

12 Memory Testing

12.1 Motivation

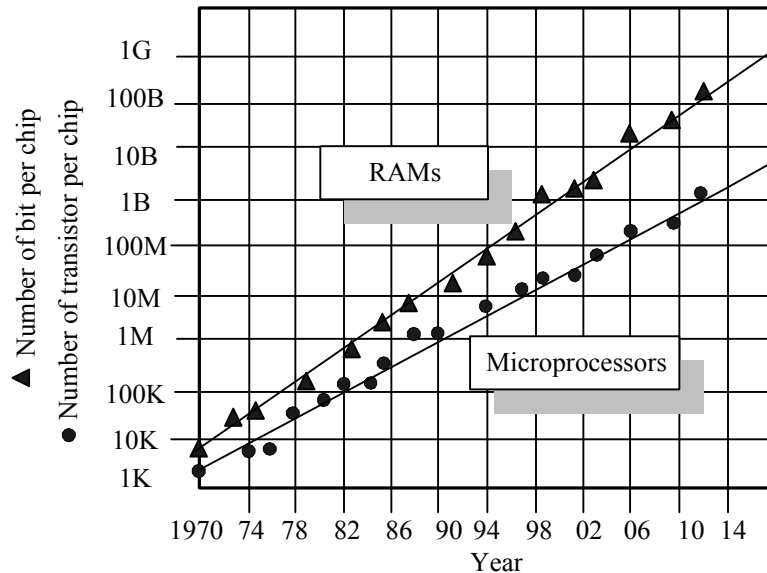


Figure 12.1 Growth of Memory Chips [Nat. Tech. Roadmap 1997]

There are a few very good reasons why memory testing deserves special attention. First, memory is vital to electronic products. We hardly find a digital system that does not include a certain type of memory. Second, like other digital circuits, the device density and the circuit complexity of memory chips are continuously increasing. This trend is shown in Fig. 12.1. The dramatic rate of increase is even higher than that of microprocessors. Third, even though unlike most digital circuits, memory chips have regular structures that initially may imply an ease of testability, the regularity, together with sequential circuit property of the cells, in some ways causes testing problems that are not necessarily encountered in other regular combinational logic circuits, such as the array multipliers that we discussed in Chapter 8. The fourth reason for testing complexity is the proliferation of memory types. Volatile memories, also known as random access memories (RAMs), may be static (SRAMs) or dynamic (DRAMs). Some nonvolatile memory devices are

read-only memories (ROMs), programmable read-only memories (PROMs), and erasable (reprogrammable) memories (EPROMs), UV erasable memories (UVPROMs), and electrically erasable memories (EEPROMs). Flash memories are also being used whenever high-density nonvolatile memories are needed. In this chapter we concentrate on RAM testing.

In Section 12.2 we present RAM models and architectures. Fault models for these RAMs include traditional fault models encountered in combinational and sequential circuits such as stuck-at and bridging faults. In addition, there are faults that better represent failures in RAMs. These are coupling fault (CF) and pattern-sensitive fault (PSF) models [Hayes 1975]. Fault models will be discussed in Section 12.3. RAM testing is covered in the next three sections. This includes functional testing and built-in self-test.

12.2 Memory Models

Circuit models are useful in simulation as well as development of algorithms for test pattern generation. Models for memory chips are available at different levels in each domain of design hierarchy: physical, logic, and system. Here we consider the logic and circuit levels to discuss failure modes in the cells and the functional level to generate test patterns.

A generic functional model of a RAM is shown in Fig. 12.2. The storage elements are usually organized to form the *array*. In this figure the array consists of $r \times c$ -bit words. This array is surrounded with peripheral circuits. The address from the *memory address register* (MAR) is decoded by row and column decoders to select the appropriate row or column of cells to be accessed. Writing and reading is then performed using the *write drivers* and the *sense amplifiers*, respectively. The data to be written to the memory or obtained from the memory are usually placed in the *memory data register* (MDR). In addition to these different components, a refresh circuit is needed when a dynamic RAM is used.

The memory array consists of rows and columns of memory cells. The cells in each word are arranged in a row and selected by the word line (WL). The data are supplied (write operation) and collected (read operation) through the bit lines (BLs) to all cells in a column. However, one cell

and only one cell of the column is affected by the operations at a time. In word-oriented memory all bits in the word are written or read simultaneously. In content-addressable memories (CAMs), the bits in a column are, instead, addressed simultaneously.

12.2.1 Functional Model

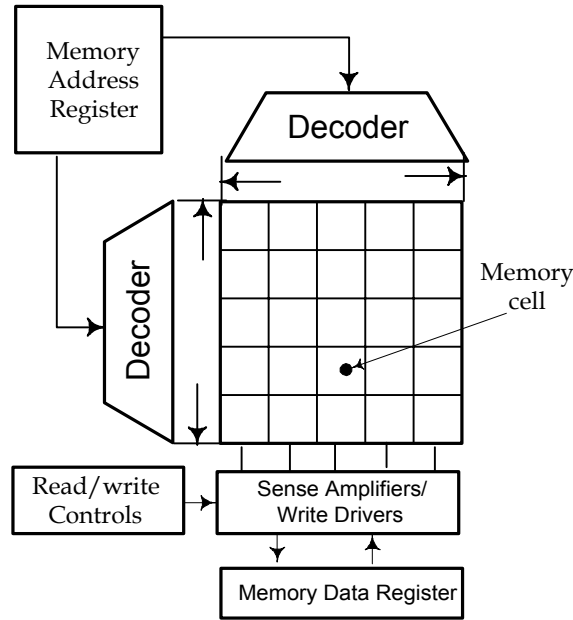


Figure 12.2 Functional Model for a RAM Chip

12.2.2 Memory Cell

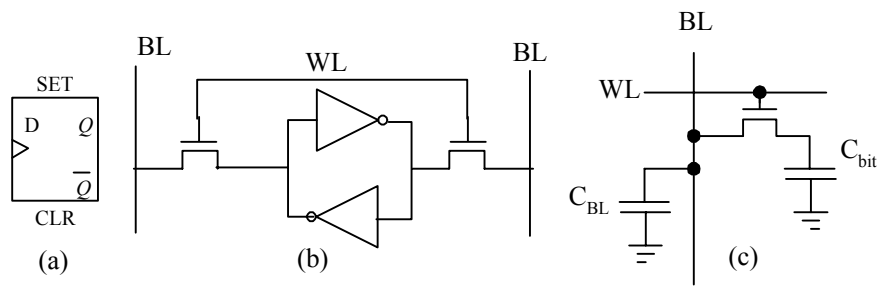


Figure 12.3 Memory Cells: a) Logic, b) Static CMOS, c) Dynamic MOS.

On the logic level, the memory cell may be represented by a D flip-flop on which one can write or store a bit, 0 or 1. The information is held until it is overwritten. It is assumed that when the cell is

not accessed, it retains the stored value. The retention depends on the type of cell implementations. Cells may be *static* or *dynamic*. The terminology refers to the method according to which the charge is stored in the cell. A very commonly used static RAM (SRAM) cell consists of two cross-coupled inverters, as shown in Fig. 12.3b. To *write* on the cell, the data and its complement are placed on bit lines BL and BL' (respectively). Then the word line is held high and the pair of inverters will store the new information. To *read* from the cell, the bit-lines are precharged to complementary values, then the word-line is again held high and one of the inverters will discharge in one of the bit lines.

Other implementations are also possible using fewer transistors. However, this number can really be decreased using dynamic logic. The dynamic RAM (DRAM) cell may consist of three, two, and even one transistor. An example of a one-transistor cell is shown in Fig. 12.3c. This cell relies on storing the bit value in a MOS capacitor [Rideout 1989]. The write operation is done in a fashion similar to the static cell. The read operation is performed by charging the BL to V_s , then holding WL high. The storing capacitance, C_{bit} , discharges in the bit line, BL. The read operation is destructive and thus requires a write back.

Because of charge sharing, this type of cell is more difficult to design. In addition, because of the possibility of charge leakage, it is necessary to *refresh* it. Refresh circuitry performs periodical data update on a row basis. No bit-line selection is necessary. As the bits of the word are read, they are then rewritten. Despite all these extra operations, dynamic RAMs are still the most widely used because compared to static RAMs, they are more compact and can operate at higher speeds. They are becoming even more compact using stacked and trench capacitors [Watanabe 1995, Tsuruda 1997].

12.2.3 RAM Organization

Two important factors that affect test pattern generation are the architecture of the array as well as whether the memory is an independent IC or a macrocell inserted as one of modules in a system on a chip (SOC). An N -bit RAM may be organized as arrays of N by 1 or r by c , where $rc = N$. For

example, a 16kB RAM may consist of 128 rows and 128 columns or 256 rows and 64 columns, or yet, 64 rows and 256 columns. One of the criteria for selecting one array versus another is space availability on the floor of an SOC. The array may also be partitioned in subarrays that are organized to share the sense amplifiers.

With the rapid increase in device density, functions that used to be on a separate chip can now be combined in one IC. This approach minimizes off-chip interactions, increases the bandwidth, and therefore improves performance. This integration is not limited to random logic only but to a mix of subsystems, including RAMs. For example, memory-intensive ASICs such as DSP processors and microprocessors, in general, include several on-chip RAMs. These on-chip RAMs are called *embedded RAMs*. It is almost certain that any system on a chip (SOC) nowadays includes embedded RAMs. Because embedded RAMs cannot be directly accessed for controllability and observability, they are difficult to test. BIST seems a more feasible approach to test them [Zorian 1990]. Memory BIST is the topic of Section 12.6.

12.3 Defects and Fault Models

12.3.1 Defects

RAM ICs are subject to the same physical defects encountered in logic ICs, which were described in Chapter 2. The defects include the presence and absence of material and of imperfection in the manufacturing process. Some defects, such as gate oxide breakdown, are more likely to cause failure in RAMs than other defects. Also, RAMs, particularly DRAMs, seem to be more susceptible than other ICs to some interference noise and disturbances. For example, because of long, parallel access and sensing wires, there is a good chance for crosstalk to occur [Yang 1999]. In addition, the memory cells are subject to single-event upsets (SEUs), which are induced by charged particles penetrating an electronic circuit. Such upsets occur not only in space but may be triggered by particle emanating from the metal used in fabricating the circuit.

12.3.2 Array Fault Models

Detection of the faults depends on where the fault is located in the various parts of the RAM circuitry. It is important to distinguish between the faults in the memory cell array and those in the surrounding circuitry. The latter consist of the decoding circuit, the write drivers, the sense amplifiers, and the registers. Although it seems that the surrounding circuitry can be handled as any random logic, the only way to observe its response to a test is through the RAM cells. Thus testing the array should also include testing the surrounding circuitry.

It might seem plausible to use for RAM cells the same structural fault models as those used for random logic. However, memory arrays are far denser than random logic. Using structural SAF or bridging faults would not be practical because of the extremely large volume of test data. Also, testing time would be excessively long. Because of these problems, a functional fault model is more appropriate for RAM arrays. A properly functioning RAM is capable of storing 1's and 0's in every cell, having each cell transition from 1 to 0, and vice versa, and having each cell return to its value after reading. In addition, the cells are expected to retain the information. A functional test that is capable of detecting all faults in these operations, while possible, is hardly practical since it would be on the order of 2^N , for an N -cell memory [Hayes 1975]. To make the test length manageable, the set of fault types is restricted.

The set of functional memory fault models used include stuck-at and bridging faults. However, these models are not sufficient to represent all types of failures in memory circuits. The sets also include faults that are very special to memory circuits and architecture. These are retention faults, transition faults, coupling faults, and pattern-sensitive faults. All the faults that we describe in the rest of this section are functional faults and not structural faults.

12.3.2.1 Stuck-at Faults. We are familiar with gate-level stuck-at faults (SAFs) from Chapter 2. For memory cells, a stuck-at fault is defined as a functional fault that makes the cells behave as if they have a 1 or a 0 stored permanently in them. This is illustrated in Fig. 12.4, where the cell is modeled as a flip-flop that is in either a state of 0 or a state of 1. To detect these

functional SAF, it is necessary to write a 1 (and a 0) on the cell and read the result of the operation.

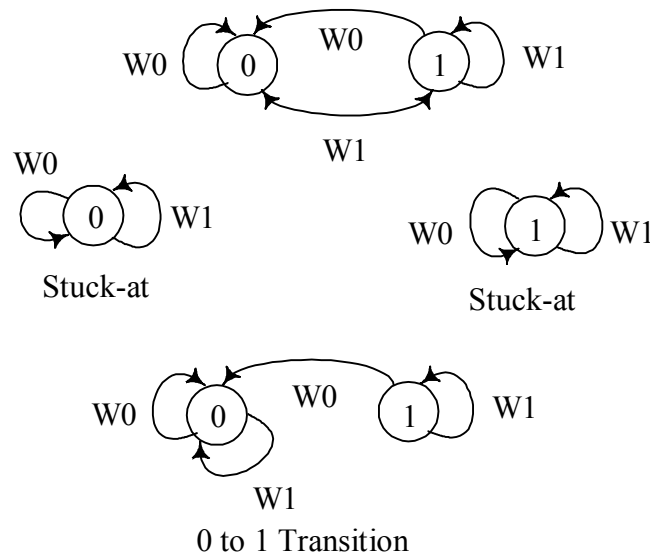


Figure 12.4 Fault Representations: a) SAF, b) Transition Fault.

This needs to be performed for all cells in the array of size N . The process can be described in an algorithmic form as follows: For all cells $C(j)$:

1. Write 0 on cell C_j ($C_j \leftarrow 0$).
2. Read C_j .
3. Write 1 on cell C_j ($C_j \leftarrow 1$).
4. Read C_j .

The test then requires accessing each cell C_j four times – two write and two read operations. If the access time is τ_a , then testing time is $4N\tau_a$, where N is the number of all cells in the array. Thus the complexity of the test is $O(n)$. This testing approach will also allow detection of multiple stuck-at faults (MSAs).

12.3.2.2 Transition Faults. This is a special case of stuck-at fault. A transition fault (TF) occurs when one of the cells cannot make the transition 0 to 1 (\uparrow) or 1 to 0 (\downarrow). The up-and down-arrow notations are due to van de Goor [van de Goor 1998]. A cell may have one type and not the other. Thus a cell may function properly from 0 to 1, and when it undergoes a 1-to-0

transition, the cell may remain at state 1 exhibiting a stuck-at-1 value. To detect these faults, it is important to make every cell undergo 0-to-1 and 1-to-0 transitions and to check the results.

12.3.2.3 Coupling Faults. Because of the regularity of its structure, a memory chip may experience a change in one cell due to an intended change in another cell. This is known as a coupling fault (CF). The coupling is due to shorts or parasitic effects such as stray capacitance.

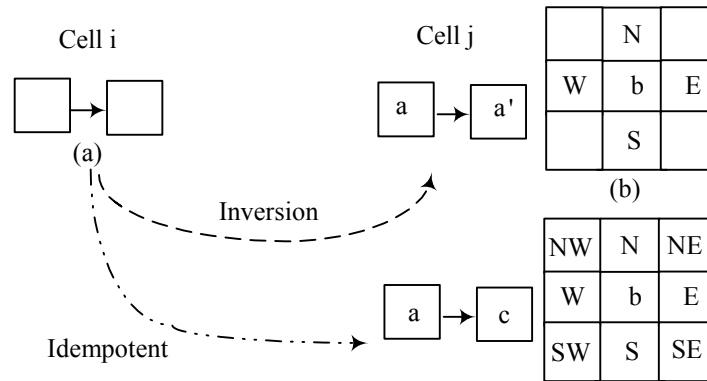


Figure 12.5 Fault Representations: a) Coupling Fault, b) NPSF Fault.

There are three CF types: inversion, idempotent, and bridging and state. *Inversion coupling* (CF_{ins}) occurs when one cell's transition causes inversion of another cell's value. That is, a 0-to-1 transition in cell i causes cell j to go from its logic value, a , to its complement, a' . An *idempotent* fault (CF_{ids}) occurs when a transition on cell i causes cell j to be in a particular logic value, 0 or 1. These faults are illustrated in Fig. 12.5.

The two-cell coupling used to define the faults is just a special case of k -way coupling, which can become too complicated because of the many ways these cells may interact. Here we concentrate on two-cell coupling. Techniques for detecting CF_{ids} can also detect CF_{ins} . We examine next how idempotent coupling can be detected. Assume that as cell C_i changes state it will affect cell C_j . We can detect the fault by performing the following operations:

1. Write 0 on all cells
2. For all $C_j, j \neq i$:

- 2.1 Change C_i .
- 2.2 Read C_j to check if it has changed.
- 2.3 Restore the value of C_i .

Also, they have to be repeated with the array initialized to 1. The time taken would then be

$$2[N + 3(N - 1)N] = 6N^2 - 4N$$

Bridging and state coupling faults (SCFs) are a special type of coupling faults. They occur when a certain state in one cell causes another specific state in another cell. They are caused by logic coupling rather than by transition. A bridging fault (BF) occurs when two or more lines are inadvertently shorted together. The shorted lines behave as if they are ANDed or ORed together, depending on the strengths of the drivers and the load transistors. Some of these faults may be detectable by stuck-at test patterns, but this is not always the case [Millman 1990]. Test algorithms have been developed for locating a bridging fault of multiplicity k [van de Goor 1991]. For a definition of multiplicity, see Section 2.7. An effective technique for detecting bridging faults is based on current testing (I_{DDQ}) monitoring. However, as the technology scales down to very deep submicron, current testing may lose its advantage, as we mentioned in Section 7.10.

12.3.2.4 Pattern-Sensitive Faults. Another way a memory cell fails is due to the different activities of other cells in the array, leading to pattern-sensitive faults (PSFs) [Brown 1972]. These faults are caused primarily by unwanted interference between the cells due to the high packed density of the cells. It is not possible to examine the effects on each other of all cells in the array, since this would require a test set of length $(3N^2 + 2N) \cdot 2^N$ [Hayes 1975]. It is probably more practical to examine the fault due to cells in the proximity, the neighborhood, of the one under test. These are called neighborhood PSFs (NPSFs). The definition of a neighborhood is centered on a *base cell*. The neighbors may be identified as E, W, N, and S, as shown in Fig. 12.5b and it is a five-cell neighborhood. It may also be extended to a nine-cell neighborhood, as illustrated in Fig. 12.5c.

NPSFs are classified into three categories: *active*, *passive*, and *static*. Given a certain pattern change in neighboring cells, the base cell may (1) change value (active fault), (2) remain

at a fixed value regardless of the attempts to change it (passive fault), or (3) change to a specific value (static fault). To detect an *active* NPSF, it is important to know the state of the base cell, to change a neighborhood cell, and to read the base cell. For a neighborhood of k cells, this will require 2^k changes, and for each change the other cells, $k - 1$, assume all possible values. This is a total of $(k - 1) \cdot 2^k$ [Suk 1980]. In the case of a *passive* fault, it is important to find out if for all possible patterns in the neighbor cells, the base cell will or will not change. Thus we perform two transitions for the base cell for all possible 2^{k-1} neighborhood patterns. In all, 2^k are required. In a similar fashion, it can be shown that 2^k patterns are needed for static NPSF faults. For all NPSF faults, a total of $(k + 1) \cdot 2^k$ patterns per neighborhood.

Detecting NPSF faults requires the exercising of the memory in such a manner as to detect other fault models. The NPSF model therefore covers many other memory fault models. However, due to the fact that these faults are not common and require very high complexity test sets, PSFs are not usually tested in today's memories.

12.3.3 Surrounding Logic

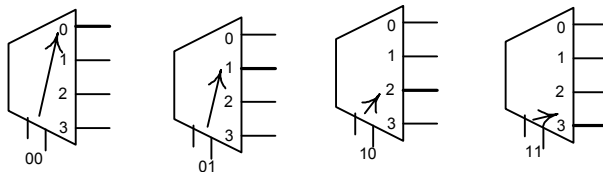


Figure 12.6 Testing the Decoding Circuitry

In addition to the array, the surrounding logic also has to be tested. A functional fault is also used in the address decoding circuit. An n -to- k decoder is exercised by making all possible selections and ensuring that a 0 and a 1 are appearing in the corresponding output. For example, for a 2-to-4 decoder, all four outputs are checked for correct operation by applying an exhaustive test on the select lines, as shown in Fig. 12.6. If any of the outputs is stuck at 1, the decoder will work correctly in one case and select the corresponding bit; otherwise, for the other cases, two bits will be selected simultaneously. However, if any output is stuck at 0, the corresponding bit will never

be selected. With this knowledge, it is possible to find out if the decoder is functioning correctly or by monitoring write and read operations on the RAM cells. The input, too, can be faulty, and this will result in selecting only half of the words. For example, if the first select line is SA0, it will read the first and second words correctly, but instead of the third and fourth, it will repeat again words 1 and 2. This pattern can then be traced to the data written and read from the array cells. Having one of the input SA0 (or SA1) is equivalent to having the MAR's cell corresponding to this select line SA0 (SA1). All these faults are detectable by patterns that are generated to test the RAM array.

12.4 Types of Memory Testing

As for any other integrated circuit, memory circuits are subjected to *parametric tests* (DC and AC), *functional tests*, *dynamic tests*, and I_{DDQ} tests. Memory functional testing, which involves generation of test patterns and their application using ATE, may be performed on the IC level, array level, or board level. However, to expedite testing, more and more built-in self-testing is used, as described in a Section 12.6. Parametric testing involves measurement of voltage, current, and frequency. Recently, I_{DDQ} testing has augmented it. Sometimes, the circuit might be performing its intended function but is not capable of driving a bus. This circuit will fail a parametric test and will immediately be considered defective. Also, a characterization testing determines its operating limits. Parametric testing is used (1) to verify that the product meets the specifications and (2) to characterize the product. The first provides the operating points of the circuit in normal operation mode.

12.4.1 Specification Testing

Memories are designed according to specific values of the supply current, the output voltage range, and seek time. The fabricated product has to conform to these specifications. However, conforming to these specifications does not guarantee that the product will actually work since

there might be changes in the ambient temperature or humidity, the power supply, or loading conditions. Both DC and AC parametric testing are usually performed.

12.4.2 Characterization Testing

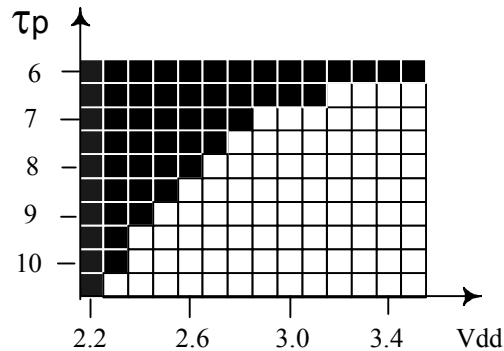


Figure 12.7 Characterization Testing: Shmoo Plot for V_{DD} vs. τ_p [Poulton 1997]

Characterization testing involves a repetitive sequence of measurements used to locate the operating limits of the circuit. For example, the Shmoo plot in Fig. 12.7 distinguishes, in the V_{dd} - τ_p space, the region in which the circuit operates correctly, as indicated by white tiles. The black tiles specify incorrect operation.

12.4.3 Functional Testing

The main task in functional testing is to generate test patterns that can detect the types of faults discussed in previous sections of this chapter. Since RAMs have many bits, the complexity of the algorithm generating the test needs to be evaluated for efficient test pattern generation and effective fault coverage. Here, application of the patterns consists of writing and reading from the cells.

As we have learned from the onset, detecting a stuck-at fault requires that we place on the faulty node the complement of the logic value at which the node is stuck. For all but the memory cell, the test pattern generation is similar to that discussed in Chapters 1, 2, 6, and 7. For the memory cells, however, we need to write a 0 (and 1) on each cell and check to determine if a 0

(and 1) is actually written by reading it. As a result of applying these test sets to the array, we may also detect other faults on the auxiliary circuitry such as the decoders. Consider, for example, the case where one of the decoder outputs experiences a stuck-at fault that results in inhibiting the accessing of the corresponding cell (or cells). It will not be possible to write a 0 and a 1 on the cell and verify the result of the writing. The stuck-at fault on the decoder will then be detected as part of the functional test of the array.

For transition faults, a test must transition each cell from 0 to 1 and then read it immediately. The test must also be repeated for transition from 1 to 0. Detecting coupling faults is more involved since it necessitates the testing of neighboring cells in a particular order. Thus to sensitize and detect coupling faults, we must write on cell i and then read cell j . This two-step operation must be done in:

1. Ascending order since any number of cells with lower address may be affected,
2. Descending order since any number of cells with higher addresses may be affected

NPSF faults are more complex and require a variety of methods. Test algorithms are available to perform such tests, but they are not as efficient for high fault coverage in a short testing time [Suk 1981, Saluja 1985, Sharma 1997]. Instead of applying a test set to detect every type of fault independently of the others, it is possible to devise one test set that detects several of these fault models, as described in Section 12.5.

12.4.4 Current Testing

Current testing discussed in Chapter 7 has been shown to be very useful in screening out potential failure of RAMs. The quiescent current for a normally functioning RAM may be on the order of tens of nanoamperes, while some failure may elevate this current several orders of magnitude [Hawkins 1985]. In a study, 1582 devices failed current testing only 1490 also failed functional testing. This implies that the 92 devices escaped functional testing [Meershoek 1990]. While current testing is applicable to present technology-feature SRAMs, it is not recommended for DRAMs.

12.5 Functional Testing Schemes

There are several test algorithms for RAMs [van de Goor 1990]. They are categorized into two classes: The first class comprises tests that require reading each cell after a specific change has been performed to other cells. For the other class of tests, only the cell that changed value is read. Commonly used algorithms are given next. Some of these algorithms are presented for historical and academic reasons. However, only those that are mostly effective are used for actual test application in production testing. An effective test is a test that detects as many faults as possible with the fewer possible patterns. The length of the test is measured in terms of the size N of an $N \times I$ memory array. Of course, if the memory is arranged in $r \times c$ array, where c is the word width and $N = rc$, it is possible to read or write in all bits of the word simultaneously and shorten the test length by r .

12.5.1 MSCAN

The memory scan (MSCAN) sequence is sometimes also referred to as a solid *pattern*. It consists of writing an all-0's pattern (and all-1's pattern), then reading all the cells. In addition to SAF testing, it serves in finding worst-case power dissipation. It also serves as a preparation for the application of other test sequences. Testing time is $T = 4N$; that is, the complexity is linear in the memory size.

12.5.2 GALPAT Algorithm

The GALloping PATtern algorithm is also known as the walking 1(0) or Ping-Pong test [Barracough 1976]. The algorithm consists of the following steps:

1. A solid all 0 (1) pattern is written in all N cells.
2. For each cell $C(j)$:
 - 2.1 Complement $C(j)$.
 - 2.2 Repeat the following sequence for $k \neq j$
 - 2.2.1. Read $C(j)$.
 - 2.2.2. Read $C(k)$.

2.3. Restore the content $C(j)$.

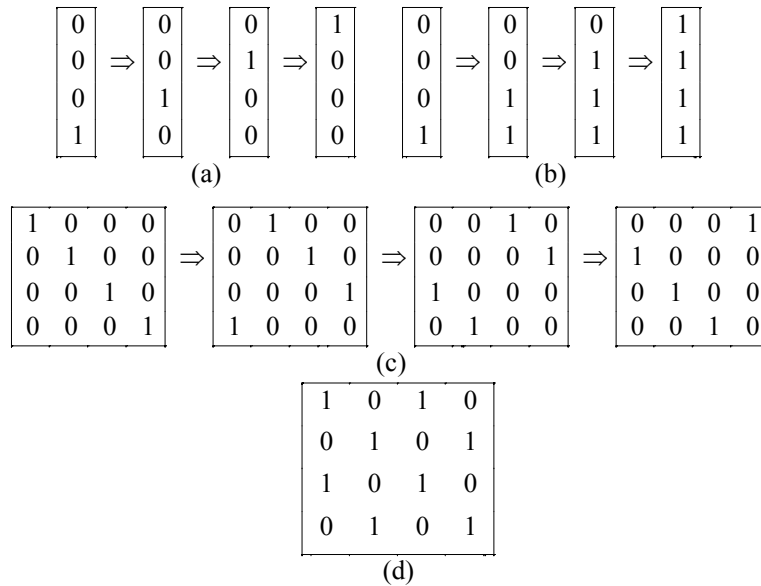


Figure 12.8 Functional Test Algorithms: a) GALPAT, b) Walking 1, c) GALDIA, d) Checker

This process looks like a sliding 1 is covering the array. As illustrated in Fig. 12.8a, step 1 requires N write operations. For each iteration of step 2, the target cell $C(j)$ is complemented twice, which is equivalent to two write operations. There are $2(n - 1)$ read operations. Thus the total number of operations is $N + N[2 + 2(N - 1)] = N + N(2N + 1)$. Since the test will be repeated with the array initialized to 1, the total test time is $T = 2(2N^2 + N)$. A variation of this sequence that omits step 2.2.2, known as the walking 0/1, is illustrated in Fig. 12.8b. Testing time is still $O(N^2)$. This test detects:

- Stuck-at faults, since each cell is read once when it has value 0 as well as value 1
- Transition faults on the cells, since each cell is switched from 1 to 0 and from 0 to 1
- Decoding circuits, as only one cell at a time contains logic value 1
- Some state coupling faults since any two arbitrary cells under states 00, 01, and 10 are read.

There are different test sequences that are based on GALPAT but are less complex than this algorithm. Although in the GALPAT only one cell has 1 at a time, all the cells in an entire

row, column or diagonal can have 1, and this configuration of 1's gallop through the RAM array. The most widely used is the GALDIA. First, the algorithm places 1's in all cells on the diagonal, that is, cells whose row and bit addresses are the same. On all other cells, 0's are placed. The cells are then read in an ascending order. The test is repeated with 0's in the diagonal. The test is illustrated in Fig. 12.8c for shifting 1's on the diagonal.

12.5.3 Algorithmic Test Sequence

The algorithmic test sequence (ATS) is one of the first test sets that were developed for RAMs [Knaizuk 1977]. The patterns are applied to three partitions of the memory array. Each partition corresponds to addresses Mod 3. Thus the cells of a partition did not occupy adjacent locations.

The three partitions are Π_0 , Π_1 , and Π_2 .

1. Write 0 in all cells of Π_1 and Π_2
2. Write 1 In all cells of Π_0
3. Read 1 from Π_1 if 0, then no faults, else, faulty
4. Write 1 in all cells of Π_1
5. Read 1 from all cells of Π_2 if 1, then no faults, else faulty
6. Read 1 from Π_0 and Π_1 if 1, then no faults, else faulty
7. Write 0 in, then read 0 from Π_0 if 0, then no faults, else, faulty
8. Write 1 in, then read 1 from Π_2 if 1, then no faults, else, faulty

This test sequence requires $4N$ operations. It detects SAF faults on the memory cells, decoders, and MDR and MAR registers. The algorithm has subsequently been improved and resulted in the MATS algorithm [Nair 1979] and an MATS+ [Abadir 1983]. These test sequences and other variations given in the next sections improve the complexity and permit the detection of more faults than with the GALPAT algorithm.

12.5.4 Marching Pattern Sequences

There is a class of test sequences called March tests [van de Goor 1998]. A special version of the walking 0/1, given in Section 12.5.1, minimizes the excessive reading operations. After initialization, each cell is read, then complemented. As the last cell is reached, the RAM array is filled with 1's. The process is repeated but in the reverse addressing order; that is, the process of reading and complementing starts with the last cell examined. This early example of a March test is due to [Cocking 1975] and was used to test memory in microprocessors [Barraclough 1976].

The algorithm requires only 10 read/write operations per cell. It is $O(N)$. The test sequences developed by [Nair 1978], March A and March B, have complexity of $30N$. Improvements on this version were developed by [Suk 1981, Marinescu 1982, and van de Goor 1998].

The March C test, a solid 0 pattern is first placed in the RAM. The contents are then read in ascending order. A check is made to assert that each cell C_j content is 0; then a 1 is written in the location. The process is repeated until 1's are written on all cells. The cells are then read in a descending order to verify that they contain 1's. The 1's are then changed to 0's. The same test is repeated but starting from a solid 1 pattern.

1. Starting from the lowest to highest address, $j = 1$ to N :
 - 1.1. Write 0 in all $C(j)$.
 - 1.2. Read $C(j)$, write 1 in $C(j)$.
 - 1.3. Read $C(j)$, write 0 in $C(j)$.
 - 1.4. Write 0.
2. Starting from the highest to the lowest address, $j = N$ down to 1.
 - 2.1. Read $C(j)$, write 1 in $C(j)$.
 - 2.2. Read $C(j)$, write 0 in $C(j)$.
 - 2.3. Read $C(j)$.

This sequence requires 11 read/write operations, and therefore testing time is $T = 11N$. It detects faults on decoders, transition faults (since it verifies that the cells can undergo transitions from 1 to 0, and vice versa), and some coupling faults.

There are several variations of this algorithm that either improve or reduce the timing sequence, or detect more faults. For example, the March C- test reduces the operations to 10 per cell by removing step 2.3 while detecting the same faults [van de Goor 1998]. March C+ adds a read operation after each write operation in the March C-. Testing time is then increased to $14N$. However, it detects, in addition to the faults detected by March C, stuck-open faults and some timing faults. It is also possible to eliminate the last two steps of March C-test reducing the test complexity to $10N$, while detecting the same faults.

12.5.5 Checkerboard Test.

This test places each cell in a state different from its immediate neighboring cells, as depicted in Fig. 12.8d. The bits are thus partitioned into two sets; those in state 0 form partition Π_0 and those

in state 1 form partition Π_1 . The algorithm writes and reads one set, then the other. The test is then repeated from the alternative logic values. This test detects and locates stuck-at faults and shorts between adjacent cells assuming that the decoding subcircuit is fully tested.

Algorithmically, it can be expressed as:

1. Write 0(1) in cells of Π_0 (Π_1).
2. Read all cells.
3. Write 1(0) in cells of Π_0 (Π_1).
4. Read all cells.

Table 12.1 Algorithms and their Complexity

Algorithm	Complexity	Detectable Faults*							
		1	2	3	4	5	6	7	8
MSCAN	$4N$		√						
GALPAT	$2(2N^2 + N)$	√	√	√	√		√		
Marching0/1	$2(N^2 + N)$		√	√	√				
GALDIA	$(2N^{1/2} + 4)N + 5N^{1/2}$		√	√	√			√	
GALCOL	$3N^{3/2} + 6$		√	√					√
March A	$30N$	√	√	√	√				
March B	$16N$	√	√	√	√				
March C	$11N$	√	√	√	√				
March C-	$10N$	√	√	√	√				
March C+	$14N$	√	√	√	√	√			
Checker	$N + 32N\log_2N$		√	√	√				

*1. Address faults, 2. SAF, 3. Transition Faults, 4. Coupling Faults, 5. Timing Faults, 6. Faulty address transitions between Each cell and every other, 7. Faulty address transitions between Each cell and every other, 8. Faulty address transition between each cell and the cell row

The test detects (1) all stuck-at faults, (2) data retention faults, and (3) 50% of transition faults.

Notice that address decoder faults and state coupling faults are not covered. The algorithm requires $4N$ read/write operations. Their complexity and effectiveness in fault detection are summarized in Table 12.1, where fault models 1,2, 3, ... refer to address faults, SAF on the cells, TF, CF, stuck-open, and timing, respectively. No one single of these algorithms is capable of detecting all fault models. A more comprehensive set of algorithms can be found in [Sharma 1997 and van der Goor 1998].

12.6 Memory BIST

Memory ICs have many features that make them easily testable: regular memory array structure, shallow sequential depth, and accessible I/O. For embedded memories, discussed in a later section, the controllability and observability are reduced. Another characteristic of memory testing is the excessive length of the tests. For example, the simple sliding diagonal algorithm requires hours for a 1-Mbit chip. The GALPAT requires a testing time of order $O(N^2)$. Although this takes only a few seconds for a 1-Kbit, it may take days for a 1-Mbit chip. This complexity of algorithm is compounded by the fact that usually, external testing is not run at speed. Moreover, with increasing size of memories, the volume of test data is becoming too large for efficient handling through automatic test equipment (ATE). An attractive solution to this bottleneck problem is the use of built-in self-testing (BIST).

In the early 1980s, a built-in testing scheme consisted of building address and data generators and an output comparator on the chip and to leave the flexibility of defining the patterns and timings to outside the chip. The test patterns are controlled from test pins and run at speed. In this fashion, only few extra pins are needed. Using transitional LFSRs for address and data generation requires a few modifications of the RAM circuitry but may not guarantee full fault coverage, due to aliasing as explained in Chapter 11. The aliasing may be reduced with special added test structure [Zorian 1990]. However, it is possible to make use of the regularity of RAM architecture and the simple configuration of the test patterns to obtain higher fault coverage. For example, to implement the ATS algorithm, we need to addresses Mod 3 [Bardell 1988]. In this case we can configure the MAR to function at testing time as a Mod 3 counter. The data applied for writing, the MDR is also configured to generate 0's or 1's according to the test routine requirement. A controller is then needed to coordinate these operations—configuring the registers for test mode and applying the patterns according to the algorithm.

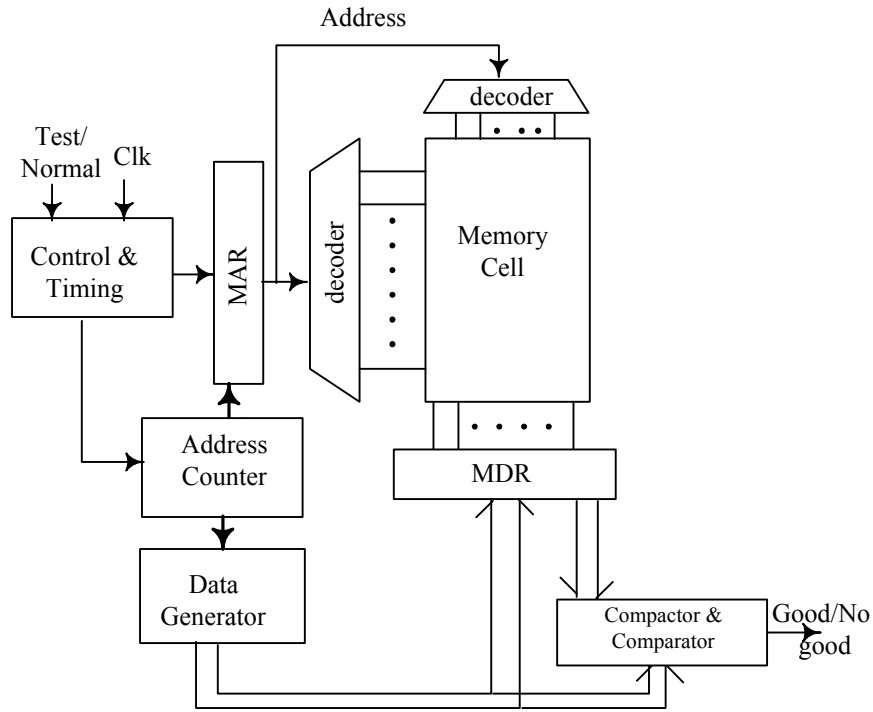


Figure 12.9 RAM BIST

Figure 12.9 shows the block diagram for the RAM configuration with BIST. In testing mode, the controller will be in one of the states listed in Table 12.2 for the application of the ATS sequence,, given in Section 12.5.3. If several RAMs are placed on the same chip, they may share the BIST circuitry. The controller would also need to be modified to select the chip [Nadeau-Dostie 1990]. BIST has also been demonstrated to be successful with DRAMs [Ohsawa 1987], FIFO [Zorian 1994], ROMs [Zorian 1992], and other types of memory.

Table 12.2. Applying ATS in BIST Configuration

Stat	R/W	Data	Partition	ATS step
1	W	0	Π_1	1
2	W	0	Π_2	1
3	W	1	Π_0	2
4	R	0	Π_1	3
5	W	1	Π_1	4
6	R	1	Π_2	5
7	R	1	Π_0	6

8	R	1	Π_1	6
9	W	0	Π_0	7
10	R	0	Π_0	7
11	W	0	Π_2	8
12	R	0	Π_2	8

12.7 Memory Diagnosis and Repairs

It has been a practice for stand-alone memory to include spare rows and columns of cells to replace possible faulty ones. Typically, a fuse-blow process using external laser equipment is required in order to swap faulty rows or columns with the spare ones. These repairs are also needed for embedded memories because ICs embed large and dense memories storing from 256 Kb to 256 Mb. It is necessary to determine the faulty cells in the embedded memory arrays and to reconfigure them. However, the amount of test data to be transferred in and out of the chip is too large. To minimize the high bandwidth interaction with the outside built-in self test and redundancy may be used and hence only the data that include information about the repairs are relayed to the tester. A more advanced solution is to include online the test and repairs resources for the embedded memories. In this case, the embedded memory test goes beyond fault detection to include failed-bit diagnosis, redundancy analysis, and self-repair.

References

- Abadir, M. S. and H. K. Reghabati (1983), Functional testing of semiconductor random access memories, *Comput. Surv.*, Vol. 15, No. 3, pp. 175 – 98.
- Bardell, P. H. and W. H. McAnney (1988), Built-in test for RAMs, *IEEE Des. Test Comput.*, Vol. 5, No. 8, pp. 29 – 37.
- Barraclough, W., A. C. L. Chiang, and W. Sohl (1976), Technique for testing the microcomputer, *Proc. IEEE*, Vol. 64, No. 6, pp. 943 – 950.
- Brown, J. R. (1972), Pattern sensitivity in MOS memories, *Digital Symposium on Testing to Integrate Semiconductor Memories into Computer Mainframes*, pp. 33 – 46.
- Cocking, J. (1975), RAM test patterns and test strategy, *Digest Of Papers Semiconductor Test Symposium*, pp. 1 – 8.

- Hayes, J. P. (1975), Detection of pattern sensitive faults in random access memories, *IEEE Trans. Comput.* Vol. C-24, No. 1, pp. 50 – 157.
- Hawkins, C. and J. M. Soden (1985) Electrical characteristics and testing considerations for gate oxide shorts in CMOS ICs, *Proc. IEEE International Test Conference*, pp. 544 – 555.
- Knaizuk, J. Jr. and C. R. P. Hartmann (1977), An algorithm for testing random access memories, *IEEE Trans. Comput.*, Vol. C-26, No. 4, pp. 414 – 416.
- Marinescu, M. (1982), Simple efficient algorithms for functional TAM testing, *Proc. IEEE International Test Conference*, pp. 236 – 239.
- Meershoek, R. et al. (1990), Functional and I_{DDQ} testing in static RAMs, *Proc. IEEE International Test Conference*, pp. 929 – 937.
- Millman, S. D. and E. J. McCluskey (1990), Diagnosing CMOS bridging faults with stuck-at fault dictionaries, *Proc. IEEE International Test Conference*, pp. 860 – 870.
- Nadeau-Dostie B., A. Silburt, and V. K. Agrawal (1990), A serial interfacing technique for external and built-in self-testing of embedded memories, *IEEE Des. Test of Comput.*, Vol. 7, No. 2, pp. 56 – 64.
- Nair, R., S. M. Thatte, and J. A. Abraham (1978), Efficient algorithms for testing semiconductor Random-access Memories, *IEEE Trans. Comput.*, Vol. C-27, pp. 672 – 576.
- Nair, R., (1979), “Efficient algorithms for testing semiconductor Random-access Memories,” *IEEE Trans. Computers*, Vol. C-28, No. 6, pp. 672-576.
- Ohsawa, T. et al. (1987), A 60-ns 4-Mbit CMOS DRAM with built-in self-test function, *IEEE J. Solid State Circuits*, Vol. 23, No. 10, pp. 663 – 668.
- Poulton, J. (1997), An embeded DRAM for CMOS ASICs, *Proc. 17th Conference on Advanced Research in VLSI*, pp. 288 – 302.
- Rideout, V. L. (1989), One-device cells for dynamic random-access memories: tutorial, *IEEE Trans. on Electron Devices*, Vol. ED-26, No. 6., pp. 839 – 851.
- Saluja, K. K. and K. Kinoshita (1985), Testing for coupled cells in Random access memories, *Proc. IEEE International Test Conference*, pp. 439 – 451.
- Sharma, A. K., (1997), *Semiconductor Memories: Technology, Testing, and Reliability*. IEEE Press, Piscataway, NJ.
- Suk, D. S. and S. M. Reddy (1980) Test procedures for a class of pattern sensitive faults in semiconductor random access memories, *IEEE Trans. Comput.*, Vol. C-29, No. 6, pp. 419 – 429.
- Suk, D. S. and S. M. Reddy (1981), A March Test for functional faults in semiconductor random access memories, *IEEE Trans. Comput.*, Vol. C-30, No. 7, pp. 982 – 985.
- Tsuruda, T. et al. (1997), High Speed/High-bandwidth design methodologies of on chip DRAM cores multimedia system LSI's, *IEEE J. Solid State Circuits*, Vol. 32, No. 3, pp. 477 – 482.

van de Goor, A. J. and C. A. Verruijt (1990), An overview of deterministic functional RAM chip testing, *ACM Computing Surveys*, Vol. 22, No. 1, pp. 5–33.

van de Goor, A. J. et al. (1991), Locating bridging faults in memory arrays, *Proc IEEE International Test Conference*, pp. 685–694.

van de Goor, A. J. (1998), *Testing Semiconductor Memories: Theory and Practice*, COMPTEx Publishing, Gouda, The Netherlands.

Watanabe, S. et al. (1995) A novel circuit technology with surrounding gate transistors for ultra high density DRAMs, *IEEE J. Solid State Circuits*, Vol. 30, No. 9, pp. 960–970.

Yang, Z, and S. Mourad (1999), Deep submicron on-chip crosstalk, *Proc. IEEE International Conference Measurement and Technology*, pp. 1788–1793.

Zorian, Y. (1990), A structured approach to macrocell testing using BIST, *Proc. IEEE Custom Integrated Circuit Conference*, pp. 28.3.1–28.3.4.

Zorian, Y. and A. Ivanov (1992), An Effective BIST scheme for ROM's, *Trans. IEEE Comput.*, Vol. 41, No. 5, pp. 646–653.

Zorian, Y. (1999), Testing the monster chip, *IEEE Spectrum*, Vol. 36, No. 7., pp. 54–60.

Problems

- 12.1.** List the various fault models used in memory testing. Consider a 3 x 3 SRAM and develop test patterns to detect each of these faults.
- 12.2.** Consider one cell of SRAM implemented as shown in Fig. 12.3*b*. Use the test patterns generated in Problem 12.1 for stuck-at faults on the cell and determine which SAF of the circuit are detected by the test.
- 12.3.** Find the complexity of the following algorithm
- ```
For $j = 1$ to N ;
 Write 0 in $C(j)$
 Read cell $C(j)$
End
For $j = 1$ to $N/2$
 Write 1 in $C(j)$
 Read $C(j)$
End For
For $j = 1$ to N ;
```

```
 Read cell $C(j)$
 End For
 For $j = N/2$ to 1
 Write 0 in $C(j)$
 Read $C(j)$
 End For
```

- 12.4.** Does the algorithm of Problem 12.3 detect decoder faults? Explain.
- 12.5.** Develop the test sequence for a marching 1/0 algorithm for a 32-bit RAM implemented in an 8 x 4 array.
- 12.6.** Compute the test time required to apply (a) a March test, then (b) a GALPAT test to a 32k RAM given that it takes 150 ns to apply each pattern.