AN EVALUATION OF MARCH-BASED TESTING ALGORITHMS USING SWITCH LEVEL MODEL OF BIT-ORIENTED SRAM

Indira Rawat[1] **V. Sahula**[2]

Abstract

This paper reviews algorithms for memory testing, particularly SRAMs-. Various fault models like stuck-at fault and coupling faults have been included. March algorithms for testing memories are compared with respect to their fault coverage. Our contribution is towards HDL description of a switch level model of SRAM. We present results of evaluation of algorithms using appropriate fault signatures along-with using the proposed model.

1. Introduction

In the last decades, advances in Very Large Scale Integration (VLSI) semiconductor technologies have accelerated the pace of development of electronic systems. The reduction in device sizes has following implications- (1) increasingly large number of transistors can lie on to a single chip; (2) the probability of defects occurrence increases leading to decreased yield.

Testing involves generating test patterns, applying them to the circuit and analyzing the output response. Testing falls into a number of categories depending on the intended goal viz. i) testing combinational circuits ii) testing sequential circuits and iii) testing memories. Testing semiconductor memories is increasingly important because of i) the high density of current memory chips and ii) older algorithms take long to finish their testing. Testing of a system is an experiment in which the system is excited and the resulting response is analyzed to ascertain whether it behaves correctly. If the system behaves incorrectly then the next step may be to diagnose or locate the cause of misbehavior. Diagnosis assumes knowledge of the internal structure of the system under test.

[1] Lecturer, Govt. Engineering College, Ajmer. indirarawat@indiatimes.com
[2] Faculty member, Deptt. Of ECE, Malaviya National Institute of Technology, Jaipur. sahula@ieee.org

The testing of bit-oriented memories is discussed in the paper. In this paper, we review and evaluate the various algorithms for testing bit-oriented SRAMs [3][4][6][7]. We present results of evaluation using switch level fault model of SRAM cell, extending the model description in [1].

2. Testing Bit-oriented memories

Semiconductor memory arrays are capable of storing large quantities of digital information are essential to all digital systems. The amount of memory required in a particular system depends on the type of application. But in general the number of transistors utilized for the information (data) storage function is much larger than the number of transistors used in logic operations and other functions. On chip memory arrays have become widely used sub-systems in many VLSI circuits. This implies the memory testing becomes an integral and significant part of SOC testing.

2.1 Fault Models

The functional fault model of an SRAM chip consists of many blocks. For fault modeling purposes, the functional fault model may be simplified to the reduced functional fault model of Figure 1. This model includes the address decoder, memory cell array and the read/write logic. In this paper, we explore testing of memory cell array only.

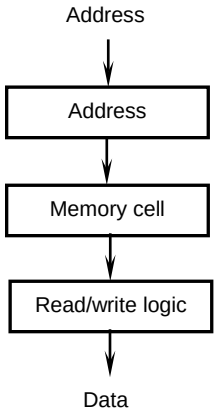

Figure 1 Reduced functional model of SRAM chip

2.1.1 Notation of faults

The faults in a single port SRAM can be divided into faults involving a single cell, and faults involving two cells. In order to describe these faults, following notation is used, similar to as in [1].

- <*S/F/R*> for faults involving single cell; S describes the sensitizing operation or state, $S \in \{0,1,w0,w1,w\uparrow,w\downarrow,r0,r1,\forall\}$, whereby 0 (1) denotes a zero (one) value, w0 (w1) denotes a write 0 (1) operation, r0

(r1) denotes a read 0 (1) operation, and ∀ denotes any operation ({01,1,w1,w0,w↓,w↑,r1,r0});

- $<S_a;S_v/F/R>$ for faults involving two cells; S_a (S_v) describes the sensitizing operation or state of the aggressor-cell, a-cell (Victim cell, v-cell);

In both the cases, F describes the value of the faulty cell (v-cell); F∈{0,1,↑,↓,?}, whereby ↑ (↓) denotes an up (down) transition, and ? denotes an undefined logical value. R describes the logical value which appears at the output of the SRAM if the sensitizing operation applied to the v-cell is a read operation; R∈{0,1,?,-}, whereby ? denotes an undefined or random logical value. A '-' in R means that the output data is not applicable in the case.

2.1.2 Simple faults

Simple faults do not influence the behavior of other faults while linked faults influence the behavior of other faults such that masking of faults may occur. Simple faults may be further classified as (i) Address decoder faults (ii) Memory Cell Array Faults. Within the Address decoder faults four sub-types exist as shown in Figure 2.

Fault1: With a certain address, no cell is accessed.
Fault2: There is no address with which a particular cell can be accessed.
Fault3: With a certain address, multiple cells are accessed simultaneously.
Fault4: A certain cell can be accessed with multiple addresses.

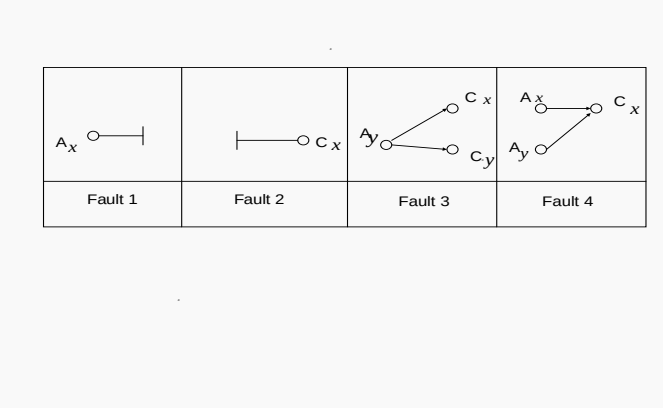

Figure 2 Address decoder faults

Memory Cell Array Faults (MCAFs) may be further classified as:

1. Single Cell faults
2. Faults between two memory cells

Single Cell faults can be further classified as:

(i) Stuck-at-faults (SAF)—The logic value of the stuck-at cell is always zero (SA0) or 1 (SA1)
(ii) Transition Faults (TF)—A cell fails to undergo a 0 to 1 (< ↑/0> TF) or a 1 to 0 transition (< ↓/1> TF)
(iii) Data Retention Faults (DRF)—A cell fails to retain its logic value after some period of time caused in SRAM by a broken (pull-up) device.
(iv) Stuck-open fault (SOF)- A cell cannot be accessed due to an open word line.

Faults between Memory Cells are known as coupling faults and involve two cells; one cell j which sensitizes the fault in other cell i. Cell i is the coupled cell called victim cell while cell j is the coupling cell or the aggressor cell. Notation used is < S_1, S_2… S_{m-1}; F> where S_1, S_2, …,S_{m-1} denote condition of (m-1 to 0) cells to sensitize the fault in the cell m (denoted by F); S_i { 0,1,↑,↓,$\underline{\updownarrow}$} for $1 \le i \le m-1$. Fault $\underline{\updownarrow}$ denotes that the coupled cell changes from x to $\bar{x}$ where $x \in \{0,1\}$. Coupling faults may be further classified as (i) Inversion coupling (ii) Idempotent coupling (iii) State coupling (iv) Disturb coupling. In Inversion coupling (CF_{in}) a ↑ or/and a ↓ write operation in the coupling cell causes an inversion in the coupled cell. This results in two CF_{in} sub-types <↑; $\underline{\updownarrow}$> and <↓; $\underline{\updownarrow}$>. Idempotent coupling (CF_{id}) is caused by an up and/or down write transition in the coupling cell, which forces a certain value in the coupled cell. Four sub-types exist: <↑; $\underline{\updownarrow}$>, <↓; $\underline{\updownarrow}$>, <↓;↓> and <↑;↑>.In State coupling (CF_{st}) a coupled cell I is forced to a certain value x only if the coupling cell is in state y. Four sub-types exist: <0;0>, <0;1>, <1;0>, <1;1>. Disturb coupling (CF_{dst}) occurs when the coupled cell is disturbed (i.e. makes an up or down transition due to r*x* or w*y* operation applied to coupling cell. Eight sub-types are: < r0; ↑>, <r1; ↑>, <w0; ↑>, <w1; ↑>, <r0; ↓>, <r1; ↓>, <w0; ↓>, <w1; ↓>.

2.1.3 Linked faults

A linked fault involves two or more simple faults. A CF is linked with another CF when both have the same coupled cell and this may cause fault masking. When a TF is linked with a CF, and the TF is in the coupling cell, the CF may not be sensitizable; for example, the TF <↑/0> in cell j linked with the CF_{id}<↑;↑> where cell j is the coupling cell. When the TF is in the coupled cell, the TF may be masked. Due to fault masking testing of linked faults is complicated. We don't discuss these faults further.

2.2 Memory test Algorithms

In the past many types of memory test algorithms have been used. The various traditional test algorithms are Zero-one, Checkerboard, Galloping pattern, Walking pattern, Sliding diagonal, Butterfly, Moving disturb. They show a poor fault coverage and high-test-time. Currently one family of tests called March test algorithms have proven to be superior in terms of test time and simplicity of the algorithm. The March test algorithms are discussed here. A March test consists of a sequence of March elements. A March element consists of sequence of operations that are applied to each cell in memory before proceeding to the next cell. An operation can consist of writing a 0 into a cell (w0), writing a 1 into the cell (w1), reading a cell with expected value 0 (r0), or reading a cell with expected value 1 (r1). After all operations of a March element have been applied to a given cell, they are applied to the next cell. The address of the next cell is determined by the address order, which can be either increasing or decreasing address order. For n addresses in increasing order, the address is from 0 to n-1 and denoted by ⇑ whereas for decreasing order the cells are addressed from n-1 to zero, denoted by ⇓. When the order is irrelevant symbol ⇕ is used. The complete notational language used for writing the algorithms is explained in Table 1.

Table 1 Notations used for March test algorithms

Notation	***Memory Action***
r	A read operation
w	Write operation
r0	Read a zero from the memory location
r1	Read a one from the memory location
w0	Write a zero to a memory location
w1	Write a 1 to a memory location
↑	Write a 1 to a cell containing 0 or the cell has a rising transition
↓	Write a 0 to a cell containing 1 or the cell has a falling transition
↕	Complement the cell contents
⇑	Increasing the memory address order
⇓	Decreasing the memory address order
⇳	Addressing order can be increasing or decreasing
∀	Denotes any memory write operation
<…..>	Denotes a particular fault described by…
<I/F>	I is the fault sensitizing condition, F is the faulty cell value

In order to describe March algorithms, following notation is used- (i) a complete march test is delimited by the curly bracket-pair '{ }'; (ii) a march element is delimited by the small bracket-pair '( )'. The March elements are separated by semicolons; (iii) and the operations within a march element are separated by commas. The various tests using such a notation for each of March algorithms are shown in Table 2. MATS++ test is written as: {⇳(w0); ⇑(r0, w1); ⇓(r1, w0);}. It has three March elements, M_0 through M_2. March element M_1 uses the ⇑ address order and performs an 'r0' operation followed by a 'w1' operation on a cell before proceeding to the next cell.

Table 2 Various steps applied to memory in March tests

Algorithm	
MATS	{⇳(w0); ⇳(r0,w1); ⇳(r1);}
MATS+	{⇳(w0); ⇑(r0,w1); ⇓(r1,w0);}
MATS++	{⇳(w0); ⇑(r0,w1); ⇓(r1,w0,r0);}
MARCH X	{⇳(w0); ⇑(r0,w1); ⇓(r1,w0); ⇳(r0);}
MARCH C-	{⇳(w0); ⇑(r0,w1); ⇑(r1,w0); ⇓(r0,w1); ⇓(r1,w0) ⇳(r0);}
MARCH A	{⇳(w0); ⇑(r0,w1,w0,w1); ⇑(r1,w0,w1); ⇓(r1,w0,w1,w0); ⇓(r0,w1,w0);}
MARCH Y	{⇳(w0); ⇑(r0,w1,r1); ⇓(r1,w0,r0); ⇳(r0);}
MARCH B	{⇳(w0); ⇑(r0,w1,r1,w0,r0,w1); ⇑(r1,w0,w1); ⇓(r1,w0,w1,w0); ⇓(r0,w1,w0);}

The faults within a memory cell array are considered, as the cell array dominates the silicon area of the memories. Tests for faults in the memory cell array will detect the same faults in the read/write logic but the tests cannot distinguish between the two, hence the faults can only be detected and not located. There are two conditions to be satisfied by a test for detecting address decoder (AF) faults. The test should have at least two March elements of opposite order ie. It should have (i) $\Uparrow$ (rx,…,w$\bar{x}$) and (ii) $\Downarrow$ (r$\bar{x}$,…wx). It can have any number of additional operations. The fault coverage of the traditional March test algorithms are as follows, see Table 3.

Table 3 Fault coverage of March based algorithms

Algorithm	***SAF***	***AF***	***TF***	***CF_{in}***	***CF_{dst}***	***CF_{st}***
MATS	All					
MATS+	All	All				
MATS++	All	All	All			
MARCH X	All	All	All	All		
MARCH C	All	All	All	All	All	All
MARCH A	All	All	All	All	All	
MARCH Y	All	All	All	All	All	
MARCH B	All	All	All	All	All	

The various March test algorithms do not ensure 100% fault coverage though March C is the most widely used industry standard algorithm along with Mats+ . Several other algorithms have been proposed from time to time for increased fault coverage. The complexity of the algorithm increases and also the test time. Given a test algorithm, a fault dictionary can be constructed which lists the faults and the corresponding syndromes. Each row (column) of the dictionary corresponds to a certain fault class (a read operation from the March test). If the March test has r read operations then $<\alpha_{i0,}R_{0,}\,\alpha_{i1}R_{1,}\,\ldots\ldots\alpha_{ir\text{-}1}R_{r\text{-}1>}$ is the signature of the i_{th} fault named March syndrome.

The fault dictionary of the various March tests has been derived using Binary Decision Diagrams. By applying the various algorithms to memory, we can compare the result with the given fault dictionary. If the result matches any of the fault syndromes then that given fault is known to exist. We have simulated the above algorithms in C after inserting faults in the memory. Fault dictionary for any of the March algorithm is constructed by binary decision diagrams as shown in Figure 3. We take March A for explanation. Start from initial stage (IS). First w0 operation is performed in arbitrary order. Then the value 0 is read from each cell followed by w1. Operation r0 will return a 1 value for a SA1 fault. Coupling fault CFin(L,↑,↕) and CFin(L,↓,↕). If r0 returns a 0 indicating correct read operation then it is possible for a SA0 and CFin(H, ↑,↕) fault to exist. After this, w1, w0, w1 operations are performed on each cell. The next step is a read operation r1. If it returns 0 then SA0 does not exist, if 1 is returned then SA0 exist. Read operation r1 cannot return 0 value, hence this path is aborted. The next read operation might again return a 0 or a 1 value. Proceeding likewise, we can trace the paths of different faults and aborting the paths that are not possible. The entire tree can be constructed.

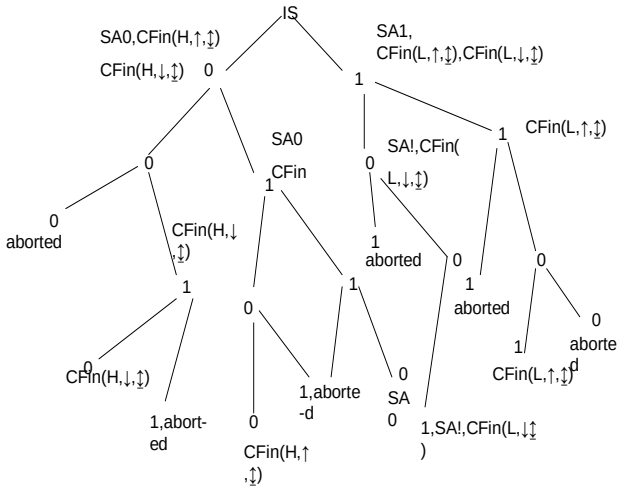

Figure 3 Finding fault signatures for March A Algorithm

As indicated in Figure 3, the signatures of faults March A algorithm is capable of detecting can be written as in Table 4.

Table 4 Fault dictionary for March A

Fault	***Signature***
SAF0	0110
CFin(H,↑,↕)	0010
CFin (H, ↓,↕)	0100
SAF1	1001
CFin (L, ↓,↕)	1001
CFin (L, ↑,↕)	1101

Thus this algorithm can detect inversion coupling and stuck-at faults. Similarly for March C algorithm, the dictionary can be constructed as in Table 5.

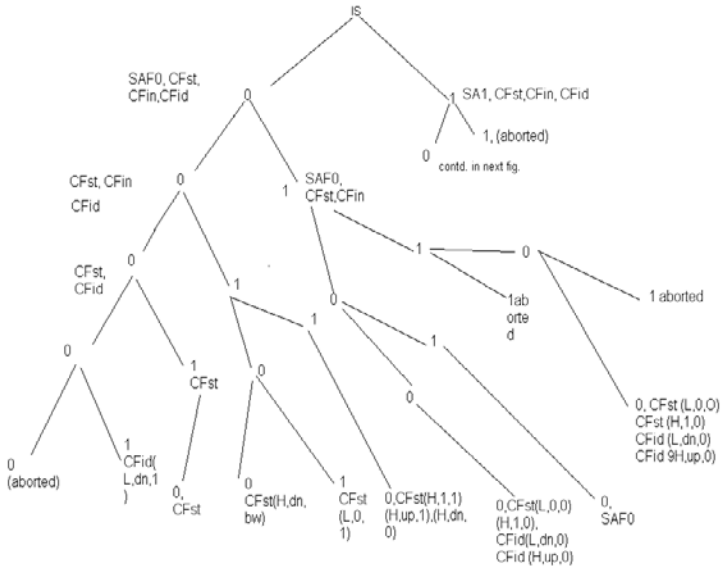

Figure 5 (a)

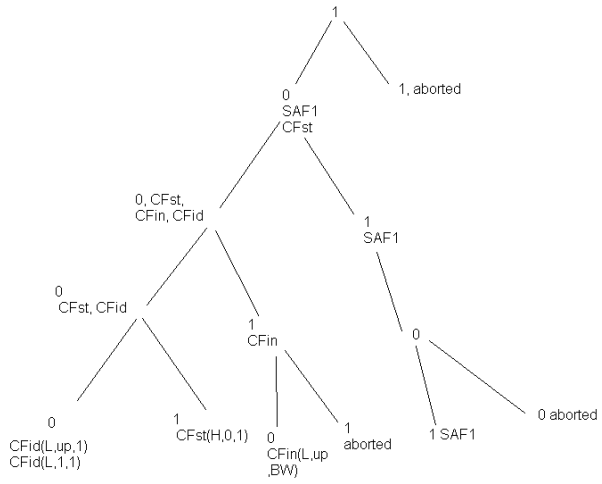

Figure 4 (a) & (b): Fault signatures for March C algorithm

The corresponding fault dictionary for March C algorithm can be constructed as in Table 5.

Table 5 Fault dictionary for March C

Fault	***Signature***
SAF0	01010
SAF1	10101
CFst(L,0,0)	01000
CFst(H,0,0)	00010

CFst(H,0,1)	10001
CFst(L,1,0)	00101
CFst(L,1,1)	10000
CFst(L,up,bw)	00010
CFst(L,0,1)	00101
CFid(L,up,1)	10000
CFid(L,up,0)	00010
CFin(L,up,bw)	10010
CFin(L,dn,bw)	01001
CFin(H,up,bw)	01100

Here the notation CF(Ap, As, Vs) represents the CF for a bit-oriented RAM where Ap is either H or L representing the relative position high or low of the aggressor cell with respect to the victim cell. 'As' represents the victim state which can be up/↑, dn/↓, bw/↕ and Vs represents the faulty state of the victim, which can be 0,1 or ↕/bw. For example CF_{in}(L,dn,bw) represents an inversion coupling fault where aggressor cell is at a higher address than the victim and a down transition in it will force the victim to a complement value.

2.3 Switch level model

SRAM cell of 6 transistors is chosen. The transistors are replaced by their switch level equivalents. We utilize for "coupling between two nodes" both the resistive as well as capacitive connection. The cell is described using Verilog HDL.

3. Results

The various March based algorithms have been reviewed and evaluated using a simplistic model of SRAM. We propose a switch level model of SRAM, which includes the coupling effects between any two adjacent cells. The switch level model is captured using HDL (Verilog). Further exploration is underway using this model to evaluate the March based algorithms.

4. Conclusions

There are many different faults that occur in a memory array. Testing memory is time consuming, as it is necessary to test each cell to detect all possible faults. The most common ones have been discussed in this paper. March tests are the most widely used tests that are fast and have a linear time complexity. In the new memories that are faster and more complex, new faults are detected. The traditional March algorithms do not detect all the faults in them and need to be modified to accommodate all the fault types. Modifications are being suggested by researchers to detect newer faults and also to have different signatures for different faults as the algorithm cannot differentiate between the faults having the same signatures. We perceive that a high level description of functional fault model of SRAM would be useful for design and evaluation of efficient testing algorithms for larger memories.

4. References

[1] S. Hamdioui and A. J. vande Goor, "An experimental analysis of spot defects in SRAMs: Realistic fault models and tests," In *IEEE Asian Test Symposium*, Dec. 2002, pp 131-138.

[2] K. Zarrineh, A. P. Deo and R. D. Adams, ",”In *Proc. IEEE International Workshop on Memory Technology, Design and Testing*, 2000, pp. 119-124.

[3] Jin-Fu Li, Kuo-Liang Cheng, Chih-Tsun Huang and Cheng-Wen Wu, “March-based RAM Diagnosis Algorithms for Stuck-At and Coupling Faults”, In *Proc. IEEE Int. Test Conf.*, 2001, pp. 758-767.

[4] A.J.van de Goor, “Using March Tests to Test SRAMs, ” *IEEE Design and Test of Computers*. Vol.-10, no.1, pp. 8-14, March1993.

[5] Y. Zorian and V.A. Vardanian, ”A March-based Fault Location Algorithm for Static Random Access Memories”, In *Proc. IEEE International Workshop on Memory Technology, Design and Testing*, 2002.

[6] A .J. van de Goor and I.B.S. Tlili, “March tests for Word-oriented Memories,” Delft University Technology, Dept. of Electrical Engg., Delft, The Netherlands, Technical Report No. 1-68340-44-(1997) 08, 1997.

[7] R. Dekker, F. Beenker and L. Thijssen, “A Realistic Fault Model and Test Algorithms for Static Random Access Memories,” *IEEE Tran. on CAD*, vol.9, no.6, pp. 567-572, 1990.