Using March Tests to Test SRAMs

AD J. VAN DE GOOR
Delft University of Technology

Many fault models for SRAMs and tests for faults of these models are available. This article gives a unified notation for these fault models and tests, and in addition the author shows the likelihood that the different types of faults will occur. Then the author discusses a set of march tests together with methods to make composite tests for collections of fault types. Empirical results showing the fault coverage of the different tests enable SRAM users to choose the fault models of interest and the test.

TESTING SEMICONDUCTOR memories is increasingly important because of the high density of current memory chips (now 16 megabits and more), and also because older algorithms take so long to complete their testing. For instance, Galpat and Walking I/O[1,2] require test times on the order of n^2 or $n^{3/2}$ (where n is the number of bits in the chip). At that rate, assuming a cycle time of 100 ns, testing a 16-Mbit chip would require 500 hours for an n^2 test and 860 seconds for an order $n^{3/2}$ test. Other older tests, such as Zero-One and Checkerboard,[1,2] are of order n, but they have poor fault coverage.

To investigate better methods for testing semiconductor memories, I first survey march tests, which are of order n, and then present a systematic way to extend those tests. Next, I introduce functional faults and describe a notation for functional faults for SRAMs. Using inductive fault analysis and physical defect analysis to demonstrate the likelihood of the proposed functional faults occurring, I go on to suggest the use of march tests. In describing such tests together with a framework to make composite tests for the fault models that are important to a particular user, I also furnish proofs of their correctness and a unified notation. Finally, I present empirical evidence to support the effectiveness of the fault coverage of these tests.

The following discussion is directed toward users of memory devices, rather than their manufacturers. Because they have better insight into the failure modes of their chips, manufacturers may be able to use shorter, more efficient tests; not so simple memory device users. Space limitations preclude discussion of parametric and dynamic tests.

Fault models

The functional model of an SRAM chip, which can often be found in the manufacturer's data sheets, consists of many blocks. Though each of the blocks of the sample model shown in Figure 1 represents a particular function and may become defective, faults in certain blocks show the same fault behavior. For fault modeling purposes, the functional model then may be simplified to the reduced functional model of Figure 2. This model includes the address decoder (blocks A, B, and C of Figure 1), the memory cell array, and the read/write logic (blocks E, F, and G of Figure 1).

0740-7475/93/0300-0008$03.00 © 1993 IEEE

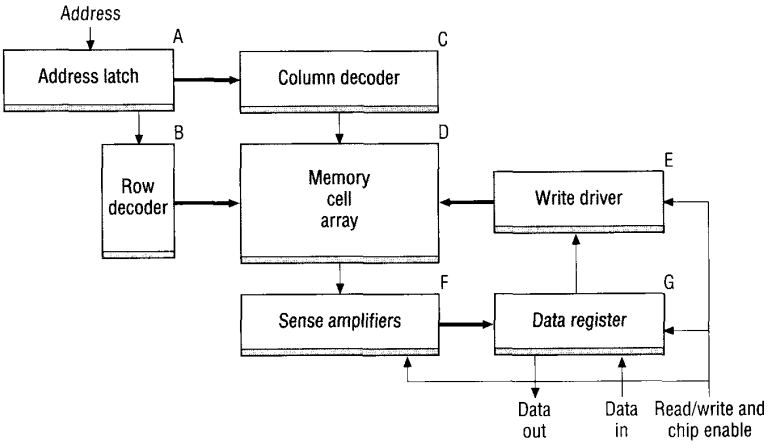


Figure 1. *Functional model of an SRAM chip.*

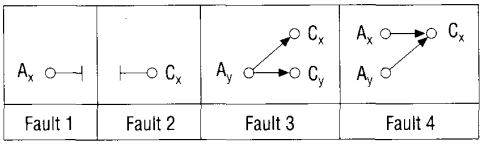


Figure 3. *Address decoder faults.*

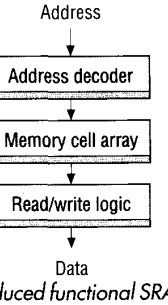


Figure 2. *Reduced functional SRAM chip model.*

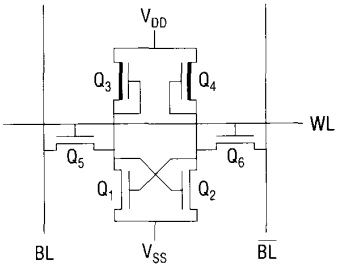


Figure 4. *Six-device SRAM cell.*

Faults in the address decoder. Address decoder faults (AFs) are faults in the address decoder.[3,4] We assume that AFs do not change the decoder into sequential logic and will be the same during read and write operations. Figure 3 shows the functional faults that can occur in the address decoder. They are:

- *Fault 1.* With a certain address, no cell will be accessed.
- *Fault 2.* A certain cell will not be accessible.
- *Fault 3.* With a certain address, multiple cells are accessed simultaneously.
- *Fault 4:* A certain cell can be accessed with multiple addresses.

Faults in the memory cell array. Many different faults can occur in a memory cell array.[2,5] These can be classified as faults which involve only a single cell (such as stuck-at, stuck-open, transition, and data retention faults) and faults whereby a cell or group of cells influences the behavior of another cell. The latter class is called coupling faults (CFs). CFs can be divided into inversion, idempotent, and state coupling faults. Also, CFs may be linked.

The following notation will help to describe the faults just mentioned:

x	denotes that a cell or line is in logical state x; $x \in \{0,1\}$
L	denotes that the faulty value is the value of the last read operation
$\uparrow$	denotes a write 1 'w1' operation to a cell containing a 0
$\downarrow$	denotes a 'w0' operation to a cell containing a 1
$\updownarrow$	denotes a 'w$\bar{x}$' operation to a cell containing an x
$\forall$	denotes any operation
<....>	denotes a particular fault
<S/F>	denotes a fault in a single cell
S	describes the value/operation sensitizing the fault; $S \in \{0,1,\uparrow,\downarrow,\updownarrow\}$
S_T	says that the sensitization effect appears after a time T
F	describes the faulty value of the cell; $F \in \{0, 1\}$
<$S_1,S_2, \ldots ,S_{m-1};F$>	denotes a fault involving m cells. $S_1,S_2, \ldots , S_{m-1}$ describes the conditions of the $m-1$ cells required to sensitize the fault in cell m (the faulty value is denoted by F); $S_i \in \{0,1,\uparrow,\downarrow,\updownarrow\}$ for $1 \leq i \leq m-1$.

In a *stuck-at fault* (SAF), the logic value of a stuck-at cell or line is always 0 (an SA0 fault) or always 1 (an SA1 fault). The notation <$\forall$/0> denotes an SA0 fault, and <$\forall$/1> denotes an SA1 fault.

A *stuck-open fault* (SOF) means that a cell cannot be accessed,[6] perhaps because of an open word line (WL). See Figure 4. When a read operation is per-

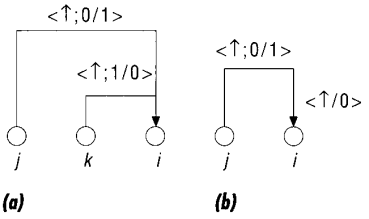


***Figure 5.** Examples of linked faults: two linked CFids (a) and a CFid linked with a TF(b).*

formed on a cell, the differential sense amplifier has to sense a voltage difference between the bit lines (BL and $\overline{BL}$) of that cell. In case of an SOF, both bit lines will have the same voltage level; consequently the output value produced by the sense amplifier (SA) depends on the way it is implemented:

- *Operation of the SA is transparent to SOFs.* When the SA has only a single input (it is implemented as a buffer rather than a differential amplifier), an SOF will produce a fixed output value (always a 0 or always a 1). The SOF will appear as an SAF and therefore is detectable.
- *Operation of the SA is nontransparent to SOFs.* To broaden the read window, the SA may contain a latch. Then a SOF may have the effect that the latch is not updated because the voltage difference between the bit lines is too small. The previous output value is produced as the output value for the SOF. The notation for this fault is <∀/L>.

In a *transition fault* (TF), a cell fails to undergo a $0 \rightarrow 1$ transition (a <↑/0> TF) or fails to undergo a $1 \rightarrow 0$ transition (a <↓/1> TF). Note that a single cell may exhibit a <↑/0> and/or a <↓/1> TF.

A *data retention fault* (DRF) occurs when a cell fails to retain its logical value after some period of time.[6] A DRF may be caused by a broken (open) pull-up device within a cell. Leakage currents then will cause the node with the broken pull-up device to lose its charge, causing a loss of information if a logic value was stored in the cell which required a high voltage at the open node. Two different DRFs can be recognized (both may be simultaneously present in a single cell): $<1_T/0>$ and $<0_T/1>$. When both are present in one cell, the cell behaves as if it contains an SOF (because there will not be a voltage difference on BL and $\overline{BL}$). Thus the test extensions for SOFs, in case the SA is nontransparent to SOFs, should be part of the test for DRFs.

An *inversion coupling fault* (CFin)[7-9] involves two cells *i* and *j*; the fault is sensitized by a transition write operation (that is, an ↑ or a ↓ write operation) to a particular cell *j*, Cell *j* is called the *coupling cell*, and inverts the contents of cell *i*, which is called the *coupled cell*. Two different CFins can be recognized: the <↑;↓> and the <↓;↑> CFins. Between a given pair of cells *i* and *j*, both faults may exist simultaneously.

An *idempotent coupling fault* (CFid)[7-9] involves two cells *i* and *j*. The fault is sensitized by a transition write operation to a cell *j*, which forces the contents of another cell *i* to a fixed value (0 or 1). Four different CFids can be recognized: <↑;1/0>, <↑;0/1>, <↓;1/0>, and <↓;0/1>.

A *state coupling fault* (CFst)[6] differs from the CFin and CFid because it is not sensitized by a transition write operation in the coupling cell but by some connection between two cells or lines. It is defined as follows: a coupled cell or line *i* is forced to a certain value *x* only if the coupling cell or line *j* is in a given state *y*. Between two cells or lines, four different CFsts can be distinguished: <0;0/1>, <0;1/0>, <1;0/1>, and <1;1/0>.

Linked faults[2,5,10] affect the same cell. In linked CFs, two or more CFs exist with the same coupled cell. Linked faults may occur between faults of the same type (see Figure 5a for two linked CFids) or between faults of different types (see Figure 5b, which shows a CFid linked with a TF). Unless special precautions are taken in a test, fault masking may occur with linked faults. In fault masking, the fault effect disappears because when sensitized by one fault it is canceled by another fault. For example, the march test {⇳(w0);⇑(r0,w1)} can detect the CFid <↑;0/1> of Figure 5a only when the CFid <↑;1/0> is not present.

Validity of fault models

Tests can detect the presence of functional faults, but they take time and therefore money. Currently, testing accounts for about half the cost of memory chips, so tests should only be performed to detect those faults which are reasonably likely to occur. The likelihood for a particular fault depends on the technology used, the feature width, the circuit design and layout, and the variations in the manufacturing process of a particular chip. This likelihood varies between chips of different manufacturers and even between chips manufactured in the same batch. We can use inductive fault analysis or physical defect analysis to determine this likelihood.

Inductive fault analysis. IFA[11,12] is a systematic procedure to predict all faults (defects) likely to occur in an integrated circuit. The effect of each defect can be translated into one or more of the functional faults. The IFA method consists of inserting physical defects into the layout of a chip. Two classes of defects can be distinguished:

- *Global defects* may be caused by a too thick gate oxide, a too thin polysilicon, mask misalignments, and so forth. They affect many chips on a wafer and are the main cause of dynamic faults.[6] Such faults are outside the scope of this article.
- *Local defects* (also called spot defects) are caused by extra, missing, or inappropriate material (for example, dust particles). A spot defect affects only a single chip and causes a functional fault.

Dekker[6] has investigated the effect of

spot defects on 16-Kbit SRAM chips, manufactured with a 1.5-μm technology. He analyzed the effect of spot defects of different sizes for the memory cell array of the 16-Kbit SRAM chip. The spot defects then were translated into electrical faults, which in turn were translated into functional fault (see Table 1).

From Table 1, which shows the effects of spot defects on the memory cell array (which is 80% of the chip area), we can conclude that SAFs contribute about 50% to the total number of faults. CFins were not found, while TFs and CFids only occur with larger spot sizes.

Physical defect analysis. Dekker[6] analyzed the physical defects in 1,192 defective devices from nine wafers, produced in three different batches, using a light microscope and a scanning electron micrograph. As shown in Table 2, the result shows a large number of unidentified faults. Also, TFs hardly occurred, while SAFs account for about 60% and SOFs account for about 14% of the faults. In addition, CFids, CFsts, and DRFs all did occur in practice. Note that the results of Table 1 (based on IFA) and of Table 2 (based on physical defect analysis) both show the occurrence of the same functional faults (except for TFs), and that SAFs and SOFs are the dominant fault types.

Concept of march tests

Many types of tests for SRAMs have been proposed in the past. Currently, one family of tests, called march tests,[8,9] has proven to be superior for test time and simplicity of the algorithms.

A *march test* consists of a sequence of march elements. A *march element* consists of a sequence of operations applied to each cell in the memory, before proceeding to the next cell. An operation can consist of writing a 0 into a cell (w0), writing a 1 into a cell (w1), reading a cell with expected value 0 (r0), and reading a cell with expected value 1 (r1). After all operations of a march element have been applied to a given cell, they will be applied to the next cell. The address of the next cell is determined by the *address order*. Two exist: an increasing address order, from address 0 to $n-1$ denoted by the $\Uparrow$ symbol; and a decreasing address order, from address $n-1$ down to 0, denoted by the $\Downarrow$ symbol. When the address order is irrelevant, the $\Updownarrow$ will be used. [Note that for the $\Uparrow$ address order any sequence may be used (for example., a pseudorandomly generated sequence), as long as the $\Downarrow$ address order uses the exact inverse address sequence (van de Goor[2]); n denotes the total number of addresses.]

The MATS+ march test[4] can be written as follows: $\{\Updownarrow(w0);\Uparrow(r0,w1);\Downarrow(r1,w0)\}$. It consists of three march elements, M_0 through M_2. March element M_1 uses the $\Uparrow$ address order and performs an 'r0', followed by a 'w1' operation on a cell before proceeding to the next cell.

Tests for SRAMs

To detect all functional faults within a chip, one should test the address decoder, the memory cell array, and the read/write logic (see Figure 2).

Tests for faults in the memory cell array will detect the same faults in the read/write logic, which means that no separate tests for the read/write logic are required. Also, tests will not be able to distinguish between faults in the memory cell array and the read/write logic, that is, faults can only be detected and not located.[5,8,9]

Similarly, AFs will be detected by march tests for the memory cell array if they satisfy the conditions of Table 3.[2,5] The table shows that the march test should consist of at least two march elements which have opposite address orders. Note that the march test will contain other march elements such as an initializing march element $\Updownarrow(w0)$', and so forth. The notation '...' in the march elements indicates the allowed presence of any number of operations. In addition, read operations can occur anywhere (because they do not disturb the state of the memory), and any number of march elements may be placed between the march elements of Conditions 1 and 2 of Table 3.

There are many march tests optimized for a particular set of functional faults.[2] The three most important march tests are MATS+, March C–, and March B.

MATS+. Equation 1 (in the box on page 12) shows the MATS+ algorithm.[4] It requires $5 \cdot n$ operations and detects all AFs because the conditions of Table 3 are satisfied (when $x=0$). In addition, all SAFs are detected because from each cell a 0 value is read (by the 'r0' operation of march element M_1) and a 1 value

Table 1. *Functional faults caused by spot defects.*

Fault class	Spot size <2 μm (%)	Spot size <9 μm (%)
SAF	51.3	49.8
SOF	21.0	11.9
TF	0.0	7.0
CFst	9.9	13.2
CFid	0.0	3.3
DRF	17.8	14.8
Total	100.0	100.0

Table 2. *Validation of fault models.*

Devices (%)	Fault
59.9	SAF*
14.1	SOF
1.5	CFid
0.8	CFst
2.2	DRF
21.5	?

**Or total device failure*

Table 3. *Conditions for detecting AFs.*

Condition	March element
1	$\Uparrow(rx, \ldots, w\bar{x})$
2	$\Downarrow(r\bar{x}, \ldots, wx)$

March test algorithms

(1) MATS+ algorithm; (2) March C– algorithm; (3) March B algorithm; (4) March test extended to detect DRFs; (5) March test extended to detect DRFs which behave as SOFs; (6) March C–test with inserted 'Del' elements; (7) March G algorithm; (8) Symmetric March G algorithm

$\{\Updownarrow(w0)\Uparrow(r0,w1)\Downarrow(r1,w0)\}$ (1)
$M_0 \quad M_1 \quad M_2$

$\{\Updownarrow(w0); \Uparrow(r0,w1); \Uparrow(r1,w0; \Downarrow(r0,w1); \Downarrow(r1,w0); \Updownarrow(r0)\}$ (2)
$M_0 \quad M_1 \quad M_2 \quad M_4 \quad M_5$

$\{\Updownarrow(w0); \Uparrow(r0,w1,r1,w0,w1); \Uparrow(r1,w0,w1); \Downarrow(r1,w0,w1,w0); \Downarrow(r0,w1,w0)\}$ (3)
$M_0 \quad M_1 \quad M_2 \quad M_3 \quad M_4$

{Existing march test;Del; $\Updownarrow(r0,w1)$; Del; $\Updownarrow(r1)$} (4)

{Existing march test;Del; $\Updownarrow(r0,w1,r1)$; Del; $\Updownarrow(r1,w0,r0)$} (5)

$\{\Updownarrow(w0);\Uparrow(r0,w1);\Uparrow(r1,w0); \Downarrow(r0,w1);$ Del:$\Downarrow(r1,w0);$ Del; $\Updownarrow(r0)\}$ (6)

$\{\Updownarrow(w0); \Uparrow(r0,w1,r1,w0,r0,w1); \Uparrow(r1,w0,w1); \Downarrow(r1,w0,w1,w0); \Downarrow(r0,w1,w0);$
$M_0 \quad M_1 \quad M_2 \quad M_3 \quad M_4$

Del; $\Updownarrow(r0,w1,r1)$; Del; $\Updownarrow(r1,w0,r0)\}$ (7)
$M_5 \quad M_6$

$\{\Updownarrow(w0); \Uparrow(r0,w1,r1,w0,w1); \Uparrow(r1,w0,r0,w1); \Downarrow(r1,w0,w1,w0); \Downarrow(r0,w1,r1,w0);$
$M_0 \quad M_1 \quad M_2 \quad M_3 \quad M_4$

Del; $\Uparrow(r0,w1,r1)$; Del; $\Updownarrow(r1,w0,r0)\}$ (8)
$M_5 \quad M_6$

is read by the 'r1' operation of M_2.

March C–. Equation 2 shows the March C–[2] algorithm, which is an improved version of March C.[9] March C– requires $10 \cdot n$ operations and detects all AFs [because M_2 together with M_3 satisfy the conditions of Table 3 (when x=1)]. It detects all SAFs (for example, M_1 detects SA1 faults and M_2 detects SA0 faults); it detects all TFs (for example, M_1 followed by M_2 detects $<\uparrow/0>$, and M_2 followed by M_3 detects $<\downarrow/1>$ TFs).

The following two steps show that March C– detects all unlinked CFins. The proof that March C– detects all unlinked CFids is similar.

1. Let C_i be coupled to any number of cells with addresses lower than i, and let C_j be the highest of those cells ($j < i$).
   (a) If C_i is $<\uparrow;\updownarrow>$ coupled to C_j, then M_1 will detect the fault. M_1 operates on C_j first, causing an $\uparrow$ transition; thereafter M_1 will operate on C_i and a 1 will be read instead of the expected 0 value.
   (b) If C_i is $<\downarrow;\updownarrow>$ coupled to C_j, then M_2 will detect the fault.
2. The proof for $j>i$ is similar to the proof of (1).

March C– also detects all CFsts.[6] The proof has to show that any two cells i and j are brought into all four states [that is $(i,j) \in \{(0,0),(0,1),(1,0),(1,1)\}$, and in each state the values of cell i and cell j are read. For example, state (1,0), denoted by S_{10}, is entered from S_{00} through the 'w1' operation into cell i of M_1 (see Equation 2). Thereafter the 'r0' operation of M_1 verifies the value of cell j. In addition, S_{10} is also entered from S_{11} through the 'w0' operation into cell j of M_4; thereafter the 'r1' operation of M_4 verifies of value of cell j.

March B. Equation 3 shows the March B algorithm.[8] It requires $17 \cdot n$ operations and detects all AFs, SAFs, TFs (also when linked with CFs), CFins (unlinked and some when linked with CFids), and linked CFids. For a proof see Suk and Reddy[8] or van de Goor.[2]

Tests for stuck-open faults (SOFs). An SOF is caused by an open word line which makes the cell inaccessible. To detect $<\forall/L>$ SOFs (that is, SOFs with a nontransparent sense amplifier), a march test has to verify that a 0 and a 1 can be read from every cell. This will be the case when a march test satisfies the conditions of Table 4. There must be a march element in which the value x and the value $\bar{x}$ are read from a cell, and another, or possibly the same, march element where the value $\bar{x}$ and the value x are read from a cell. For example, march element M_1 of Equation 3, $\Uparrow$(r0,w1,r1,w0,r0,w1), satisfies the requirements of Table 4. MATS+ of Equation 1, and March C– of Equation 2 can be modified to detect SOFs by extending M_1 with an 'r1' operation and M_2 with a 'r0' operation. They will then have the form (r0,w1,r1) and (r1,w0,r0).

The IFA-13 test[6] $\{\Updownarrow(w0); \Uparrow(r0,w1,r1); \Uparrow(r1,w0,r0); \Downarrow(r0,w1,r1); \Downarrow(r1,w0,r0);$ Del; $\Updownarrow(r0,w1);$ Del; $\Updownarrow(r1)\}$ includes four ex-

tra read operations for detecting SOFs. Two are redundant, that is, one last 'r0' and one last 'r1' operation of the march elements consisting of three operations can be removed.

Tests for data retention faults (DRFs). Any march test can be extended to cover DRFs as well. The detection of a DRF requires that a memory cell be brought into one of its logic states. A certain time must pass while the DRF develops (the leakage currents have to discharge the open node of the SRAM cell). Thereafter the contents of the cell are verified. This test must be repeated with the inverse logic value stored into the cell to test for a DRF due to an open connection in the other node of the cell. The amount of time to wait depends on the amount of charge stored in the capacitor of the node and the magnitude of the leakage current (which is difficult to determine). Empirical results[6,13] show that a wait time (called *delay time*) of 100 ms is adequate for the SRAM cells studied.

Equation 4 shows how an existing march test, which leaves all cells of the memory array in state 0, can be extended to detect DRFs. Assume that the existing march test ends with all cells in state 0; the Del elements represent the delay time which one must wait before applying the next march element. When one suspects that both pull-up devices may be open, the DRF behaves as an SOF When the sense amplifier is nontransparent to SOFs, the existing march test must be extended according to Equation 5 rather than Equation 4.

Equation 6 shows a version of the March C– test that is capable of detecting DRFs. The test is not extended. Instead, the Del elements are inserted into the existing March C– test of Equation 2 to shorten the test (reduce the test time). This procedure has the disadvantage that fault masking may occur; for example, a CF may not be detected when the coupled cell also has a DRF because the DRF may mask the CF.

Note that <↓;0/1> CFids for coupling cells j with a lower address than the coupled cell i can be detected by March C– (see Equation 2) by march element M_4 followed by M_5. The IFA-9 test[6] {⇕(w0); ⇑(r0,w1);⇑(r1,w0);⇓(r0,w1);⇓(r1,w0);Del; ⇕(r0,w1);Del; ⇕(r1)} however, has a delay Del element between the march elements ⇓(r1,w0) and ⇕(r0,w1), which will cause <↓;0/1> CFids to be masked by DRFs. In addition, the IFA-9 test will not detect DRFs involving both nodes of a cell, in a chip using a sense amplifier that is nontransparent to SOFs.

March G (most general march test). March G is a new test obtained by extending March B to cover DRFs and SOFs (see Equation 7). March G consists of seven march elements and two delay elements. It requires a test time of $23 \cdot n + 2 \cdot \text{Del}$ and is of interest when this larger test time can be tolerated for covering all faults discussed. Equation 8 shows an alternative version of March G. The two extra read operations in march element M_1 (which have been included in M_1 of March B to detect TFs linked with CFs) are distributed over M_2 and M_4 to make the test more symmetric and therefore somewhat better for BIST.[13]

Effectiveness of the functional tests

Veenstra[14] has published fault coverage results of applying functional tests to wafers with 16-Kbit SRAM chips; see Figure 6. These results apply only to the particular chips tested. However, because of the similarity between the different SRAM technologies and circuit designs, the results can be considered indicative for SRAM chips in general. Figure 6 shows the results of 11 functional tests, applied to two different wafers (indicated by the hatched lines under +45° and –45°), consisting of the following:

- *Traditional tests.*[1,2] The Zero-One, Sliding Diagonal, and GALCOL showed a poor fault coverage and will not be considered any further.

Table 4. Conditions for detecting SOFs.

Condition	March element
1	... , rx, ... , $r\bar{x}$, ...
2	... , $r\bar{x}$, ... , rx, ...

The very poor performance of the Zero-One test can be attributed to the fact that it cannot detect AFs.

- *Tests for SAFs.* The MATS-OR and MATS-AND (special versions of the MATS+ test) and the MATS+ test can detect AFs and SAFs, which account for about 80% of the faults. Marching 1/0[1,2] is a traditional test which also can detect AFs, SAFs, and TFs.
- *Tests for CFs.* Figure 6 shows that March C (which is equivalent to March C–), March A, and March B cover almost all faults.
- *Tests for neighborhood pattern-sensitive faults.* The TLSNPSF1G[2] (test to locate static neighborhood pattern sensitive faults in a type-1 neighborhood, using the two-group method) can detect SAFs and static neighborhood pattern sensitive faults. Figure 6 shows that this is not a good fault model for SRAMs.

In light of these findings, I propose the use of march tests for detecting SAFs and (linked/unlinked) TFs and CFs, and have provided a notation for describing these tests. Also, I have provided requirements for march tests to detect AFs and SOFs (when they are nontransparent to the sense amplifier), as well as extensions for march tests to allow for the detection of DRFs. Empirical results show the effectiveness of the proposed tests. The SRAM user now should have a set of tests and methods to extend those tests such that they can select and compare the tests for the faults of interest. Note especially that a new test (March G) described here detects all faults discussed in this article.

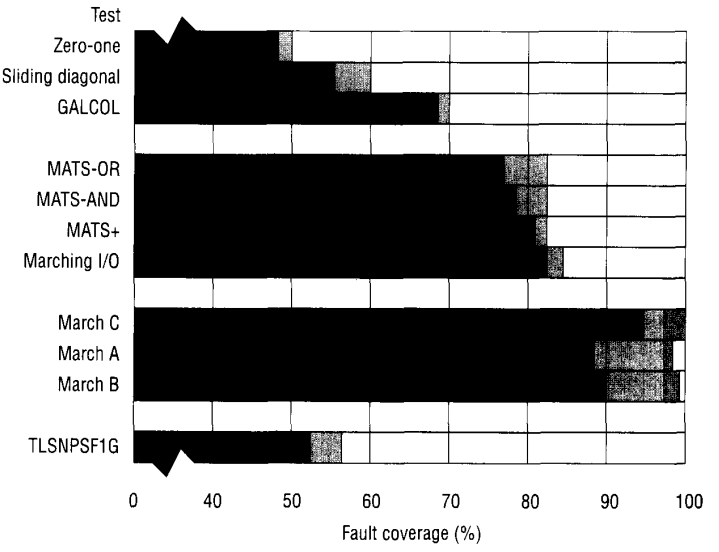
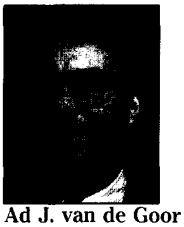


Figure 6. *Fault coverage of some tests.*

While these tests apply only to bit-wide memories, which allow only external access to a single bit, they can easily be extended to word-wide memories.[2] Space limitations precluded discussion of DC and AC parametric tests,[2,15] I_{DDQ} tests,[2,16] and tests for (dynamic) recovery faults[2] (for example, sense amplifier and write recovering faults). In addition, DRAMs require tests for neighborhood pattern-sensitive faults.[2,17] D&T

References

1. M.A. Breuer and A.D. Friedman, *Diagnosis and Reliable Design of Digital Systems*, Computer Science Press, Woodland Hills, Calif., 1976.
2. A.J. van de Goor, *Testing Semiconductor Memories, Theory and Practice*, John Wiley & Sons, Chichester, UK, 1991.
3. S.M. Thatte and J.A. Abraham, "Testing of Semiconductor Random Access Memories," *Proc. Fault-Tolerant Computer Symp.*, IEEE Computer Society Press, Los Alamitos, Calif., June 1977, pp. 81-87.
4. R. Nair, "Comments on an Optimal Algorithm for Testing Stuck-at Faults in Random Access Memories," *IEEE Trans. Computers*, Vol. C-28, No. 3, 1979, pp. 258-261.
5. A.J. van de Goor and C.A. Verruijt, "An Overview of Deterministic Functional RAM Chip Testing," *ACM Computing Surveys*, Vol. 22, No. 1, 1990, pp. 5-33.
6. R. Dekker et al., "A Realistic Fault Model and Test Algorithms for Static Random Access Memories," *IEEE Trans. Computers*, Vol. C-9, No. 6, 1990, pp. 567-572.
7. R. Nair et al., "Efficient Algorithms for Testing Semiconductor Random Access Memories," *IEEE Trans. Computers*, Vol. C-28, No. 3, 1978, pp. 572-576.
8. D.S. Suk and S.M. Reddy, "A March Test for Functional Faults in Semiconductor Random-Access Memories," *IEEE Trans. Computers*, Vol. C-30, No. 12, 1981, pp. 982-985.
9. M. Marinescu, "Simple and Efficient Algorithms for Functional RAM Testing," *Proc. IEEE Int'l Test Conf.*, IEEE Computer Society Press, 1982, pp. 236-239.
10. C.A. Papachristou and N.B. Saghal, "An Improved Method for Detecting Functional Faults in Random Access Memories," *IEEE Trans. Computers*, Vol. C-34, No. 2, 1985, pp. 110-116.
11. W. Maly, "Modeling of Lithography Related Yield Losses for CAD of VLSI Circuits," *IEEE Trans. CAD*, Vol. CAD-4, No. 3, 1985, pp. 166-177.
12. J.P. Shen et al., "Inductive Fault Analysis of CMOS Integrated Circuits," *IEEE Design & Test of Computers*, Dec., 1985, pp. 13-26.
13. D.R. Aadsen et al., "Automated BIST for Regular Structures Embedded in ASIC Devices," *AT&T Technical J.*, Vol. 69, No. 3, 1990, pp. 97-109.
14. P.K. Veenstra et al., "Testing of Random Access Memories: Theory and Practice," *IEE Proc. G*, Vol. 135 , No. 1, 1988, pp. 24-28.
15. A.K. Stevens, *Introduction to Component Testing*, Addison-Wesley, Reading, Mass., 1986.
16. R. Meershoek et al., "Functional and I_{DDQ} Testing of Static RAM," *Proc. IEEE Int'l Test Conf.*, IEEE Computer Society Press, 1990, pp. 929-937.
17. K.K. Saluja and K. Kinoshita, "Test Pattern Generation for API Faults in RAM," *IEEE Trans. Computers*, Vol. C-34, No. 3, 1985, pp. 284-287.



Ad J. van de Goor is a professor of computer architecture in the Electrical Engineering Department of the Delft University of Technology, Delft, The Netherlands. Prior to his current position he worked with Digital Equipment Corporation and with IBM. His research interests include computer architecture and testing. Van de Goor received his PhD degree from Carnegie Mellon University.

Send correspondence to the author at the Department of Electrical Engineering, Delft University of Technology, PO Box 5031, 2600 GA Delft, The Netherlands.