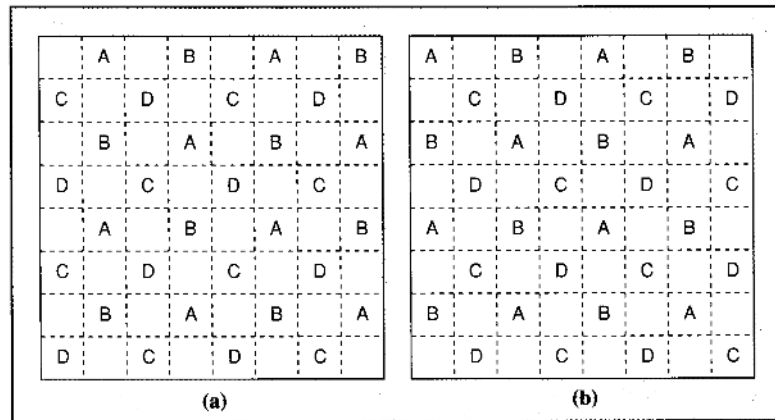


Figure 3. Tiling a memory array for the static-pattern-sensitive fault test: (a) phase one; (b) phase two.

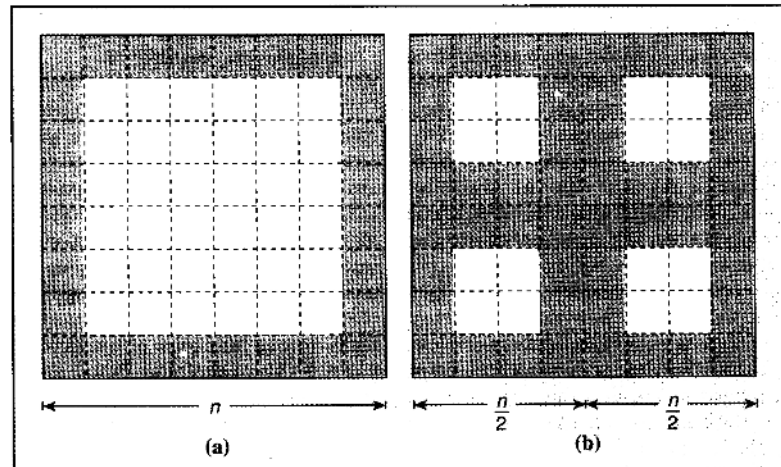


Figure 4. Partitioning a memory array into four in the row/column weight-sensitive fault test: (a) memory array with the border cells tested; (b) memory array partitioned into four.

Table 2. Summary of faults detected by the test algorithms.

Test Procedure	Order of Test Length	Detected Faults			
		Stuck-at-Faults	Coupling Faults	Restricted PSF	Row/Column WSF
Mscan Test	$O(N)$	Does not detect decoder faults			
Marching Test	$O(N)$	All	Does not detect all single coupling faults		
Checkerboard	$O(N)$	Does not detect decoder faults			
SPSF Tests	$O(N)$	All		Unidirectional	
Row/Column Test	$O(N^{3/2})$	All	Unidirectional	Unidirectional	Unidirectional

Mscan - memory scan  
PSF - pattern-sensitive faults

SPSF - static-pattern-sensitive fault  
WSF - weight-sensitive faults

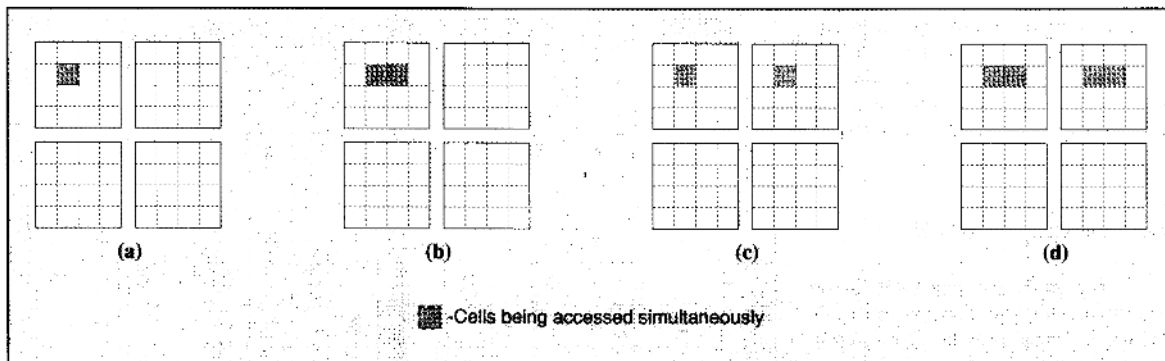


Figure 5. Memory cell accesses for (a) single-array single-bit, (b) single-array multiple-bit, (c) multiple-array single-bit, and (d) multiple-array multiple-bit architectures.

is, a new BIST logic design is carried out for each new algorithm implemented. Much design time goes into the implementation of features common to all test algorithms. This ad hoc approach also makes it difficult to integrate a number of test algorithms on the same chip. An alternative (structured) approach is to develop generic test architectures for implementing a number of test algorithms. For example, in a microcode-based test architecture, it might be possible to implement a class of different test algorithms by changing the microcode, just like changing the microarchitecture of a computer.\* The test architecture proposed by Matsuda et al. seems to be a step in the right direction.<sup>4</sup>

**Taxonomy.** Although a number of RAMs have been implemented with BIST functions, to the best of our knowledge, no one has classified these implementations. Here we provide a taxonomy for classifying BIST RAM test architectures. The categories in a rigorous taxonomy must be exclusive to avoid ambiguity and exhaustive to avoid incompleteness, providing an unambiguous category for every instance presented to it. Our taxonomy matches the number of simultaneously tested arrays and the number of simultaneously accessed bits within an array. Accordingly, we can classify all BIST RAM implementations into one of four test architectures:

- single-array single-bit,
- single-array multiple-bit,

- multiple-array single-bit, and
- multiple-array multiple-bit.

*Single-array single-bit* (SASB) test architectures are those in which a single array of the RAM chip is tested at a time and a single bit of the tested array is accessed at a time. Since a maximum of one bit from the entire memory chip is accessed at any instant, SASB architectures require the maximum amount of time for testing. Some classes of faults, such as arbitrary coupling faults, restrict the choice of test architecture to SASB architectures. Before the introduction of design for testability and BIST, external tester-based testing also limited the choice mostly to SASB architectures, because only one address can be transmitted from the tester to the chip at a time.

*Single-array multiple-bit* (SAMB) architectures test a single array at a time, but within the tested array, multiple bits are accessed simultaneously. Generally, the accessed multiple bits are all from the same row; multiple cells from the same column are not accessed simultaneously, as this slows memory access. Multiple bits can be accessed by modifying the address decoder. An SAMB test architecture in which all the  $n$  cells of a row (word) within an array are accessed simultaneously has often been referred to as *line-mode testing*.

*Multiple-array single-bit* (MASB) test architectures can be used if a memory chip is organized as a number of independent arrays, allowing multiple arrays to be tested simultaneously. A single bit from each array is accessed at a time. The concept is similar to the simultaneous testing of many memory chips using external test equipment. In the MASB architecture, a maximum of  $kl$  cells can be accessed at a time,

where  $kl$  is the number of arrays in the memory chip.

*Multiple-array multiple-bit* (MAMB) architectures use a combination of multiple-array and multiple-bit testing. A number of arrays are tested simultaneously, with a number of cells (normally within a row) in each array accessed simultaneously. Therefore, as many as  $kln$  cells can be accessed simultaneously. Sridhar's parallel simultaneous testing of a number of bits from *all* the arrays is an example.<sup>5</sup>

Figure 5 illustrates these concepts with a RAM organized as four arrays. The above discussion demonstrates that the SAMB, MASB, and MAMB architectures provide a speedup over the SASB architecture. An SAMB architecture can, at best, reduce the test length by a factor of  $n$ , if all the  $n$  cells in a row within an array are accessed simultaneously. The effective speedup can be less than  $n$  for certain classes of test algorithms because, during some stages of testing, these algorithms require the contents of part of the row to be kept unchanged when the rest of the row is being tested. Examples are the ping-pong test for coupling faults<sup>6</sup> and the row/column weight-sensitive fault test.<sup>2</sup> The MASB and MAMB architectures can give a maximum speedup of  $kl$  and  $kln$ , respectively.

Some algorithms are inherently serial and therefore do not attain the maximum speedup offered by the test architecture. Given a test algorithm, the BIST designer must choose one of the four architectures, considering test time, speedup, and technology. Alternatively, given a memory chip design, the BIST designer can select a test architecture based on the available technology and silicon area and then select test algorithms that can be implemented on the

\*This concept is similar to the idea of developing general-purpose computers (architectures) to do a variety of computations, as opposed to developing a special-purpose computer for each computation.

selected architecture. Such an approach uses more efficiently the silicon area set aside by the memory designers for the test logic. Only algorithms with good fault coverage should be selected.

All the test implementations reported so far can be categorized into one of the above four test architectures. Memory sizes have now reached the stage where an SASB architecture is almost impractical. Generally, as the memory size increases, the number of arrays increases, with the size of an array remaining more or less constant. Therefore, in the future we can expect many more designers to use the MASB and MAMB test architectures.

**Modifying test algorithms.** If the arrays are independent of each other and cells of different arrays do not interact, an algorithm developed for an SASB architecture need not be modified for a MASB architecture. However, modifications may be required to implement a conventional SASB algorithm in an SAMB or MAMB architecture.

Most test algorithms can be modified to benefit from the simultaneous access of multiple bits of a row. When multiple bits of an array are accessed simultaneously, faults due to interactions between the simultaneously accessed cells may not be detected, unless special care is taken. Sridhar describes a method to detect errors caused by interactions between cells accessed simultaneously.<sup>5</sup>

## BIST logic

Memory chip designers generally use aggressive design rules to maximize the number of cells in a chip and to minimize the memory access time. This imposes rather hard constraints on the BIST logic designer. In general, the BIST logic designer tries to minimize

- the area occupied by the BIST hardware,
- the performance penalty incurred for the normal memory operation,
- the number of additional pins required,
- the disparity between the functional speed and testing speed, and
- the test time, by using the memory's internal structure.

Conceptually, the BIST logic can be divided into four parts: control logic, address-generation logic, data-generation and response-verification logic, and test-trigger logic. Figure 6 (based on Ohsawa et

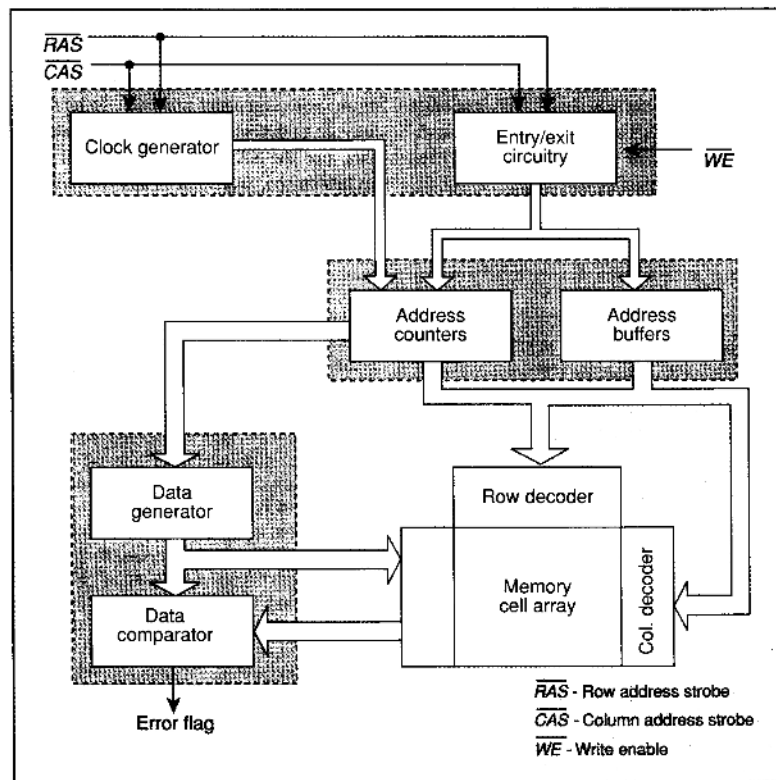


Figure 6. A generic block diagram of random-logic-based BIST logic for RAMs (based on Ohsawa et al.<sup>7</sup>).

al.<sup>7</sup>) shows a generic BIST organization for testing RAMs. We have modified the figure to show the main parts of the BIST logic clearly.

**Control logic.** The control logic initiates and stops testing and supervises the control flow of the test algorithm. It can be implemented using random logic or microcode. Random logic offers higher speed and has traditionally been used for designing the control logic; nevertheless, recent designs seem to prefer microcode-based control. For large memories (4 megabits and more), microcode-based BIST design has been shown to have an area overhead which does not exceed that of random-logic-based designs.<sup>3</sup> Therefore, the flexibility and implementational ease offered by microcode makes it superior to random logic for large RAMs.

Microcode fits well with RAM technology because of its regular structure. For BIST RAMs, it may be even more area efficient than random logic, because the aggressive design rules used for RAM cells

can also be used for the microcode array. Furthermore, the designer can use such microcode-optimization techniques as microprocedures, microstacks, and encoding by grouping of microinstruction fields, developed for microcode-based computers.

**Address-generation logic.** Almost all test algorithms require the addresses to be generated in a fairly uniform manner. The control logic can be designed to generate the addresses, but leaving this task to a separate unit is better. For most algorithms, address generation can be achieved by linear-feedback shift registers, registers, or counters, with occasional intervention from the control logic. With the MASB and MAMB architectures, a single address-generation unit can be used for testing multiple arrays.

**Data-generation and response-verification logic.** The data-generation unit produces the test pattern(s) to be written in the cells. Given a test architecture, differ-

ent strategies can be used for data generation as well as response verification. Linear-feedback shift registers or counters assisted by the control logic can generate data. In an SASB architecture, the correctness of the read values can be verified either by comparing them against the ex-

pected values or by signature analysis. Direct comparison is superior, because it can locate single stuck-at faults. Furthermore, with signature analysis, some faults may go undetected because of aliasing errors. For the SAMB, MASB, and MAMB architectures, other fault-detection meth-

ods — in addition to comparison against expected values — are comparison of values read from multiple bits, AND reading, and OR reading.

In the MASB and MAMB architectures another convenient verification method is comparing the outputs of symmetrically

## BIST implementations

Below we examine several academic and commercial BIST implementations. Our purpose is not a definitive comparison; rather, it is to discuss important implementation features such as test algorithms, test architectures, and control logic. We have tried to include most of the BIST implementations reported in the literature and to present them more or less in chronological order. The terminology is that used by the authors.

**On-chip compact test scheme.** Saluja, Sng, and Kinoshita proposed a BIST design for testing RAMs for five-cell-neighborhood pattern-sensitive faults.<sup>1</sup> They grouped the patterns required to test a cell for such faults into different classes, depending on the "weight" of the patterns. Then they developed procedures to apply these patterns using optimal test length sequences. The grouping of the patterns helped to keep the test generators and response verifiers simple for BIST implementation, without causing information loss due to data compaction at the response-verification end.

For the control logic, they proposed both random-logic-based and microcode-based designs. They also proposed a simple and effective instruction set for the microcode-based implementation. The on-chip-compact test scheme was proposed and studied in an SASB architecture, although the feasibility of using it in a MASB environment was considered. Estimated area overhead for a 64-kilobit static RAM is 1.21 percent; for a 1-megabit static RAM, 0.09 percent.

**Parallel test using signature analyzer.** Sridhar proposed a scheme that uses a parallel signature analyzer to access bit lines from different arrays simultaneously.<sup>2</sup> The parallel signature analyzer operates in three modes: the scan mode, the write mode, and the signature/read mode. In the scan mode, the analyzer is loaded with a specific pattern from outside the chip by serially scanning in the bits. This mode is also used for scanning out the contents of the analyzer (the signature) at

the end of the test. In the write mode, the value stored in the analyzer is written to a number of bit lines in parallel. Finally, in the signature mode, the contents of the memory cells written earlier are read, and a new  $k$ -bit signature is generated. This signature determines whether an error has occurred.

This is not a fully BIST scheme, because it requires the scanning in of data from outside the chip. The scheme uses the marching-test algorithm, modified to suit parallel testing. It can be categorized as a MAMB approach, as the parallel signature analyzer can access multiple bits from multiple arrays simultaneously. The MAMB architecture results in very fast testing.

A potential problem with the scheme is that it requires an external tester to scan in the data. Another problem is the low fault coverage offered by the marching test. If a wide enough parallel signature analyzer is used, then the probability for aliasing errors will be very low. The error-detection capability can be significantly enhanced by monitoring the quotient bit of the analyzer, in addition to verifying the final signature. Estimated area overhead for a 256-kilobit dynamic RAM is 1.0 to 2.2 percent; for a 64-kilobit static RAM, 1.8 to 2.9 percent.

**Self-testing dynamic RAM.** You and Hayes proposed another type of parallel testing: reconfiguring the cells of an array into a circular shift register and using a built-in test generator to test multiple bits concurrently.<sup>3</sup> The dynamic RAM is organized as two identical arrays, and the arrays are tested in parallel to reduce the overall testing time. The test patterns are generated by reconfiguring the cells of each array to act as a circular shift register during testing. When a row (that is, a word line) within an array is activated, the contents of the  $n$  cells of the row are transferred to  $n$  bit lines, sensed by  $n$  sense amplifiers, and then written to the adjacent cells in the same row. Each  $n$ -cell row thus acts as an  $n$ -bit shift register. The data shifted out from the right-most sense amplifier is saved in a flip-flop and is stored in the left-most cell of the next row in the next shift cycle.

Thus, all  $m$  rows of an array effectively form an  $mn$ -bit shift register. By saving the initial contents of the right-most cell of the last row, the array realizes an  $mn$ -bit circular shift register. The standard sense amplifier circuits are modified so that when a bit value is read from one cell, it can be written into the adjacent cell in the same row. This is accomplished by introducing pass transistors between the physically adjacent bit lines and may adversely affect the sensitivities of the sense amplifiers and the RAM access time in the normal operation mode.

An on-chip comparison circuit consisting of exclusive-OR gates detects faults by comparing the outputs of symmetrically placed cells of the two arrays. The self-testing dynamic RAM implementation can be categorized as a MAMB test architecture, since two arrays are tested in parallel and multiple bits of an array (all the bits of a row) are accessed simultaneously. This scheme detects bit-line imbalance faults and restricted types 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. It does not detect faults caused by transitions in the neighborhood. The area overhead for a 4-kilobit dynamic RAM is about 12 percent, and the estimated overhead for a 1-megabit dynamic RAM is about 5 percent.

**Parallel testing for VLSI memories.** Inoue et al. proposed the line-mode test, a special case of SAMB testing.<sup>4</sup> In the line-mode test, all cells connected to a word line are tested simultaneously. The on-chip test circuit can perform parallel write and parallel compare. The parallel write circuit writes data into all cells connected to a word line, and the parallel compare circuit compares the data in parallel with the expected data. Apart from the memory cell arrays, separate tests check the decoders, the test logic, and the I/O circuits. The memory cells are tested with the marching-test algorithm.

The test circuit occupies less than 1 percent of the chip area for a 2-megabit dynamic RAM. The parallel write operation allows only certain patterns to be applied to the cells; therefore, the technique





Column Address Strobe. Using unique timing sequences is better than using over-voltages and extra package pins, because the latter methods may be incompatible with existing systems. Also, the overvoltage method requires either an additional power supply or the generation of an addi-

tional voltage signal. Voss et al. describe an implementation of a 256-kilobit  $\times$  1-bit static RAM with multiple test modes in which the test modes are entered by a unique timing sequence.<sup>8</sup> They also describe how a particular test mode can be selected using the normal address input

pins, if there are multiple test modes. Miyaji et al. describe the design and implementation of a test trigger circuit for megabit static RAMs that uses the Chip Enable and Write Enable signals to generate a unique timing sequence for entering the test mode.<sup>9</sup>

10-bit control store, which controls the initialization, sequencing, and completion of testing. The control store is conceptually divided into four microroutines. Control passes from one microroutine to another when the former issues a call signal to the latter; control passes back to the former when the latter issues a return signal. A microstack stores the return addresses in the proper order during nested calls. A 4-bit microprogram counter points to the microinstruction currently being executed.

The address-generation logic consists of a register file and some combinational

logic (glue logic). The registers hold the row and column addresses of the cell being tested and the cell being read or written. The width of the registers depends on the memory organization and the size of the cell array. The microcode initializes and updates the register file.

A major innovation of this scheme is the implementation of a moderately complex algorithm with a small control store, using microcode-optimizing techniques such as microprocedures and microstacks. The row/column weight-sensitive fault test has higher fault coverage than the other algo-

rithms (see table below). A potential problem with the SASB implementation is that the test time is comparatively long. However, the test time can be reduced by using the MASB or MAMB test architectures. The area overhead of the random logic design for a 4-megabit RAM is less than 0.8 percent.

**Serial interfacing for embedded-memory testing.** Nadeau-Dostie, Silbert, and Agarwal proposed a serial interfacing scheme for testing embedded RAMs.<sup>9</sup> Embedded RAMs are on-chip RAMs

**Summary of different implementations. (Fault coverage for each implementation can be inferred from Table 2 in the main text, which shows the coverage for each type of algorithm.)**

Implementation	Algorithm	Test Architecture	Control Logic	Type of RAM
On-chip compact test scheme	SPSF test	SASB	Random logic microcode	SRAM
Parallel test using signature analyzer	Marching test	MAMB	Random logic	SRAM, DRAM
Self-testing DRAM	Restricted PSF test	MAMB	Random logic	DRAM
Parallel test for VLSI memories	Marching test	SAMB	Random logic	DRAM
Parallel test for PSFs	PSF test	SAMB	Random logic	DRAM
Built-in processor for self-test	Not specified	SAMB	Processor	
CMOS DRAM with BIST	Checkerboard test	MAMB	Random logic	DRAM
Row/column test implementation	Row/column test	SASB	Random logic microcode	SRAM
Embedded-memory testing	Marching test Galpat Walk	MAMB	Random logic	SRAM
16-Mbit CMOS DRAM	Marching test Mscan test	SAMB	Microcode	DRAM

MAMB - multiple-array multiple-bit  
SAMB - single-array multiple-bit  
SASB - single-array single-bit  
SPSF - static-pattern-sensitive fault  
PSF - pattern-sensitive fault

## Future trends

BIST technology combines several different areas: fault models, test algorithms, test implementation, and fault diagnosis. Changes can be expected in each of these areas as technology progresses. Below, we

describe how these areas are likely to be affected.

**Fault models.** State-of-the-art memory chips are designed with spare rows and columns meant for reconfiguration. During manufacture, the memory is tested and

repaired (if necessary) by bringing in the spare rows and columns. With such new fault-tolerance techniques as dynamic reconfiguration, fault models based on logical adjacency become irrelevant, whereas those based on physical adjacency and electrical connectivity become more relevant. Future fault models must consider the effects of such reconfiguration within memory chips. With the new high-speed RAM realizations, such as gallium arsenide RAMs, future models must also consider delay faults.

**Test algorithms.** When a memory chip is reconfigured, physically adjacent cells may no longer have consecutive addresses. Test algorithms for the detection of physical neighborhood pattern-sensitive faults have to account for this fact. Furthermore, we believe that there will be a trend to find optimal or near-optimal, yet simple, test algorithms. These will save testing time as memory size continues to grow, and BIST logic will be used for maintenance testing of RAMs embedded in a system.

**Test implementation.** When a memory chip is being used in a system (that is, when the chip contains valid data), it cannot be tested on line because the test procedure might destroy the memory contents. Future systems may implement BIST algorithms that have on-line test capabilities. Further research is required not only to develop such algorithms, but also to determine the merits and demerits of such an approach, especially since most memory systems use error-correction code at some level.

**Fault diagnosis and self-reconfiguration.** In general, current BIST implementations cannot diagnose faults. In the future, BIST will potentially be used in field diagnosis. Such diagnosis will help in reconfiguration of memory chips and repair of multichip memory modules (silicon mass storages).

The separation of test algorithm and test architecture clearly shows the range of possible implementations for a given test algorithm. The test architecture-based approach is more versatile than the ad hoc design approach for BIST logic design, especially with various design constraints. It also facilitates the integration of a number of test algorithms within the same chip.

Our taxonomy for classifying BIST ar-

whose address, data, and read/write controls cannot be directly controlled or observed through the chip's I/O pins, making them good targets for BIST applications. The implemented scheme involves shifting data from one memory cell to another, similar to the self-testing dynamic RAM method described earlier. Although both schemes use the MAMB architecture, there are some differences. While the self-testing dynamic RAM scheme shifted data only within an array and independently tested two arrays in parallel, the new scheme shifts data within an array as well as across arrays, by shifting the data at the end of one array to the beginning of next array in a daisy-chained fashion. This makes sharing the BIST logic among multiple arrays easier, because fewer interconnection lines need to be routed between the BIST logic and the RAM blocks. Furthermore, in the serial interfacing scheme, multiplexers implement the shifting along the I/O data path. Therefore, no modification is required in the RAM. The implemented algorithms are adaptations of the marching test, Galpat (galloping patterns), and walk algorithms.

**16-Mbit CMOS DRAM with BIST function.** Using microcode-based control logic, Takeshima et al. have implemented the marching test and a scan read/write test with a checkerboard pattern for a 16-megabit dynamic RAM.<sup>10</sup> The size of the control store is 18 × 10 bits. Perhaps this is the first industrial BIST RAM implementation using microcode. The dynamic RAM enters the test mode through a unique timing sequence.

## References

1. K.K. Saluja, S.H. Sng, and K. Kinoshita, "Built-In Self-Testing RAM: A Practical Al-

ternative," *IEEE Design & Test of Computers*, Vol. 4, No. 1, Feb. 1987, pp. 42-51.

2. T. Srihar "New Parallel Test Approach for Large Memories," *Proc. Int'l Test Conf., Computer Society Press, Los Alamitos, Calif., Order No. 641 (microfiche only)*, 1985, pp. 462-470.
3. Y. You and J.P. Hayes, "A Self-Testing Dynamic RAM Chip," *IEEE J. Solid-State Circuits*, Vol. 20, No. 1, Feb. 1985, pp. 426-435.
4. J. Inoue et al., "Parallel Testing Technology for VLSI Memories," *Proc. Int'l Test Conf., Computer Society Press, Los Alamitos, Calif., Order No. 798 (microfiche only)*, 1987, pp. 1,066-1,071.
5. P. Mazumder and J.H. Patel, "Parallel Testing for Pattern-Sensitive Faults in Semiconductor Random-Access Memories," *IEEE Trans. Computers*, Vol. 38, No. 3, Mar. 1989, pp. 394-407.
6. H.C. Rittler and B. Muller, "Built-In Test Processor for Self-Testing Repairable Random Access Memories," *Proc. Int'l Test Conf., Computer Society Press, Los Alamitos, Calif., Order No. 798 (microfiche only)*, 1987, pp. 1,078-1,084.
7. T. Ohsawa et al., "A 60-ns 4-Mbit CMOS DRAM with Built-In Self-Test Function," *IEEE J. Solid-State Circuits*, Vol. 22, No. 5, Oct. 1987, pp. 663-668.
8. M. Franklin, K.K. Saluja, and K. Kinoshita, "Built-In Self-Test Algorithm for Row/Column Pattern Sensitive Faults in RAMs," *IEEE J. Solid-State Circuits*, Vol. 25, No. 2, Apr. 1990, pp. 514-524.
9. B. Nadeau-Dostie, A. Siburt, and V.K. Agarwal, "Serial Interfacing for Embedded-Memory Testing," *IEEE Design & Test of Computers*, Vol. 7, No. 2, Apr. 1990, pp. 52-63.
10. T. Takeshima et al., "A 55-ns 16-Mb DRAM with Built-in Self-Test Function Using Microprogram ROM," *IEEE J. Solid-State Circuits*, Vol. 25, No. 4, Aug. 1990, pp. 903-911.

chitectures provides a framework to describe widely differing implementations at a level of abstraction that eliminates many algorithm-related details, while preserving the important implementation characteristics. We expect that most future implementations in large RAMs will use the test-architecture-based approach, since it can easily adapt to changes in technology. ■

## Acknowledgments

This work was supported by the University of Wisconsin Graduate Research Committee, an IBM graduate fellowship, and the National Science Foundation under contract MIP 8509194. We thank the referees for their comments and suggestions, which greatly enhanced the quality of presentation of this work.


## References

1. A.J. van de Goor and C.A. Verruijt, "An Overview of Deterministic Functional RAM Chip Testing," *ACM Computing Surveys*, Vol. 22, No. 1, Mar. 1990, pp. 5-33.
2. M. Franklin, K.K. Saluja, and K. Kinoshita "Built-In Self-Test Algorithm for Row/Column Pattern Sensitive Faults in RAMs," *IEEE J. Solid-State Circuits*, Vol. 25, No. 2, Apr. 1990, pp. 514-524.
3. K. Kinoshita and K.K. Saluja, "Built-In Testing of Memory Using an On-Chip Compact Testing Scheme," *IEEE Trans. Computers*, Vol. 35, No. 10, Oct. 1986, pp. 862-870.
4. Y. Matsuda et al., "New Array Architecture for Parallel Testing in VLSI Memories," *Proc. Int'l Test Conf.*, Computer Society Press, Los Alamitos, Calif., Order No. 1962, 1989, pp. 322-326.
5. T. Sridhar "New Parallel Test Approach for Large Memories," *Proc. Int'l Test Conf.*, Computer Society Press, Los Alamitos, Calif., Order No. 641 (microfiche only), 1985, pp. 462-470.
6. M.M. Breuer and A.D. Friedman, *Diagnosis and Reliable Design of Digital Systems*, Computer Science Press, Potomac, Md., 1976.
7. T. Ohsawa et al., "A 60-ns 4-Mbit CMOS DRAM with Built-In Self-Test Function," *IEEE J. Solid-State Circuits*, Vol. 22, No. 5, Oct. 1987, pp. 663-668.
8. P.H. Voss et al., "A 14-ns 256K x 1 CMOS SRAM with Multiple Test Modes," *IEEE J. Solid-State Circuits*, Vol. 24, No. 4, Aug. 1989, pp. 874-880.
9. F. Miyaji et al., "Multibit Test Trigger Circuit for Megabit SRAMs," *IEEE J. Solid-State Circuits*, Vol. 25, No. 1, Feb. 1990, pp. 68-71.



**Manoj Franklin** is a PhD student in computer science at the University of Wisconsin--Madison. He has been awarded an IBM graduate fellowship. Recently he was a summer intern at Cray Research. His research interests include high-performance computing, memory testing, built-in self-test, and design for testability. Before graduate studies, he was an engineer at BHEL, Bangalore, India.

Franklin received his BSc (engineering) in electronics and communications from the University of Kerala, Trivandrum, India, in 1984.



### IEEE EDUCATIONAL ACTIVITIES INTRODUCES

**ICASSP/90**

The following four video tutorials were recorded April 1990 in Albuquerque, New Mexico at ICASSP/90.

- The Structure of FFT and Convolution Algorithms, presented by James W. Cooley, IBM T.J. Watson Research Labs
- High-Resolution and Higher-Order Spectral Analysis, presented by Larry Marple, Chief Scientist, ORINCON Inc.
- Synthetic Aperture Radar: A Signal Processing Viewpoint, presented by David C. Munson, Jr., University of Illinois
- VLSI For Signal Processing, presented by Edward A. Lee, UC Berkeley and K. Wojtek Przytula, Hughes Research Laboratories

ICASSP/90 Package (All Four Programs)  
IEEE member price, \$225.95, list \$129.50.  
Plus shipping and handling.

Individual tutorials IEEE member price, \$65.95, list \$135.95. Plus shipping and handling. PAL video standard available upon request.


**VLSI  
Design Principles and  
Practices**

This Self-Study Course provides practical application of comprehensive information on circuit and logic design offering an opportunity to become a valuable interface for VLSI selection.

Program includes a study guide, answer book, applications workbook, diskette, and a textbook, *VLSI Handbook* by Joseph DiGiacomo, McGraw Hill, 1989.

VLSI is IEEE member price, \$249, list \$498. Plus shipping and handling.

For a full description of these programs and complete ordering information, call IEEE at 1-800-678-IEEE, FAX# 201-981-1686, or Telex: 0833233. For shipments to CA, DE, NJ, and NY add appropriate sales tax. Please call for appropriate overseas Air Mail Charges.



IEEE Educational Activities, 445 Hoes Lane  
PO Box 1331, Piscataway, NJ 08855-1331



**Kewal K. Saluja** is an associate professor in the Department of Electrical and Computer Engineering at the University of Wisconsin-Madison, where he teaches logic design, computer architecture, microprocessor-based systems, VLSI design, and testing. Previously, he was at the University of Newcastle, Australia. He has also held visiting and consulting positions at the University of Southern California, University of Iowa, State University of New York, and Hiroshima University. His research interests include design for testability, fault-tolerant computing, VLSI design, and computer architecture.

Saluja received his BE in electrical engineering from the University of Roorkee, India, in 1967, and his MS and PhD from the University of Iowa in 1972 and 1973.

Saluja can be contacted at the Department of Electrical and Computer Engineering, University of Wisconsin, 1415 Johnson Dr., Madison, WI 53706.