Memory read faults: taxonomy and automatic test generation

Authors: Benso A., Di Carlo S., Di Natale G., Prinetto P.,

Published in the Proceedings of the IEEE 10th AsianTest Symposium (ATS), 19-21 Nov. 2001, Kyoto, JP.

DOI: 10.1109/ATS.2001.990275

Memory Read Faults: Taxonomy and Automatic Test Generation

ALFREDO BENSO, STEFANO DI CARLO, GIORGIO DI NATALE, PAOLO PRINETTO
Politecnico di Torino
Dipartimento di Automatica e Informatica
Corso Duca degli Abruzzi 24 - I-10129, Torino, Italy
Email: { benso, dicarlo, dinatale, prinetto }@polito.it
http://www.testgroup.polito.it

Abstract

This paper presents an innovative algorithm for the automatic generation of March Tests. The proposed approach is able to generate an optimal March Test for an unconstrained set of memory faults in very low computation time. Moreover, we propose a new complete taxonomy for memory read faults, a class of faults never carefully addressed in the past.

1. Introduction

Memory devices play a crucial role in terms of *availability* and *serviceability* of electronic systems. They can appear in a variety of sizes, technologies (SRAMs, DRAMs, RamBus, etc.), and packaging (IP cores, chips, dedicated boards). The increasing scale of integration and the reduction of circuits' size and power supply levels make memories one of the most sensitive devices to *permanent* and *transient* faults caused by production process variations, environmental stresses, and interferences. This situation is mainly due to the reduced circuits size that require less energy to be damaged or change their state. The test of memory devices is therefore necessary to ensure the correct behavior of these components both at the end of production and during the product life cycle.

Among the algorithms proposed in the past to test random access memories (RAM), March Tests have proven to be faster, simpler, regularly structured, and linear in complexity [1].

March Tests are able to cover a wide range of memory faults such as Stuck-at-Faults (SAF), Transition Faults (TF), Stuck-Open Faults (SOF), Inversion and Idempotent Coupling Faults (CFin and CFid), Address Fault (AF), and Data Retention Faults (DRF). Several March Tests of variable complexity have been proposed in literature, each optimally covering a different set of memory faults. Despite this rich literature, the problem of detecting memory *read faults*, i.e., faults caused by a read operation on a memory cell, has never been analyzed in detail.

This paper presents a new methodology to automatically generate March Tests able to detect all known memory faults including read faults. Moreover, a new complete taxonomy for read faults is presented.

The paper is structured as follows: Section 2 summarizes the state of the art. Section 3 presents the model used to represent the good and fault memory behavior, whereas Section 4 details all the steps of the automatic March Test generation process. Section 6 presents experimental results reporting a set of new March Tests able to cover read faults. Section 7 summarizes the main contributions and future developments of this research.

2. State of the Art

In this section we focus on the detection of memory read faults only, since the problem has not been clearly addressed in the past.

A read fault is caused by a read operation performed on a memory cell. With the increasing scale of integration and the reduction of circuits' size and power supply levels, the importance of these kinds of faults is becoming more and more relevant. In [2] read faults have been split into the following two categories:

- *Read Disturb Faults (RDFs):* the content (d) of a memory cell (c) is inverted *during* a read operation, i.e. a read operation on a cell c containing the value d returns del value $\overline{d}$, and changes the content of the cell c from d to $\overline{d}$. This fault class can be further split into two subclasses: (i) *RDF*↑ when the initial state of the memory cell is 0 and the read operation change it to 1, (ii) *RDF*↓ when the initial state is 1 and the read operation change it to 0;

- *Deceptive Read Disturb Faults (DRDFs):* the content (d) of the memory cell (c) is inverted *as a result* of a read operation, i.e. a read operation on a cell c containing the value d returns the correct value d but the content of the cell c is corrupted to $\bar{d}$ at the and of the operation. Since the faulty transition happens *after* the read operation, this fault class is not directly detected by the initial read but it needs additional operations to be covered. Once again, there are two subclasses of these faults: (i) *DRDF*↑ and (ii) D*RDF*↓ depending on the initial state of the memory cell.

As a matter of fact, despite the automatic generation of March Tests has already been faced in [3], [4], and [5], only few papers deal with read faults. In [6] the authors mainly target the diagnosis of memory faults and use a fault description that allows modeling all possible single cell and two cells faults that occurs in memory arrays. This approach uses exhaustive search to find the best march test and thus it is very time expensive.

3. Memory Model

The problem of the automatic generation of March Tests needs the definition of a formal model able to represent the behavior of both the good and the faulty memory. This section presents a formal model to describe classical memory faults.

As proposed in [7] and [8], a n one-bit cells memory can be represented using a deterministic Mealy Automata $M = (Q, X, Y, \delta, \lambda)$, where Q is the set of possible *memory states*, X is the *input alphabet* composed by all the possible memory operations, Y is the output alphabet, and finally δ and λ are the *state transition* and the *output transition function*, respectively.

Using the proposed model, a fault free two cells RAM can be represented by the Finite State Machine (FSM) shown in Figure 1, conventionally named M_0 from now on. In M_0, the letters i and j are used to identify the first and the second cell, respectively. The use of an FSM for modeling the memory behavior allows easily describing a faulty RAM. A faulty memory can be modeled using an FSM differing from M_0 in the output and/or the transition function. The set of states used to represent a faulty memory is a subset of the whole set Q since only the cells involved in the fault should be represented.

This consideration makes possible to use the proposed model also for very large memories. The given representation is general enough to be used to model all known faults, including memory read faults. In the remainder of this section we propose a new classification of memory read faults using the above described behavioral memory model.

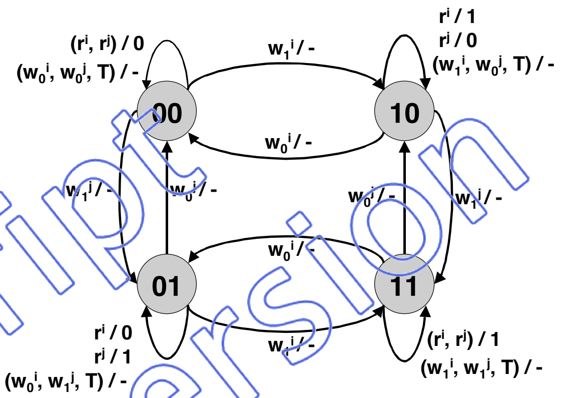

Figure 1: M_0 FSM representing a fault free RAM

The parameters used to define a class of read fault can be obtained by extending some well-known concepts:

- *Target Cell*: is the cell were the fault effect can be observed. It can be the cell on which the read operation is performed (*Single-Cell Read Fault*), or a different cell (*Read Coupling Fault*);
- *Excitation Value*: is the value of the read cell able to excite the fault. The fault can appear if the read cell value is equal to 0, to 1 or in both cases (*Any*);
- *Fault effect*: is the effect caused by the fault on the faulty cell. The cell can be inverted, forced to 0, forced to 1 or not changed;
- *Read Value*: is the value returned by the read operation. It can be 0, 1, the correct value (when the fault effect is inside another cell), the value of the cell before the fault effect (*deceptive*), or the value of the cell after the fault effect.

Combining these four independent parameters, it is possible to define the complete classification of the read fault classes shown in Table 1 and Table 2. Table 1 summarizes read faults from [2], whereas Table 2 shows some new faults classes.

Name of the fault	Target cell	Excitation Value	Fault Effect	Read Value
RDF	Same	Any	Inverted	Cell value after fault effect
DRDF	Same	Any	Inverted	Cell value before fault effect
RDF↑	Same	0	Inverted	Cell value after fault effect
RDF↓	Same	1	Inverted	Cell value after fault effect
DRDF↑	Same	0	Inverted	Cell value before fault effect
DRDF↓	Same	1	Inverted	Cell value before fault effect

Table 1: Read faults from [2]

Name of the fault	Target cell	Excitation Value	Fault Effect	Read Value
Read Error 0 (RE0)	Same	Any	Nothing	0
Read Error 1 (RE1)	Same	Any	Nothing	1
Read Stuck-At (RSA)	Same	Any	0/1	Cell value after fault effect
Deceptive Read Stuck-At (DRSA)	Same	Any	0/1	Cell value before fault effect
Read Coupling Inversion (RCIn)	Another	Any	Inverted	Correct value
Read Coupling Idempotent (RCId)	Another	Any	0/1	Correct value
Read Coupling Idempotent 0 (RCId0)	Another	Any	0	Correct value
Read Coupling Idempotent (RCId1)	Another	Any	1	Correct value

Table 2: New read faults

Considering as an example the Read Coupling Idempotent (RCId0) Fault, we obtain the FSM shown in Figure 2. As previously mentioned, since the fault involves two cells, only the cardinality of Q_i is four. The difference between the M_0 and M_1 machine is in the δ function, as pointed out by the four-bolded edges shown in Figure 2.

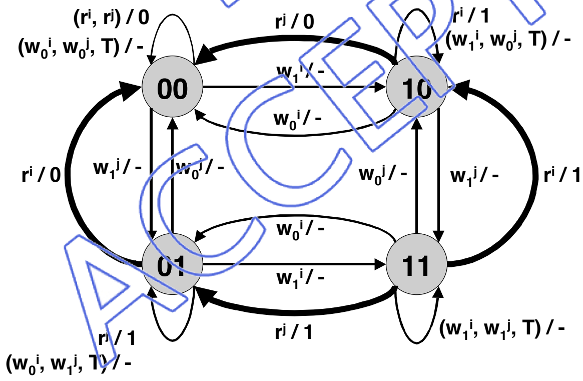

Figure 2: Read Coupling Idempotent 0 Fault Representation

Looking at M_1, we can split each fault into a set of *Basic Fault Effects (BFEs)* [3] [6]. A BFE_i can be described by a M_i FSM with a state transition function that differs from the one of M_0 by one transition only, or with a λ_i function that differs from the one of M_0 by one output value only. Using this formalism the example of Figure 2 generates four different BFEs, as shown in Figure 3. For the sake of simplicity only the relevant edges are represented.

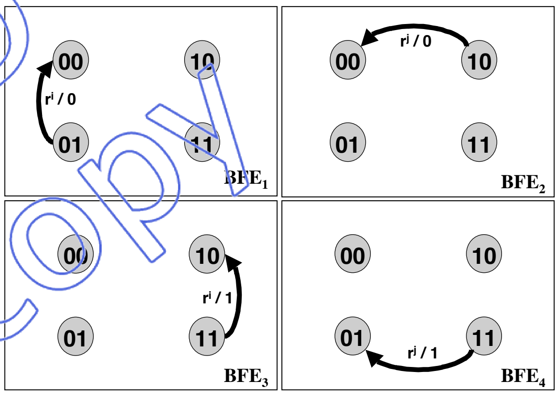

Figure 3: BFE model for Read Coupling Idempotent 0

Each BFE_i can be covered by generating a Test Pattern (TP_i) defined as a triplet $TP_i = (I, E, O)$ where I is the initial state of the memory, E is the operation needed to excite the fault, and O is the operation needed to observe the fault effects. For the proposed example, the four BFEs can be tested by the following four TPs: $TP_1 = \left(01, r_0^i, r_1^j\right)$, $TP_2 = \left(10, r_0^j, r_1^i\right)$, $TP_3 = \left(11, r_1^j, r_1^i\right)$, $TP_4 = \left(11, r_1^i, r_1^j\right)$

4. March Test Generation Algorithm

This section explains the algorithm used to automatically generate a March Test starting from the memory model proposed in Section 3. The algorithm, starting from an unconstrained list of target BFEs, generates a non-redundant March Test able to cover all of them.

The first phase analyzes the BFE list and, in particular, the set of TPs needed to cover each one of them. The analysis produces a strongly connected weighted graph named *Test Pattern Graph (TPG)* where each TPG node is associated with a TP. The *weight* of each edge represents the number of memory operations needed to reach the initialization state of the target node (S_T) starting from the observation state of the source node (S_S), i.e., it represents the Hamming distance between the initialization state and the observation state.

Figure 4 shows the TPG generated starting from the BFEs of Figure 3.

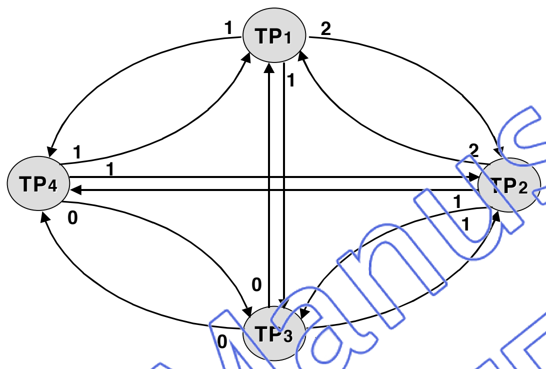

Figure 4: RCId0 TPG

From the TPG a *Global Test Sequence (GTS)* is built. A GTS is a set of memory operations able to detect all the target BFEs. Different GTSs can be obtained by simply concatenating the different TPs in multiple ways, i.e., visiting the TPG in different ways. The total number of possible GTS is $\#GTS = V!$ where V is the number of nodes in the TPG.

Since the space of all the possible GTSs is not manageable for very long fault lists, the algorithm exploits some heuristics to find the GTS able to generate a non-redundant March Test. In particular, the GTSs corresponding to minimum weight graph visits are selected; they are able to test the target faults with the minimum number of memory operations. The use of GTSs with minimum number of operation seems a good choice since there is a tight correlation between the GTS length and the March test complexity.

Using this heuristic, the generation of minimum length GTSs is a typical instance of the *Asymmetric Traveling Salesman Problem (ATSP)* [9]. The ATSP is a *combinatorial optimization (CO) problem*, for which a lot of heuristic and algorithms able to find solutions with a low computation time (especially for very small problems like the one discussed in this paper) can be found in literature. Referring to Figure 4, a possible ATSP solution produces the following GTS:

$$GTS_E = w_0^i, w_1^j, r_0^i, r_1^j, w_1^i, r_1^i, r_1^j, r_1^j, r_1^i, w_0^j, r_0^j, r_1^i$$

The GTSs obtained by the ATSP solution are able to test all the addressed BFE but are not yet March Tests. A March Test is a particular test sequence satisfying a set of constraints [1]. A GTS must be modified to transform it into an equivalent March Test.

The process of March Test generation from a GTS is performed in 4 steps:

- *GTS read-faults adjustment;*
- *GTS reordering;*
- *GTS minimization;*
- *March Test Generation.*

The GTS reordering, the GTS minimization, and the March Test Generation steps correspond to a different set of *Rewrite Rules* [10]. Since the GTS can be considered as a string where each symbol is a memory operation, the rewrite rules can be effectively represented resorting to the *Regular Expression* formalism [11].

For the sake of simplicity we define two subsets of instructions:

- $w = \{w_d^i, w_d^j\}$ is the set of possible memory write operations;
- $r = \{r_d^i, r_d^j\}$ is the set of possible memory read operations.

The regular expression formalism is extended introducing four new operators:

- *Read Excite Operator*: $[s]_E$ marks the symbol s as a read operation needed to excite a read fault (read-excite operation);
- *End Symbol Operator:* $\hat{s}$ marks the symbol s as not further modifiable (*terminal symbol*);
- *Red Operator:* $[s]_R$ marks the symbol s with the red color;
- *Blue Operator:* $[s]_B$ marks the symbol s with the blue color.

The use of colored symbols is useful during the March Test generation phase to identify the boundaries of the different March Elements. The next subsections summarize the rewrite rules used during the three different phases.

Pattern	*Rewrite Rule*
$(\hat{w} \mid \hat{r})* w_d^i w_d^j (w \mid r)*$	$w_d^i w_d^j \xrightarrow{M1} \hat{w}_d^i \hat{w}_d^j$
$(\hat{w} \mid \hat{r})* w_d^i w_d^i (w \mid r)*$	$w_d^i w_d^i \xrightarrow{M2} \hat{w}_d^i w_d^i$
$(\hat{w} \mid \hat{r})* w_d^i w_{\bar{d}}^j (w \mid r)*$	$w_d^i w_{\bar{d}}^j \xrightarrow{M3} \hat{w}_d^i w_{\bar{d}}^j$
$(\hat{w} \mid \hat{r})* \hat{r}_d^i \underbrace{(\hat{w}_d^i \mid \hat{w}_d^j \mid \hat{w}_{\bar{d}}^j)*}_{s1}\underbrace{(w_d^j \mid w_{\bar{d}}^j)*}_{s2} r_d^i (w \mid r)*$	$\hat{r}_d^i s_1 s_2 r_d^i \xrightarrow{M4} \hat{r}_d^i \lfloor \hat{r}_d^i \rfloor_R \lfloor s_1 s_2 \rfloor_B$

Table 3: Reordering Rewrite Rules

4.1. GTS Read-Faults Adjustment

This preliminary step is useful to mark the read operations that excite a read fault. This operation is not needed in case of a memory read fault-free model because there are not read operations able to excite a fault. The proposed algorithm reduces the complexity of the final march test exploiting this feature. In fact, this step is needed to avoid deleting excitation read during the minimization step (see Section 4.3) where sequences of consecutive read performed on the same cell are reduced to a single operation. In case of read operations able to excite a fault, this minimization must not be performed (i.e., if the first read operation inverts the content of the cell but the read value is correct, it is not possible to remove the second read operation).

The actual GTS is modified to reflect this constraint. Each read-excite operation followed by another read operation of a different value is marked with the *[ ]$_E$* operator. This operator will be taken into account during the following steps. By applying this step on the GTS proposed in Section 4 we obtain the following result: $GTS_E = w_0^i, w_1^j, r_0^i, r_1^j, w_1^i, \lfloor r_1^i \rfloor_E, r_1^j, \lfloor r_1^j \rfloor_E, r_1^i, w_0^j, r_0^j, r_1^i$

4.2. GTS Reordering

The reordering phase reorders the GTS memory instructions taking into account the constraints needed to obtain a March Test [1]. In this phase each modification is defined by a *Pattern* and by a *Rewrite Rule* (see Table 3). The pattern is a regular expression that identifies all the strings on which the rewrite rule must be applied. The reordering process stops when all the GTS symbols are modified into terminal ones. Appling the reordering rules on the GTS$_E$ we obtain the following reordered sequence:

$$GTS_R = \hat{w}_0^i, \lfloor \hat{r}_0^j \rfloor_R, \lfloor \hat{w}_1^j \rfloor_B, \hat{r}_1^j, \hat{w}_1^i, \lfloor \hat{r}_1^i \rfloor_E,$$
$$\hat{r}_1^j, \lfloor \hat{r}_1^j \rfloor_E, \hat{r}_1^i, \lfloor \hat{r}_1^j \rfloor_R, \lfloor \hat{w}_0^j \rfloor_B, \hat{r}_0^j$$

4.3. GTS minimization

The minimization phase deletes redundant subsequences to consider the minimum set of needed operations only. The rewrite rules applied in this phase consider the GTS starting from left to right (see Table 4). This phase is repeated until no further minimization can be applied. In this context the $ symbol is used to denote the end of the GTS and the color of the symbols (see Section 4) does not affect the application of the rules.

Rewrite Rules	
$\hat{w}_d^i \hat{w}_d^j \xrightarrow{R1} \hat{w}_d^i$	$\hat{r}_d^i \hat{r}_d^j \xrightarrow{R1} \hat{r}_d^i$
$\hat{w}_d^i \hat{w}_d^i \xrightarrow{R2} \hat{w}_d^i$	$\hat{r}_d^i \hat{r}_d^i \xrightarrow{R2} \hat{r}_d^i$
$\hat{r}_d^i \hat{w}_{\bar{d}}^i \hat{w}_d^i \hat{r}_d^j \hat{w}_{\bar{d}}^j \hat{w}_d^j \xrightarrow{R3} \hat{r}_d^i \hat{w}_{\bar{d}}^i \hat{w}_d^i \hat{r}_d^j$	
$\hat{r}_d^i \hat{w}_{\bar{d}}^i \hat{w}_d^i \hat{r}_d^j \hat{w}_{\bar{d}}^j \$ \xrightarrow{R3bis} \hat{r}_d^i \hat{w}_{\bar{d}}^i \hat{w}_d^i \hat{r}_d^j \$$	
$\lfloor \hat{r}_d^i \rfloor_E \lfloor \hat{r}_d^j \rfloor_E \xrightarrow{R4} \lfloor \hat{r}_d^i \rfloor_E$	
$\lfloor \hat{r}_d^i \rfloor_E \hat{r}_d^j \xrightarrow{R5} \lfloor \hat{r}_d^i \rfloor_E$	$\hat{r}_d^i \lfloor \hat{r}_d^j \rfloor_E \xrightarrow{R5bis} \lfloor \hat{r}_d^j \rfloor_E$
$\hat{w}_d^i \lfloor \hat{r}_d^i \rfloor_E \xrightarrow{R6} \lfloor \hat{r}_d^i \rfloor_E$	$\lfloor \hat{w}_d^i \rfloor_B \lfloor \hat{r}_d^i \rfloor_E \xrightarrow{R6bis} \lfloor \hat{r}_d^i \rfloor_E$

Table 4: Modification Rewrite Rules

Appling several times the minimization rewrite rules on the reordered *GTS$_R$* (see Section 4.2) we obtain the following minimal sequence:

$$GTS_R = \hat{w}_0^i, \lfloor \hat{r}_0^j \rfloor_R, \lfloor \hat{w}_1^j \rfloor_B, \hat{r}_1^j, \hat{w}_1^i, \underbrace{\lfloor \hat{r}_1^i \rfloor_E, \hat{r}_1^j}, \underbrace{\lfloor \hat{r}_1^j \rfloor_E, \hat{r}_1^i}, \lfloor \hat{r}_1^j \rfloor_R, \lfloor \hat{w}_0^j \rfloor_B, \hat{r}_0^j$$

$$\Big\downarrow \text{R5} \qquad \Big\downarrow \text{R5}$$

$$GTS_M^I = \hat{w}_0^i, \lfloor \hat{r}_0^j \rfloor_R, \lfloor \hat{w}_1^j \rfloor_B, \hat{r}_1^j, \hat{w}_1^i, \underbrace{\lfloor \hat{r}_1^i \rfloor_E, \lfloor \hat{r}_1^j \rfloor_E}, \lfloor \hat{r}_1^j \rfloor_R, \lfloor \hat{w}_0^j \rfloor_B, \hat{r}_0^j$$

$$\Big\downarrow \text{R4}$$

$$GTS_M^{II} = \hat{w}_0^i, \lfloor \hat{r}_0^j \rfloor_R, \lfloor \hat{w}_1^j \rfloor_B, \hat{r}_1^j, \underbrace{\hat{w}_1^i, \lfloor \hat{r}_1^i \rfloor_E}, \lfloor \hat{r}_1^j \rfloor_R, \lfloor \hat{w}_0^j \rfloor_B, \hat{r}_0^j$$

$$\Big\downarrow \text{R6}$$

$$GTS_M^{III} = \hat{w}_0^i, \lfloor \hat{r}_0^j \rfloor_R, \lfloor \hat{w}_1^j \rfloor_B, \underbrace{\hat{r}_1^j, \lfloor \hat{r}_1^i \rfloor_E}, \lfloor \hat{r}_1^j \rfloor_R, \lfloor \hat{w}_0^j \rfloor_B, \hat{r}_0^j$$

$$\Big\downarrow \text{R6bis}$$

$$GTS_M = \hat{w}_0^i, \lfloor \hat{r}_0^j \rfloor_R, \lfloor \hat{w}_1^j \rfloor_B, \lfloor \hat{r}_1^i \rfloor_E, \lfloor \hat{r}_1^j \rfloor_R, \lfloor \hat{w}_0^j \rfloor_B, \hat{r}_0^j$$

4.4. March Test Generation

This last phase uses the minimized GTS$_M$ to generate a March Test. Before applying any rules to convert the GTS into a March Test, the minimized GTS$_M$ is modified removing the marker $[]_E$. The new input sequence is then analyzed from left to right and the March Elements are generated according to the following rules:

- *Rule 1:* subsequences identified by $(\hat{w}_d^i \mid \hat{r}_d^j)(\hat{w}_d^j \mid \hat{r}_d^j)$ regular expression close a March Element and open a new one;
- *Rule 2:* subsequences identified by $[\hat{r}]_R([\hat{w}]_B *)$ regular expression are joined in a single March Element despite they are execute on cell i or on cell j. The last blue marked operation closes the March Element.

The addressing order is generated using the following rules:

- *Rule 3*: March Elements starting with colored operation performed on *i* cells have addressing order $\Uparrow$;
- *Rule 4:* March Elements starting with colored operation performed on *j* cells have addressing order $\Downarrow$;
- *Rule 5:* March Elements starting with non-colored operations have address order $\Updownarrow$.

Applying the generation rules on the *GTS$_M$* (see Section 4.3) we obtain the following *7n* non-redundant March Test: $M = \Updownarrow w_0 \Downarrow r_0 w_1 \Uparrow r_1 \Downarrow r_1 w_0 \Updownarrow r_0$

5. Experimental Results

This section reports experimental results in generating March Tests using the proposed approach, considering sets of both well known faults and read faults never used in previous researches on memory testing. The algorithm has been implemented in about 5000 lines of C code. The ATSP has been solved using a Fortran code able to give an exact solution to the problem [12]. The computation time needed to generate the March Tests is not reported since it is negligible (all the experiments last less than 1 second).

All generated March Tests have been verified using an ad hoc memory fault simulator able to validate the correctness of them given the list of target BFE and able to check the non-redundancy of the given test algorithm. Table 5 and 6 show a set of March Test automatically generated starting from different sets of memory fault classes.

The fault simulator has been also used to gather additional information about faults covered by the algorithm but not directly inserted in the BFE list as pointed out in Table 6. In the table the bigger bullets identify the fault targeted by the generation algorithm, whereas the smeller bullets show the additional faults detected by the obtained March Test.

6. Conclusion

Memories are among the most critical devices in terms of *availability* and *serviceability* of electronic systems. The number of possible defects that can appear in a memory array increases with the advances in the manufacturing technology. The high scale of integration and the reduction of the power supply levels make memories very sensitive to *permanent* and *transient* faults. Many fault models, such as Stuck-at-Faults, Transition Faults, Stuck-Open Faults, Coupling Faults, Address Fault, and Data Retention Faults, have been introduced to cover the wide range of possible defects.

In this paper a general model to represent memory faults is presented. It includes a complete classification of memory read faults, a class of faults never carefully addressed in previous researches. Moreover, the papers presents an automatic March Test generation process able to cover all of them. Experimental results are reported to demonstrate the effectiveness of the approach in generating March Tests in a very low computation time.

7. References

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

[2] S. Hamdioui, Ad. van de Goor, "March Tests for Word-Oriented Two-Port Memories", 8th Asian Test Symposium, 1999, pp. 53-60.

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

[4] A. J. van de Goor, B. Smit, "Automatic the Verification of March Tests",IEEE VLSI Test Symposium, pp. 312-318, 1994

[5] A. J. van de Goor, B. Smit, "The Automatic Generation of March Tests", IEEE International Workshop Memory Technology, pp. 86-91, 1994

[6] D. Niggemeyer, M. Redeker, E. M. Rudnick, "Diagnostic Testing of Embedded Memories based on Output Tracing", IEEE International Workshop Memory Technology, pp. 113-118, 2000

[7] J.A. Brzozowski, H. Jurgensen "A Model for Sequential Machine Testing and Diagnosis" J. Electronic Testing: Theory and Application, Vol. 3, No. 3, pp. 219-234, August 1992

[8] J.A. Brzozowski, B.F. Cockburn "Detection of Coupling Faults in RAMs" J. Electronic Testing: Theory and Application, Vol. 1, No. 2, pp. 151-162, May 1990.

[9] A. Gibbons, "Algorithmic Graph Theory", Cambridge University Press 1985.

[10] S. Even, I. Kohavi, A. Paz, "On minimal module-2 sum of products for switching functions", IEEE Trans. Electron. Comput., vol. EC-16, pp. 671-674, Oct. 1967.

[11] G. Rozemberg, "Handbook of Graph Grammars and Computing by Graph Transformation", Vol. I: Foundation, World Scientific, 1997.

[12] G.Carpaneto, E. Dell'Amico, I. Toth, "A Branch-and-Bound Algorithm for large scale Asymmetric Traveling Salesman Problems", Technical Report Dipartimento di Economia Politica, Facoltà di Economia e Commericio, Modena University 1990, ftp://netlib2.cs.utk.edu/toms/index.html ,"ACM Collected Algorithms no. 750", 1994.

Fault List					**Generated March Tests and their complexity**		**Equivalent Known March Test**
SAF	**TF**	**ADF**	**CFin**	**CFid**			
•					$\{\Uparrow w_1 \Downarrow r_1 w_0 \Downarrow r_0\}$	$4n$	MATS ($4n$)
•		•			$\{\Uparrow w_1 \Uparrow r_1 w_0 \Downarrow r_0 w_1\}$	$5n$	MATS+ ($5n$)
•	•	•			$\{\Uparrow w_0 \Uparrow r_0 w_1 \Downarrow r_1 w_0 \Uparrow r_0\}$	$6n$	MATS++ ($6n$)
•	•	•	•		$\{\Uparrow w_0 \Downarrow w_1 \Downarrow r_1 w_0 \Uparrow r_0 w_1\}$	$6n$	MarchX ($6n$)
•	•	•	•	•	$\{\Uparrow w_1 \Uparrow r_1 w_0 \Uparrow r_0 w_1 \Downarrow r_1 w_0 \Downarrow r_0 w_1 \Downarrow r_1\}$	$10n$	March C- ($10n$)
			•		$\{\Uparrow w_0 \Uparrow r_0 w_1 w_0 \Downarrow r_0\}$	$5n$	Not Found

Table 5: Experimental Results

Fault List								**Generated March Test**	
RSA	DRSA	RDF	DRDF	RCIn	RCId	RCId0	RCId1		**Complexity**
●		•						$\{\Uparrow w_0 r_0 w_1 r_1\}$	$4n$
•	●	•	•					$\{\Uparrow w_0 r_0 r_0 w_1 r_1 r_1\}$	$6n$
		●	•					$\{\Uparrow w_0 r_0\}$	$2n$
			●					$\{\Uparrow w_0 r_0 r_0\}$	$3n$
•		•		●				$\{\Uparrow w_0 \Uparrow r_0 \Downarrow r_0 w_1 \Downarrow r_1\}$	$5n$
•	•	•	•	•	●	•	•	$\{\Uparrow w_0 \Uparrow r_0 w_1 r_1 \Downarrow r_1 \Uparrow r_1 w_0 r_0 \Downarrow r_0\}$	$9n$
•		•		•		●		$\{\Uparrow w_0 \Uparrow r_0 \Downarrow r_0 w_1 \Downarrow r_1 w_0 r_0\}$	$7n$
•		•		•			●	$\{\Uparrow w_1 \Uparrow r_1 \Downarrow r_1 w_0 \Downarrow r_0 w_1 r_1\}$	$7n$

Table 6: Experimental Results