Automatic March tests generation for multi-port SRAMs

Automatic March Tests Generation for Multi-Port SRAMs

A. Benso, A. Bosio, S. Di Carlo, G. Di Natale, P. Prinetto
Politecnico di Torino
Dipartimento di Automatica e Informatica
Torino, Italy
E-mail {benso, bosio, dicarlo, dinatale, prinetto}@polito.it
http://www.testgroup.polito.it

Abstract

Testing of Multi-Port (MP) SRAMs requires special tests since the multiple and simultaneous access can sensitize faults that are different from the conventional single-port memory faults. In spite of their growing use, few works have been published on testing MP memories. In addition, most of the published work concentrated only on two ports memories (i.e., 2P memories). This paper presents a methodology to automatically generate march tests for MP memories. It is based on generations of single port memory march test firstly, then extending it to test a generic MP SRAMs. A set of experimental results shows the effectiveness of the proposed solution.

1. Introduction

Multi-Port memories (MP) peculiarity is their capability of performing more than one operation simultaneously. Semiconductor MPs are composed of a unique array of memory cell and a *p*-port to access it ($p \geq 2$). Each port has an independent set of address, control, and data buses, making possible writing a value on a cell while another cell is being read. Multi-port SRAMs are nowadays widely used as embedded memories in a plenty of digital systems, like telecommunications ASICs and multiprocessor systems [**1**].

The problem of testing multi-port memories has been faced using and *ad-hoc* technique, without targeting specific functional fault models, In [2] [3] the authors assume that Single Port (SP) test algorithms provide a high fault coverage when applied to MP memories. The test methodology performs SP test algorithms on each port, but the deep fall of the effectiveness of the applied tests shows that ad-hoc fault models for MP must be adopted.

In [4] a new theoretical fault model (*complex coupling fault*) and its test solution are presented. Unfortunately the fault model is not validated by experimental analysis (i.e., it isn't a *realistic* fault model), and the test complexity (i.e., the length of the test algorithm) is exponential w.r.t. the number of port: $O(n^p)$.

In [5] the authors present realistic fault models validated by industrial analysis. Taking into account the simultaneous access in memories, march tests were developed.

All the published tests solutions have been manually generated, a task that always requires a lot of time, expertise, and that sometimes does not succeeds in covering particularly complex memory faults.

Although several methodologies to automatize the march tests generations have been proposed [6] [7] [8] [9] [10], none of them faces the problem of the MP test.

In this paper, we present a systematic approach to automatically generate March Tests for MP SRAMs based on the tests generator engine presented in [10]. Moreover taxonomy of realistic fault models for generic p-port memories will be presented.

The paper is structured as follows: section 2 presents the proposed test generation methodology. In Section 3 a complete description of memory faults modeling will be exploit. Section 4 details the notation used to represent march tests for both single and multi port memories. In section 5 a detailed analysis of the methodology is presented, while section 6 provides experimental results that proof the efficiency of our approach. Section 7 summarizes the main contributions and future developments of this research.

2. The Proposed Test Generation Methodology

The adopted methodology relies on a formal model representing the fault behaviour (see Section 3).

Proceedings of the Third IEEE International Workshop on Electronic Design, Test and Applications (DELTA'06)
0-7695-2500-8/05 $20.00 © 2005 IEEE

To automatize the test generation phase we first generate the single port march test by resorting the march test generation tool published in [10].
The main steps of the methodology are:

(i) automatically translate the FPs to an "*operational*" representation of the faulty behaviour, referred to as *Addresses FP*, or AFP.

(ii) The original memory graph model is then automatically modified according to the AFP, to build the fault graph that is then traversed to generate the test. An efficient implementation has been done, profitably exploiting pruning conditions imposed by the goal of primarily generating March Test.

(iii) After the generation of Single Port (SP) March test, we apply the Multi Port translation able to extend the SP March in to MP march.

Each generated march test has been validate by simulation performed by memory fault simulator tool [11]

The overall generation methodology is summarized in Figure 1.

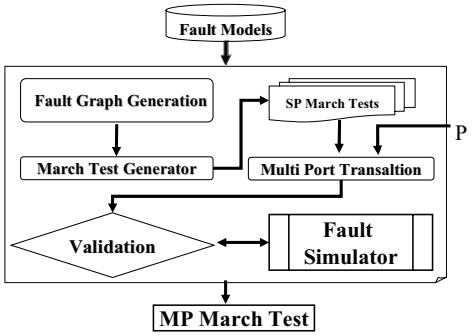

Figure 1 : Automatic MP march tests generation flow

3. Fault modeling

A *Functional Fault Model* (FFM) is a deviation of the memory behavior from the expected one under a set of performed operations. A FFM involves one or more *Faulty Memory Cells* (FC) classified in two categories: *Aggressor cells* (*a*-cells), i.e., the memory cells that sensitize a given FFM and *Victim cells* (*v-cells*), i.e., the memory cells that show the effect of a FFM.

Each faulty behavior is sensitized by a sequence of *stimuli* applied on the FCs.

In testing SRAMs, the stimuli to be applied are *memory operations*. When dealing with MP SRAMs, each stimulus could be applied on a different port. MP faults can thus be ranked into two main classes:

- *Strong fault*: a memory fault that can be **fully** sensitized by an operation; e.g., a single-port write or read operations fails, two simultaneous read operations fail, etc.
- *Weak fault:* a fault which is **partially** sensitized by an operation; e.g., due to a defect that creates a small disturbance of the voltage of the true node of the cell. However, a fault can be *fully sensitized* (i.e., become strong) when two or more weak faults are sensitized simultaneously, since their faults effect can be additive. This may occur when a MP operation is applied.

Fault modeling requires a rigorous formalism; first of all we have to specify the *initial conditions* of the cell, i.e., the value (state) of the memory cell, where we are going to apply the operations. Hereinafter we use n as the size of the memory (i.e., the number of memory cells)

Definition 1: C is the set of the memory states (values), formalized as

$$C = \{0^{[i]}, 1^{[i]}, -^{[i]} \mid 0 \le i \le n\text{-}1\} \qquad (1)$$

where apex identifies the address of the cell. If the address is omitted, it means that the state could be applied on every memory cell indifferently. The '-' denotes a *don't care* condition.

Definition 2: X is the set of the memory operations, formalized as

$$X = \{r^{[i]}_{[d]}, w^{[i]}_{d} \mid 0 \le i \le n\text{-}1;\ d \in (0,1)\} \cup \{t\} \qquad (2)$$

where:

- w^i_d : a *write* operation of the value d performed in the cell i;
- r^i_d : a *read* operation performed in the cell i. The value d it is not strictly needed in case of a read operation. If used, it means the expected value that should be red from the i-th memory cell;
- t : a *wait* operation for a defined period of time. This additional element is needed to deal with *Data Retention Faults* [6].

If the address is omitted, it means that the operation could be applied on every memory cell, indifferently.

Each FFM can be described by a set of Fault Primitives (FPs) [12].

Definition 3: A *Sequence of conditions/operations* (S) is the minimum sequence of stimuli and conditions of length m needed to sensitize the fault. The j-th condition/operation is represented as $c[x]$, where $c \in C$, and $x \in X$.

Definition 4: A *Fault Primitive* FP represents the difference between an expected (fault-free) and the observed (faulty) memory behavior, denoted by:

$$< SA\ ;\ SV\ /\ F\ /\ R > \qquad (3)$$

Where *SA* and *SV* are the set of *S* respectively applied to *a*-cell and *v*-cell, needed to sensitize the given fault.

Since *S* could be applied via several ports in parallel, *SA* and *SV* are represented as:

$$(S_1)^0 : (S_2)^1 : \ldots : (S_p)^{p-1} \qquad (4)$$

The ":" denotes the fact that the sequences of operations (from 0 to p-1) are applied simultaneously via the *p* ports. The apex denotes the target port.

$F = \{(f)^n \mid f \in C\}$ is the faulty behavior, i.e., the value (state) stored in the victim cells after applying *S*. $R = \{(r)_n \mid r \in C\}$ is the sequence of values read on the aggressor cell when applying *S*.

As an example FP = $< 0w_1 : r_1 ; 0 / 1 / - >$ means that the operations 'w1' and 'r1' performed on the a-cell, trough the two ports, when the initial state is 0 for both a and v cells, causes the victim to flip. No addresses are specified; therefore this fault can affect each couple of memory cell.

The terminology of weak and strong faults is used in representing the MP FFMs as follow:

- *FP* denotes a *strong fault* represented by its FP, while *wFP* denotes the *weak fault* FP. For example, *RDF* denotes a strong Read Destructive Fault, while *wRDF* denotes a weak Read Destructive Fault.
- *w*FP1&*w*FP2…&*w*FPp: denotes a *p*PF consisting of *p* weak faults; "&" denotes the fact that the *p* faults *in parallel* (i.e., simultaneously) form the *p*-port fault (*p*PF). For example the wRDF&wRDF&wRDF denote a 3PF based on three weak RDFs [1]

Several FPs classification rules can be adopted, based on the number of memory operations (m) needed to sensitize the FP (e.g., *static* when m = 1 or *dynamic* fault elsewhere); or based on the number of memory cells (#FC) involved by the FP (e.g., *single-cell* where #FC = 1 or *n-cells* fault, elsewhere) [12].

3.1. Multi Port Constraints

As discussed in the previous section, a MP FFM requires the use of the ports to perform the sensitizing operations in parallel. Physical constraints impose some limitations on the set of allowed concurrent memory operations:

- simultaneous *write* operations are not allowed;
- simultaneous *read* operations are allowed;
- simultaneous *read* and *write* are allowed. In this case the write operation has the highest priority and therefore the read data will be discarded;
- simultaneous operations are symmetric: $(0w_0{:}r_0)$ sensitize the same fault as $(r_0{:}0w_0)$;

All the above constraints have been validated by simulation experiments in [1].

4. March Test notation

As pointed out in [13] a so called *March Test* is composed of a sequence of *March Elments* (MEs). A *March Element* is a sequence of memory operations applied on every cell of the memory. The way one moves from a certain address to the next one is called *address order* (AO) and it characterizes each ME. The address order can be specified resorting to the following symbols:

- '⇑' : *Increasing Address Order* (Up AO)
- '⇓' : *Decreasing Address Order* (Down AO)
- '⇕' : *Don't care address order* : it is possible to use either the *up* or *down* AO

Not necessarily an up/down AO means that the ME starts from the lowest/highest memory address to the highest/lowest address. One can choose an arbitrary AO and labeling it as up, without reducing the fault coverage of a given March Test [14]. The only constraint is that the down AO must be exactly the reverse of the Up AO. Hereinafter we denote a March Test by a '*{…}*' bracket, and a ME by a '(…)' bracket. The *i*-th operation is defined as $op_i \in X$, where the address of the target cell is not indicated since already specified by the address order. The *complexity* of a March Test is defined as the number of memory operations it includes. We can formalize the above definitions resorting to the following context free grammar [15]:

Definition 5: A *SP March Test* is defined as:

$$MTGsp = (N, \sum, S, P) \qquad (5)$$

where:

- $N = \{MT, ME, AO, OP, D\}$ is the collection of the nonterminal symbols;
- $\sum = \{$'0', '1', '*w*', '*r*', ',', '(', ')', '{', '}', '⇑', '⇓', '⇕'$\}$ is the set of terminal symbols (i.e., the alphabet) ;
- $S = MT$ is the start symbol. $S \in N$;
- $P \subseteq N \times (N \cup \sum)^*$ is the set of productions detailed as follows:

$$\begin{cases} 1)\ MT \rightarrow \text{'\{'ME'\}'} \\ 2)\ ME \rightarrow AO\text{'('OP')'} \mid AO\text{'('OP')'ME} \mid \varepsilon \\ 3)\ OP \rightarrow \text{'}w'_D \mid \text{'}r'_D \mid \text{'}w'_D\text{',' } OP \mid \text{'}r'_D\text{','}OP \\ 4)\ AO \rightarrow \text{'}\Uparrow\text{'} \mid \text{'}\Downarrow\text{'} \mid \text{'}\Updownarrow\text{'} \\ 5)\ D \rightarrow \text{'0'} \mid \text{'1'} \end{cases}$$

As an example consider the follow March Test:

$$\{\Uparrow (w_1)\ \Downarrow (r_1, w_0)\ \Downarrow (r_0)\} \qquad (6)$$

Starting from (5) we can extended it to apply the operation simultaneously.

Definition 6: A *MP March Test* is defined as:

$$MTGmp = (N_p, \sum_p, S_p, P_p) \qquad (7)$$

where:

- $N_p = N \cup \{OPs\}$ is the set of the nonterminal symbols;
- $\sum_p = \sum \cup \{':', '-', 'n'\}$ is the set of terminal symbols (i.e., the alphabet). Don't care '-' denotes that any operation is allowed on the selected port, and '*n*' denotes that no operations are allowed on the selected port;
- $S_p = S$ is the start symbol. $S_p \in N_p$;
- $P_p \subseteq N_p \times (N_p \cup \sum_p)^*$ is the set of productions detailed as follows:

$$\begin{cases} 1)\ MT \rightarrow \text{'\{'ME'\}'} \\ 2)\ ME \rightarrow AO\text{'('OP')'} \mid AO\text{'('OP')'ME} \mid \varepsilon \\ 3) OP \rightarrow OPs \mid OPs\text{','}OP \\ 4)\ OPs \rightarrow \text{'}w'_D\text{':'}OPs \mid \text{'}r'_D\text{':'}OPs \mid \text{'}w'_D\text{':'}OPs1 \mid \text{'}r'_D\text{':'}OPs1 \\ 5) \text{OPs1} \rightarrow \text{'}w'_D \mid \text{'}r'_D \mid \text{'–'} \mid \text{'}n' \\ 5)\ AO \rightarrow \text{'}\Uparrow\text{'} \mid \text{'}\Downarrow\text{'} \mid \text{'}\Updownarrow\text{'} \\ 6)\ D \rightarrow \text{'0'} \mid \text{'1'} \end{cases}$$

The march test (6) could be extended to MP test purpose as:

$$\{\Uparrow (w_1 : n)\ \Downarrow (r_1 : -,\ w_0 : r_1)\ \Downarrow (r_0 : -)\} \qquad (8)$$

This march test has been translated for two port memories (i.e., only two operations at each time are applied in parallel).

5. Multi Port Translation

The translation of a single port march test to a generic *p* Port march test is feasible under the constraints presented in Section 2.1.This phase requires as input the single port march test previously generated, and the number *p* of port (Figure 1)

The input march test has to be formatted by the march test generator phase in order to evidence the nature of the memory operations (i.e., by labeling each operations of the march test), that can be clustered in three categories:

1) *Initializing operations* : their can be only write operations;
2) *Sensitizing operations* : their could be either write or read operations;
3) *Observing operations* : their can be only read operations;

Note that an operation could be, at the same time, sensitize and observe the fault (i.e., read fault [13]) or initialize and sensitize the fault (i.e., state fault [13])

This labeling procedure is done by the SP march test generator, where the information about each operations (i.e., if an operation is a sensitizing or initializing or observing) directly from the fault model (Section 3).

This phase corresponds to a set of rewrite rules, since the single port march test can be consider as a string accepted by the grammar defined in (5) where each symbol is a memory operation. Each rewrite rules is represented by the *regular expression* formalism [15].

Table 1 shows the rewrite rules, as an example if an operation is tagged "*Sensitizing*", then rule #1 will be adopted. In case of multiple labelling (i.e. the operation is labelled both "*Sensitizing*" and "*Observing*"); the operator precedence has been implemented by the order of rewrite rules.

Table 1 : rewrite rules

#	***Operation***	***Rewrite Rules***
1	*Sensitizing*	$w_d \rightarrow w_d : r_x : \ldots : r_x$ $r_d \rightarrow r_d : r_d : \ldots : r_d$
2	*Initializing*	$w_d \rightarrow w_d : n : \ldots : n$
3	*Observing*	$r_d \rightarrow r_d : - : \ldots : -$

The rule having the highest precedence (#1 table 1) is that related to sensitizing operations, since we must add *p*-1 different operations to apply in parallel to fully sensitize the fault.

The problem of what kind of added operations (write or read) is solved by constraints detailed in Section 2.1. Only simultaneous *p* read operations are supported or one write and *p*-1 read operations are supported. Therefore rule #1 inserts *p*-1 read operations. The expected value to read from the memory cell (*x*) depends from the previous memory state.

6. Experimental results

This section reports some experimental results obtained applying the proposed algorithm to automatically generate March Tests to cover different sets of faults. We first generate march tests able to cover 3 port FFMs detailed in [5] and here summarized for sake of readability.

FFMs involving one cell are:

- *w*DRDF&*w*DRDF&*w*DRDF : applying three simultaneous read operations to the v-cell causes the cell to flip, but returning the correct values. (Deceptive Read Destructive Fault, DRDF);
- *w*RDF&*w*RDF&*w*RDF : applying three simultaneous read operations to the v-cell causes the cell to flip, returning the incorrect value. (Read Destructive Fault, RDF)

FFMs involving two cells are:

- *w*CFds&*w*CFds&*w*CFds : applying three simultaneous operations to the a-cell causes the cell to flip. (Disturb Coupling Fault, CFds)
- *w*CFds&*w*DRDF&*w*DRDF : applying three simultaneous read operations to the v-cell causes the cell to flip if the a-cell is in a specific state, but returning the correct values.
- *w*CFds&*w*RDF&*w*RDF : applying three simultaneous read operations to the v-cell causes the cell to flip if the a-cell is in a specific state, returning the incorrect values.

Consider as an example the *w*CFds&*w*CFds&*w*CFds that is described by 8 FPs in Figure 2.

$$\langle 1w_0 : r_d : r_d;0/\uparrow/-\rangle,\langle 1w_0 : r_d : r_d;1/\downarrow/-\rangle,\langle 0w_1 : r_d : r_d;0/\uparrow/-\rangle,$$
$$\langle 0w_1 : r_d : r_d;1/\downarrow/-\rangle,\langle r_x : r_x : r_x;0/\uparrow/-\rangle,\langle r_x : r_x : r_x;1/\downarrow/-\rangle.$$

Figure 2 : $x \in \{0,1\}$, d = don't care

The FFM is fully sensitized by the applications of the three weak faults on the different memory port. We generate first the SP march test covering the first FPs and summarized in Figure 3

$$\langle 1w_0;0/\uparrow/-\rangle,\langle 1w_0;1/\downarrow/-\rangle,\langle 0w_1;0/\uparrow/-\rangle,$$
$$\langle 0w_1;1/\downarrow/-\rangle,\langle r_x;0/\uparrow/-\rangle,\langle r_x;1/\downarrow/-\rangle.$$

Figure 3 : single port FPs, $x \in \{0,1\}$

The generated SP march test is

$$\{\Uparrow (w_1)\ \Downarrow(r_1,w_0)\ \Downarrow(r_0,w_1)\ \Uparrow(r_1,w_0)\ \Uparrow(r_0,w_1)\ \Downarrow(r_1)\} \quad (9)$$

After MP translation (i.e., applying the rewrite rules Table 1) we obtain:

$$\{\Updownarrow (w_1\text{-}:\text{-})\ \Uparrow(r_1{:}r_1{:}r_1,w_0{:}r_1{:}r_1)\ \Downarrow(r_0{:}r_0{:}r_0,w_1{:}r_0{:}r_0) \quad (10)$$
$$\Downarrow(r_1{:}r_1{:}r_1,w_0{:}r_1{:}r_1)\ \ \Uparrow(r_0{:}r_0{:}r_0,w_1{:}r_0{:}r_0)\ \Updownarrow (r_{1:}\text{-}:\text{-})\}$$

That is able to cover *w*CFds&*w*CFds&*w*CFds [5].

Table 3 shows the resulting March Tests. For each march test we report its complexity (length of march test) and the equivalent march test found in literature, and the targeted fault list, the last column shows the cpu time (in second). The algorithm has been implemented in about 900 lines of C++ code, compiled with *gcc* compiler. All the experiments are performed on an *ASUS, AMD 1500Mhz* based Laptop with 512 MB of RAM. Table 2 reports the fault list covered by each march test. The first four generate march tests have been already published [5], the last three are unknown, and #7 (whose complexity is 22n) has the same structure of march SS [16]. It is able to detect all the static faults (one and two-cells) extensions for multiple-port memories. All generated March Tests have been verified using an ad hoc memory fault simulator [11] able to validate their correctness w.r.t. the target FP list. The fault simulator is also used to check the non-redundancy of each generated March Test.

7. Conclusion

This paper presented a methodology to automatically generate March Tests for multiple-port memories. A general model has been used to represent known memory static faults, and to possibly add new user-defined faults. The generation process stems from the generation of SP march tests, then properly translated into MP march tests by applying a set of rewrite rules. Experimental results have been presented to prove the applicability and the efficiency of the proposed approach. On going activities are focused on the automatic generation of MP march tests targeting additional classes of memory fault, including *Dynamic* and *Linked Faults*.

Table 2 : fault list

#	Fault List
#1	*w*DRDF&*w*DRDF&*w*DRDF *w*RDF&*w*RDF&*w*RDF
#2	*w*CFds&*w*CFds&*w*CFds
#3	*w*CFds&*w*DRDF&*w*DRDF *w*CFds&*w*RDF&*w*RDF
#4	All the 3port FFM
#5	All the single cell Static Fault
#6	All the CFds
#7	All static FFMs

Table 3 : experimental results

#	Algorithm	O (n)	Known March Test	CPU time (s)
#1	{⇕(w_0:-:-) ⇕(r_0:r_0:r_0,r_0:-:-) ⇕(w_1:-:-) ⇕(r_1:r_1:r_1,r_1:-:-)}	6n	3PF1	0.030
#2	{⇕(w_1-:-) ⇑(r_1:r_1:r_1,w_0:r_1:r_1) ⇓(r_0:r_0:r_0,w_1:r_0:r_0) ⇓(r_1:r_1:r_1,w_0:r_1:r_1) ⇑(r_0:r_0:r_0,w_1:r_0:r_0) ⇕(r_1:-:-)}	10n	3PF2a	0.028
#3	{⇕(w_1:-:-) ⇑(r_1:r_1:r_1, r_1:-:-,w_0:-:-) ⇓(r_0:r_0:r_0, r_0:-:-,w_1:-:-) ⇓(r_1:r_1:r_1, r_1:-:-,w_0:-:-) ⇑(r_0:r_0:r_0, r_0:-:-,w_1:-:-)}	13n	3PF2v	0.210
#4	{⇕(w_1:-:-) ⇑(r_1:r_1:r_1,r_1:-:-,w_0:r_1:r_1) ⇓(r_0:r_0:r_0,r_0:-:-,w_1:r_0:r_0) ⇓(r_1:r_1:r_1,r_1:-:-,w_0:r_1:r_1) ⇑(r_0:r_0:r_0,r_0:-:-,w_1:r_0:r_0) ⇕(r_1:-:-)}	14n	3PF	0.204
#5	{⇕(w_1:-:-) ⇕(w_0:r_1:r_1) ⇕(r_0:r_0:r_0, w_0:r_0:r_0, r_0:-:-) ⇕(w_1:r_0:r_0) ⇕(r_1:r_1:r_1, w_1:r_1:r_1,r_1:-:-)}	9n	-	0.093
#6	{⇕(w_0-:-) ⇑(r_0:r_0:r_0,w_0:r_0:r_0,w_1:r_0:r_0) ⇓(r_1:r_1:r_1,w_1:r_1:r_1,w_0:r_1:r_1) ⇓(r_0:r_0:r_0,w_0:r_0:r_0,w_1:r_0:r_0) ⇑(r_1:r_1:r_1,w_1:r_1:r_1,w_0:r_1:r_1) ⇕(r_0-:-)}	14n	-	0,201
#7	{⇕(w_0-:-) ⇑(r_0:r_0:r_0,r_0:-:-,w_0:r_0:r_0,r_0:-:- ,w_1:r_0:r_0) ⇓(r_1:r_1:r_1,r_1:-:-,w_1:r_1:r_1,r_1:-:- ,w_0:r_1:r_1) ⇓(r_0:r_0:r_0,r_0:-:-,w_0:r_0:r_0,r_0:-:- ,w_1:r_0:r_0) ⇑(r_1:r_1:r_1,r_1:-:-,w_1:r_1:r_1,r_1:-:- ,w_0:r_1:r_1) ⇕(r_0:-:-)}	22n	-	0.212

8. References

[1] S.Hamdioui, "*Testing Multi-Port Memories: Theory and Practice*"

[2] M.J. Raposa, "*Dual-Port Static Ram Testing*" ITC 1988, IEEE International Test Conference, 1988, pp. 362-368.

[3] T. Matsumara, "*An Efficient Test Method for Embedded Multi-Port RAM with BIST Circuitry*" MTDT 1995: IEEE International Workshop on Memory Technology, Design and Testing, 1995, pp. 62-67.

[4] K. Chakrabortry P. Mazumder, "*New March Test for Multi-Port RAM Devices*", Journal of Electronic Testing: Theory and Application , vol 16, 2000, pp. 389-395.

[5] S. Hamdioui; A.J. Van de Goor; D. Eastwick, M. Rodgers, "*Detecting unique faultsi in multi-port SRAMs*" ATS 2001, 10th IEEE Asian Test Symposium, 2001, pp. 37 -42

[6] A. J. van de Goor, B. Smit, "*Generating March Tests Automatically"*, ITC 1994, IEEE International Test Conference, 1994, pp. 870-877

[7] K. Zarrineh, S. J. Upadhyaya, S. Chakravarty, "*A New Framework for Generating Optimal March Tests for Memory Arrays*", ITC 1998, IEEE International Test Conference, 1998, pp. 73-82

[8] D. Niggemeyer, E.M. Rudnick, "*Automatic Generation of Diagnostic Memory Tests Based on Fault Decomposition and Output Tracing*", IEEE Transactions on Computers, Volume: 53 , Issue: 9 , Sept. 2004 pp. 1134 – 1146.

[9] A. Benso, S. Di Carlo, G. Di Natale,P. Prinetto, "*An optimal algorithm for the automatic generation of March tests*" DATE 2002, IEEE Design, Automation and Test in Europe Conference and Exhibition, 2002 pp. 938 -943

[10] A. Benso, A. Bosio, S. Di Carlo, G. Di Natale, P. Prinetto, "*Automatic March Tests Generation for Static and Dynamic Faults in SRAMs*", ETS 2005, 10th IEEE European Test Symposium, 2005.

[11] A. Benso, S. Di Carlo, G. Di Natale, P. Prinetto, "*Specification and design of a new memory fault simulator*", ATS 2002, 11th IEEE Asian Test Symposium, 2002. pp. 92 – 97.

[12] A. J. van de Goor, Z. Al-Ars, "*Functional Memory Faults: A Formal Notation and a Taxonomy*", VTS 2000, 18th IEEE VLSI Test Symposium, 2000, pp. 281-289.

[13] A. J. van de Goor, "*Testing Semiconductor Memories: theory and practice*", Wiley, Chichester (UK), 1991

[14] D. Niggemeyer, M. Redeker, J. Otterstedt, "*Integration of non-classical faults in standard March tests*", MTDT 1998, IEEE International Workshop on Memory Technology, Design and Testing, 1998, pp. 91 -96

[15] A.V. Aho, R. Sethi, J. D. Ullman, "*Compilers: Principles, Techniques and Tools*", AddisonWesley, 1986.

[16] S. Hamdioui, Ad J. van de Goor, M. Rodgers, "*March SS: A Test for All Static Simple RAM Faults*", MTDT 2002: IEEE International Workshop on Memory Technology, Design and Testing, 2002 pp. 95 – 100