Specification and design of a new memory fault simulator

# Specification and design of a new memory fault simulator

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

Published in the Proceedings of the IEEE 11th AsianTest Symposium (ATS), 18-20 Nov. 2002, Guam, USA.



**DOI: 10.1109/ATS.2002.1181693**

# SPECIFICATION AND DESIGN OF A NEW MEMORY FAULT SIMULATOR

A. Benso, S. Di Carlo, G. Di Natale, P. 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
www.testgroup.polito.it



**Abstract**

*This paper presents a new Fault Simulator architecture for RAM memories. The key features of the proposed tool are: 1) user-definable fault models, test algorithm, and memory architecture; 2) very fast simulation algorithm; 3) ability to compute the coverage of any provided test sequence w.r.t. a user-defined set of fault models, and to eliminate redundant operations; 4) assessment of the power consumption generated by the test application. Moreover, the tool is able to modify the test algorithm in order to guarantee the compliance to user-defined power consumption constraints.*

## 1. Introduction

RAM memories are widely considered to be one of the most critical components in digital systems. Not only, according to Moore's Law [1], it is expected that the capacity of memory will quadruplicate every three years, but the shrinking of the technology will make memories more and more subject to faults.

The main issue in memory testing is to define comprehensive fault models able to carefully represent the most common defects occurring in the production phase of the chips. Along with fault models, new test algorithms have to be developed and validated. Memory fault simulation is therefore necessary to compute the Fault Coverage of a test sequence every time a new defect is discovered and the corresponding fault model defined. Computing and limiting the test application power consumption is also another important issue, especially when the memory test is implemented as a Built-In Self-Test procedure.

Due to the complexity of both the fault models and the memory architecture, manual analysis [2] of the memory fault coverage is not anymore possible. In [3], a memory simulator (Memory Animation Package Plus, MAP+) has been proposed. This tool, developed at the Delft University of Technology, has been employed as a simulation tool for the evaluation of new and known test algorithms in presence of different faults (such as stuck-at, transition, coupling, address decoder, neighborhood pattern sensitive, read disturb faults, etc.). Although very interesting especially from an academic point of view, this tool does not allow a very detailed fault simulation.

In this paper we present the architecture for a new flexible memory fault simulator, designed to address all the most critical issues in today's memories test generation and validation. Besides the fault coverage computation, already addressed by other similar tools [4], the proposed simulator supports the test engineer in optimizing the test algorithm and in addressing power consumption constraints. The tool is in fact able to compute the power consumption generated by the test sequence, and to suggest a modification of the test algorithm in case its application does not fulfill a user-defined power consumption constraint.

The paper is organized as follows. Section 2 introduces the overall tool architecture; Section 3 and 4 describe the memory and fault model representation used by the simulator. The format of the test sequence is described in Section 5, whereas Section 6 and 7 detail the fault simulation engine. Some experimental results are shown in Section 8. Conclusions and future work are summarized in Section 9.

## 2. The Fault Simulator Architecture

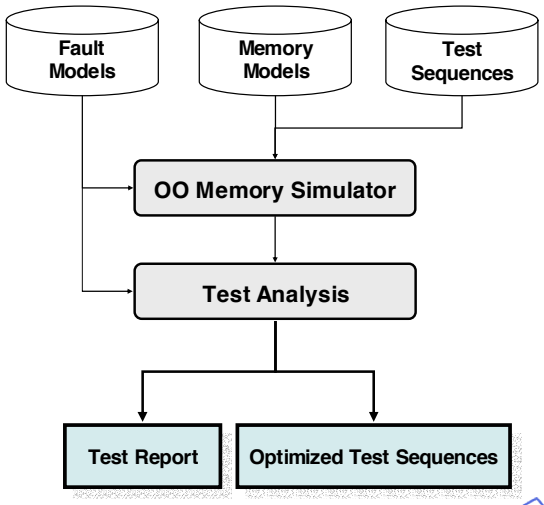


*Figure 1: Simulator architecture*

Figure 1 presents the simulator overall architecture. The *Object Oriented Memory Simulator* reads three main input files. The first two contain the memory functional and electrical models whereas the third the test sequences. The tool simulates the execution of the test sequence according to the models and stores, for each memory cell, the logical and electrical temporal evolution. After the simulation, a Test Analysis module reads the target Fault Model files and computes their coverage w.r.t. the test sequence. It generates two output files storing a detailed test report, and, whenever possible, an optimized test sequence able to provide the same results of the original one.

The memory model is split in two parts:

- a *functional model*, represented as a Finite State Machine (see Section 3);
- an *electrical and physical model*, storing all the operating, technological, and topological characteristics of the memory (see Section 3.1);

The Fault Model files, formalized as collections of Basic Fault Effects (see Section 4), describe all the faulty behavior that the test sequence is designed to detect.

The Test Sequence files describe the sequence of operations applied to test the memory array. Using a proprietary language, it is possible to describe complex test algorithms as well as simple sequences of input patterns.

The Test Report file contains detailed information about:

- the Fault Coverage of each fault model and, when necessary, diagnostic information about the cells where the fault is not covered;
- the total power consumption estimated by the application of the test sequences;
- if the computed power consumption is higher than the allowed limit, if possible, the simulator provides the suggested maximum clock frequency that allows to meet the power requirements;

Finally, the Test Analysis module outputs an optimized test sequence, where redundant elementary operations not affecting the final fault coverage are removed.

In the following sections we will detail the different models introduced in this section, focusing in particular on the memory and fault models.

## 3. Memory model

The proposed memory fault simulator uses a memory functional model based on Finite State Machines (FSM), as proposed in [5] and [6].

An $n$ one-bit cells memory can be represented using a deterministic Mealy Automata:

$$M = (Q, X, Y, \delta, \lambda) \qquad \text{(f.2.1)}$$

where:

- $Q = \{0,1,-\}^n$ is the set of the possible *memory states* where the symbol $(-)$ represents the value of a non initialized memory cell;
- $X = \{r^i, w_0^i, w_1^i \mid 0 \le i \le n-1\} \cup \{T_t\}$ is the *input alphabet*. This alphabet is composed by all the possible memory operations. In particular:

  $r^i$ corresponds to a read operation performed on the cell $i$;

  $w_d^i$ corresponds to a write operation of the value $d \in \{0,1\}$ performed on the cell i;

  $T_t$ corresponds to a wait operation for a defined period of time $t$. This additional element is needed to deal with Data Retention Faults [7].
- $Y = \{0,1,-\}$ is the *output alphabet*;
- $\delta : Q \times X \mapsto Q$ is the *state transition function*;
- $\lambda : Q \times X \mapsto Y$ is the *output function*.

Using the outlined model, a fault free two cells RAM can be represented by the FSM shown in Figure 2, conventionally named $M_0$ in the reminder of this paper. In $M_0$, the letters *i* and *j* are used to identify the first and the second cell, respectively.

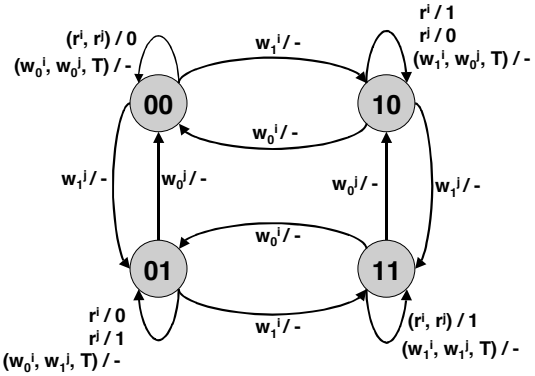


*Figure 2: $M_0$ FSM representing a fault free RAM*

### 3.1. Electrical and Physical Model

Besides the memory behavior, the user can specify a set of electrical parameters that constitute the memory electrical and physical model. In particular, it is possible to specify:

1. the typical *operating conditions* (e.g., supply voltage, operating temperature, …): these model is included in the simulator for future developments, where we will take into account not only the functional behavior of the memory but also its operating conditions;
2. the actual row/column topological organization of the memory array: this information is necessary to compute the coverage of faults involving adjacent cells;
3. timing and electrical characteristics (e.g., access time, operating and stand-by current, …): these characteristics allow the simulator to compute different parameters as the average power dissipation caused by the application of a given test algorithm.

## 4. Fault Models

Considering the fault-free memory formalization, the behavior of a faulty memory can be modeled using a deterministic Mealy Automata:

$$M_i = (Q_i, X, Y_i, \delta_i, \lambda_i) \qquad \text{(f.2.2)}$$

where:

- $Q_i \subseteq Q$ is the set of states;
- $Y_i \subseteq Y$ is the output alphabet;
- $\delta_i : Q_i \times X \mapsto Q_i$ is the state transition function
- $\lambda_i : Q_i \times X \mapsto Y_i$ is the output function

The set of states used to represent a faulty memory is a subset of the whole set Q (see (f.2.1)) since only the cells involved in the fault should be represented. A faulty memory functional model can be therefore described by a $M_i$ FSM with a $\delta_i$ function that differs from $\delta_0$ by one transition only, or with a $\lambda_i$ function that differs from $\lambda_0$ by one output value only. The $M_i$ FSM is called *Basic Fault Effect* (*BFE*) [8], [9] and each fault model can be represented with a set of BFEs.

The two-cell memory model presented in Section 3 is general enough to model the most well-known memory faults like stuck-at faults, coupling faults, transition faults, and address faults. To model more complex faults like neighborhood pattern sensitive faults (*npsf*), the model is extended to include the minimum number of cells involved in the faulty memory behavior. The simulator will then verify the coverage of the fault model for every possible placement of the cells involved in the fault. This consideration makes possible the use of the proposed model for very large memories, since the proposed tool will always simulate the minimum number of cells required to represent each target fault model. The results obtained on the minimum cell set are then extended to the target memory size.

To cover a BFE on a memory cell, it is necessary to execute on that cell a sequence of operations, or Test Pattern (TP), defined as a triplet:

$$TP = (I, E, O) \qquad \text{(f.2.3)}$$

where:

- $I = \{(0,1)^k \mid 0 \le k \le n-1\}$ is the *initialization state*;
- $E = \{e \mid e \in X\}$ is the operation needed *to excite* the BFE;
- $O = \{r_d^k \mid d \in (0,1), 0 \le k \le n-1\}$ is the operation needed *to observe* the fault effect. We introduce here the concept of *Read and Verify* operation. The notation $r_d^i$ means "read the content of the cell *i* and verify that its value is equal to *d*".

Besides Read and Verify operations, it is also possible to specify verification and modification of the memory *operating conditions*; these operations may be necessary when modeling faults occurring in particular operating conditions.

Initialization, Excitation, and Observation instructions have to be executed in a fixed order but, depending on the fault model, they may or may not have to be executed consecutively. For example, it is possible to specify the maximum delay between two consecutive operations on a cell of the memory.

Given the above definitions, the simulator considers each BFE as a set of Test Patterns. To verify the coverage of a given fault model, the simulator has to check that all the operations have been executed on all the memory cells.

## 5. Test Sequences

To define the memory test sequences we defined a language allowing to describe complex test algorithms as well as simple sequences of input patterns.

In particular, the language constructs have been defined in order to facilitate the description of:

- Simple *Read* or *Write* operations scheduled at a given time;
- *March elements*: set of instructions repeated on all the cells of the memory array [2];
- *Burst cycles*: a single instruction to be executed on consecutive memory cells;
- *Neighborhood cells*: a set of neighborhood cells on which executing a given set of operations;
- *Transparent Tests*: write operations where the written value is a function of the memory cell content;
- *Background patterns*: the set of patterns to be used to test word-oriented memories;
- *C-like* statements: *for* cycles, *if-then-else* constructs, and evaluation of simple Boolean expressions that allow the definition of complex test algorithms;
- *Changes in the operating conditions*: operations that, for example, simulate a change of the operating temperature of the memory. These operations have been introduced to allow, in the future, the modeling of fault depending on the memory operating conditions.

Figure 3 presents an example of part of the Walking 1/0 algorithm described using the proposed language.

```
// Walking 1/0 (first part)
walking10::    any ( w 0 );
for ( i = 0; i < words_in_array; i = i + 1 )
{
                w [i] 1;
      neigh array any [i] ( r 0 );
      r [i] 1;
      w[i] 0;
}
```

*Figure 3: Example of test algorithm including Neighborhood cells and a cycle statement*

## 6. Simulator Engine

The proposed memory fault simulator has been designed using a layered object oriented approach. The models describing the memory, the faults, and the test sequences are classes with a predefined set of methods and properties.

The simulator engine is designed using an onion skin-like approach (Figure 4), where each layer targets a different functionality of the simulation: access to the memory array, the electrical behavior, the temporal behavior, and the I/O behavior. This approach allowed us to design an efficient, modular, and very easily upgradeable tool. The only constraint of each layer is its interface; any layer internal behavior or structure can be redesigned, modified, or upgraded without redesigning the whole simulator.

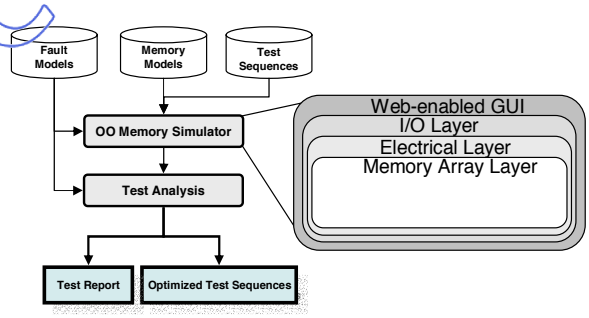


*Figure 4: The simulator engine layered structure*

The *Graphical User Interface* (GUI) of the simulator is completely web-enabled. This feature allows us to have a user-platform-independent tool, which is always up-to-date with the most recent version, and does not require any installation process. The user does not require a high computational power on its machine, and can easily access the tool from any internet connection point. This

characteristic is particularly useful in an academic environment, where students always outnumber the available workstations.

The *I/O Layer* implements the possible I/O operations on the memory. The allowed operations, with their temporal constraints, are read from the memory model. In this way, the simulator is also able to check if all the operations executed by the test algorithm are legal for the target memory chip. The I/O Layer is also in charge of reading and applying the test sequence defined in one of the input files. In order to optimize the simulation time, the simulator computes from the Fault Model files and the input test files, the smallest number of memory cells that guarantee the ability to compute the final result for the real memory size. For example, as explained in Section 2, if the target fault models are stuck-at and coupling faults, and the test sequence a March test, the minimum number of cells that need to be simulated is equal to 2.

The *Electrical Layer* computes and logs the temporal evolution of the electrical and physical characteristics of each cell during the application of the test sequence. At each instant, the state of a cell is defined as the voltage level of the cell plus a *timeout*, after which the voltage of the cell has to be re-evaluated even if the value of the cell is not changed by an external operation. A transition to another state is triggered by an event. Possible events are: a read or write operation on a cell, the expiration of the timeout of the cell, or the variation of the memory operating conditions. The user can define the behavior of the memory cells when a transition is triggered. For example, it is possible to define the function that computes the voltage level of a cell upon the expiration of a timeout. This layer can be made transparent (and therefore disabled) if the electrical evolution of the memory is not required by the user.

Finally, the *Memory Array Layer* is used to log only the logical evolution of the memory content. This layer considers the memory as a matrix of words as defined by the memory model.

## 7. Test Analysis Module

For each Fault Model, the *Test Analysis Module* considers each set of Test Patterns defining a BFE, and verifies, using a pattern matching algorithm, if it has been executed during the simulation of the test sequence. If all the Test Patterns belonging to a Fault Model have been executed on a cell, then, for that cell, the fault is covered. Besides fault coverage, the Test Analysis module also computes the total power consumption caused by the application of the test sequences.

An interesting feature of the Test Analysis Module is its ability to suggest optimizations to the input test algorithm. In particular, it is able to:

- check the non-redundancy of each *elementary operation* in the test sequence, and suggest a possible optimization;
- suggest a possible modification of the test sequence in order to fulfill the power consumption constraints.

To check the non-redundancy of each *elementary operation*, the Test Analysis Module builds a *Coverage Matrix* where each row represents the elementary operations whereas the columns the target fault models. A matrix cell is set to the value one if the corresponding elementary operation contributes to the coverage of the fault represented by the column. A test sequence is able to detect all the target faults if, for each CF column, exists at least one row containing a cell set to one. The test sequence is non-redundant if all the matrix rows are needed to cover the target faults. If this is not the case, the module outputs a new test sequence trimmed of all the redundant elementary operations. This is a typical instance of the *Set Covering* problem applied on the Coverage Matrix [10]. The Set Covering finds the minimum number of rows needed to cover all the columns. This approach has been successfully applied on many known March Tests, where redundant blocks have never been found [11].

Finally, to address power consumption, the Test Analysis Module is able to compute either the maximum clock frequency that allows meeting the power constraints, or, if the clock is not modifiable, it inserts *delay* instructions in the test sequence.

Modern approaches to reduce power consumption like supply voltage reduction or multiple voltage nets [12] are not in the scope of the proposed tool, since we do not deal with hardware modification of the memory under test.

## 8. Experimental results

To evaluate the performances of our simulator, we setup the same set of experiments described in [13]. In particular, we run 8 different march tests (MATS+, March C-, March B, PMovi, March U, March LR, March SR, and March SS) on a memory of 32Kbit targeting a fault list of 18 fault models (Stuck-At, Transition, Write Disturb, Read Destructive, Deceptive Read Destructive, Incorrect Read, 10 different Coupling, and Data Retention).

The execution time of the simulator on a Pentium II, 400MHz with 64MB of RAM is reported in Table 1.

| | MATS+ | C- | B | PMovi | U | LR | SR | SS |
|---|---|---|---|---|---|---|---|---|
| Time (s) | 1.65 | 6.44 | 6.35 | 2.25 | 4.70 | 4.58 | 2.46 | 17.27 |

*Table 1: Execution time*

For each fault, the simulator shows the test operations allowing its coverage. Table 2 shows a subset of the input and result files. The Stuck-At-0 fault model is described in terms of initialization (I), excitation and observation (EO) whereas the test algorithm in terms of march elements. The result file shows the elementary test operations that allow covering the target fault. In this example, the initialization is covered by the second operation of the *m1* march element and the excitation/observation is obtained by the first operation of the march element *m2*.

| **Fault Model** | **March Test** |
|---|---|
| `SF_0`<br>`{`<br>`  I::  w 1;`<br>`  EO::  r 1;`<br>`}` | `// MATS+`<br>`{`<br>`  m0:: any  (w 0);`<br>`  m1:: up  (r 0, w 1);`<br>`  m2:: down (r 1, w 0);`<br>`}` |
| **Results** | |
| `SF_0  100.00 %  of 32768`<br>`----`<br>`I  ->  m1.2`<br>`EO  ->  m2.1` | |

*Table 2: Result Table*

## 9. Conclusions

In this paper we presented the structure of a new memory fault simulator. The proposed tool addresses the problem of the efficient and fast validating of memory test algorithm w.r.t. new fault models or new memory structures.

The main features of the proposed tool are its modularity, its flexibility in the description of the memory and fault models, and its ability to suggest optimizations to the input test algorithm.

Currently our activity is focused in the implementation of a full functional version of the tool. The future activities will address its testing and the expansion of the Simulation and Test Analysis modules in order to cover multi-port memories.

## 10. Acknowledgments

We would like to thank Prof. A. J. van de Goor for the invaluable suggestions he gave us during the specification of the tool. We would also like to thank the web-enabled technologies division and the test division of MoleSystems, for their contribution to the development of the tool.

## 11. Bibliography

[1] G. E. Moore, Progress in digital integrated electronics, In Proc. IEEE IEDM, pages 11-13, 1975

[2] A. J. van de Goor, Testing Semiconductor Memories: Theory and Practice, John Wiley & Sons, Chichester, England, 1991.

[3] S. Demidenko, A. van de Goor, S. Henderson, P. Knoppers, Simulation and Development of Short Transparent Tests for RAM, IEEE 10th Asian Test Symposium (ATS 2001), Kyoto (J), November 2001

[4] C. Wu, C. Huang, C. Wu, RAMSES: a fast memory fault simulator, International Symposium on Defect and Fault Tolerance in VLSI Systems, 1999, Page(s): 165 -173

[5] 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.

[6] 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

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

[8] 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

[9] 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

[10] A. Caprara, M. Fischetti, P. Toth, A Heuristic Algorithm for the Set Covering Problem, 5th International IPCO Conference, pp. 72-84, 1996.

[11] A. Benso, S. Di Carlo, G. Di Natale, P. Prinetto, An Optimal Algorithm for the Automatic Generation of March Tests, IEEE Design Automation and Test Conference in Europe (DATE 2002), Paris (F), February 2002

[12] J. Vollrath, M. Huebl, E. Stahl, Power Analysis of DRAMs, 7th Asian Test Symposium (ATS'98), pp. 334-339, 1998.

[13] S. Hamdioui, A. J. Van De Goor, M. Rodgers, March SS: a Test for All Static Simple RAM Faults, Memory Technology, Design and Testing Workshop (MTDT'02), pp. 95-100, 2002.