Automating defects simulation and fault modeling for SRAMs

Stefano Di Carlo, Paolo Prinetto, Alberto Scionti
Politecnico di Torino
Control and Computer Engineering Department
Torino, Italy
Email: {stefano.dicarlo, paolo.prinetto,alberto.scionti}@polito.it

Zaid Al-Ars
Delft University of Technology
Delft, The Netherlands
Email: z.al-ars@tudelft.nl

Abstract—**The continuos improvement in manufacturing process density for Very Deep Sub Micron technologies constantly leads to new classes of defects in memory devices. Exploring the effect of fabrication defects in future technologies, and identifying new classes of realistic functional fault models with their corresponding test sequences, is a time consuming task up to now mainly performed by hand. This paper proposes a new approach to automate this procedure. The proposed method exploits the capabilities of evolutionary algorithms to automatically identify faulty behaviors into defective memories and to define the corresponding fault models and relevant test sequences. Target defects are modeled at the electrical level in order to optimize the results to the specific technology and memory architecture.**

I. Introduction

Semiconductor memories are the predominant majority of semiconductor devices production, and have been used for a long time to push the state-of-the-art in the semiconductor industry. The Semiconductor Industry Association (SIA) forecasts that in the next 15 years up to 95% of the entire chip area will be used to create memory blocks [1]. Precise fault modeling and efficient test design are therefore key elements to keep test cost and time within economically acceptable limits.

Complex Functional Fault Models (FFM) together with very efficient test algorithms such as march tests are a very effective solution to deal with emerging classes of memory defects [2], [3]. Several algorithms are available in the literature to automate the generation of march tests for a target set of FFMs [4], [5], [6], [7], [8], [9].

While traditional memory FFMs (e.g., stuck-at faults, coupling faults, address faults, etc.) are independent of the specific memory technology and architecture, the continuos improvement in manufacturing process density for Very Deep Sub Micron (VDSM) technologies leads to new classes of defects, in most cases tightly linked to the internal structure and technology of the target memory [10], [11], [12], [13]. Exploring the effect of fabrication defects in future technologies by means of simulations, and identifying new classes of realistic functional fault models with their corresponding test sequences, is one of the most time consuming tasks in defect-oriented testing. The identification of automatic solutions for this complex task is therefore a challenging problem of growing importance [14].

Only few publications partially addressed this area. In [15] the authors presented FAME, a Fault-Pattern Based Memory Failure Analysis framework. The proposed approach applies diagnosis-based fault analysis to narrow down the potential causes of failure for a specific memory architecture. The result of this analysis is then used to optimize a given test sequence (march test) by removing those operations introduced to test faults that have never been observed during the fault analysis. Even if the experimental results show the effectiveness of the approach, it mainly targets the optimization of existing test sequences designed for already known fault models. It does not allow the automatic definition of new fault models and the generation of new sequences customized for the target defects. In [16] the authors proposed a framework for fault analysis and test generation in DRAMs. The proposed approach uses Spice to model both the memory under test and target defects. Spice simulations are used to perform fault analysis starting from well known test algorithms available in the literature. The approach suffers of the same drawbacks of [15], moreover no experimental results are provided in the paper to analyze the effectiveness of the proposed solution.

This paper proposes a preliminary tentative of designing a software framework to automate the process of memory defects simulation and fault models extraction, trying to overcome some of the limitations of previous publications. The target memory is modeled at the electrical level using Spice, and defects are directly inserted in this model. Even if the proposed solution is built over a Spice simulator, it can be easily extended to other types of electrical or layout level models and simulators. In order to guarantee an efficient exploration of a huge space of simulation alternatives, the proposed framework implements an evolutionary approach able to drive the simulation towards those sequences that more probably allow to highlight faulty behaviors.

The effectiveness of the proposed solution has been validated by performing an extensive set of experiments on well known memory defects, using different technologies.

The paper is organized as follows: Section II overviews the characteristics of the different elements composing the framework while Section III validates the effectiveness of the proposed approach by proposing an extensive set of experimental results. Finally, Section IV summarizes the main contributions of the work and concludes the paper identifying future research activities on the same topic.

II. Generation Framework Architecture

Figure 1 introduces the main functional blocks composing the proposed framework.

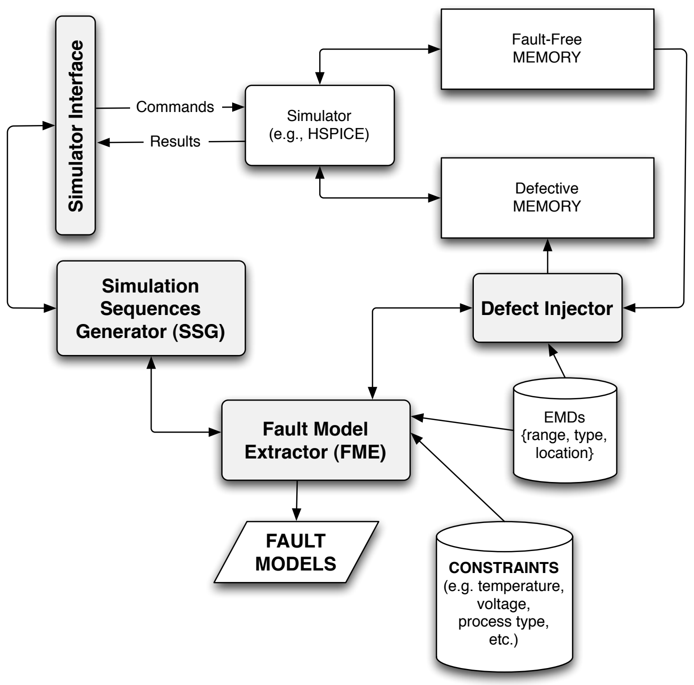
Simulator Interface
Commands
Results
Simulator (e.g., HSPICE)
Fault-Free MEMORY
Defective MEMORY
Simulation Sequences Generator (SSG)
Defect Injector
Fault Model Extractor (FME)
EMDs {range, type, location}
FAULT MODELS
CONSTRAINTS (e.g. temperature, voltage, process type, etc.)

Fig. 1. Fault models extraction framework general architecture

The *Fault Model Extractor* (FME) represents the core of the framework. It drives the modeling process and collects the information required to identify and to define relevant fault models. We use an electrical Spice model to represent the internal memory design and behavior, as well as the characteristics of the fabrication process (*fault-free memory* of Figure 1). This provides a fairly accurate representation of the specific behavior of the memory under analysis. Target memory defects are directly modeled as electrical components (e.g., resistors, capacitors, etc.) on the fault-free memory to create the *defective memory* model. The *Defect Injector* is the block in charge of modifying the fault free memory by inserting the target defects according to the instructions provided by the FME.

The identification of the set of target defects is usually the result of layout read-back and Inductive Fault Analysis out of the scope of this paper [17].

The fault model extraction process analyzes the behavior of the defective memory in presence of a set of m *Electrical Memory Defects* (EMD). All defects defined for a single experiment are concurrently injected in the memory. An EMD is characterized by a set of different parameters, e.g., type, location, range of possible values, etc..

The goal of the FME is to efficiently explore the range of values defined for the different EMDs to identify faulty behaviors arising from the defects injection, and to derive the corresponding fault models. This operation is assisted by the *Simulation Sequences Generator* (SSG).

Given a set of EMDs values, the SSG is in charge of: (i) generating a set of simulation sequences, (ii) applying them to both the fault free and the defective memory, and (iii) analyzing the simulation results to highlight faulty behaviors correlated with the inserted defects. The comparison between the fault-free and the defective memory outputs is performed by analyzing their electrical simulations (*Simulator* block of Figure 1) when the target simulation sequence is applied. To be able to work with different types of memories, description levels, description languages, and simulators, a *Simulator Interface* is placed between the SSG and the *Simulator* to virtualize the specific commands and result formats.

Simulation sequences are generated in the form of march tests fully resorting to all degrees of freedom provided by the march test definition [18]. The SSG returns the observed faulty behaviors using the fault primitive formalism [3]. In order to guarantee an efficient exploration of a huge space of simulation alternatives the SSG implements an evolutionary approach based on a genetic algorithm able to drive the simulation towards those sequences that more probably allow to highlight faulty behaviors, thus reducing the number of required simulations.

The following subsections detail the characteristics of each block composing the framework focusing on the FME and on the SSG that represent the most relevant parts.

A. Fault Model Extractor

The Fault Model Extractor (FME) coordinates the overall defect analysis process. Its main activity is the implementation of an efficient strategy to analyze the memory behavior in all considered defective conditions. This in turn requires exploring the range of values the target EMDs can assume.

Each EMD is characterized by the following parameters:

- *Type*: any type of component that can be modeled into a Spice netlist represents a potential defect type. Examples are: resistors, capacitors, shorts, opens, etc.;
- *Location*: it represents the location of the defect in the memory model. The location identifies the nodes where the component modeling the defect should be placed;
- *Range*: represents the range of values $[EMD_{min}, EMD_{max}]$ the defect may assume during the analysis (e.g., resistor's size for resistive defects).

The defect range is one of the most critical parameters affecting the efficiency of the defect analysis and fault extraction process. An efficient exploration of this space of values is mandatory to keep the simulation time and therefore the overall fault model extraction time under control. In this preliminary work, to allow a first assessment of the proposed approach, we focused on single defects analysis. This limitation concerns the defect range exploration strategy implemented in the FME only. It does not impact all remaining blocks of the framework that are already designed to work with multiple defects. While representing a simplification of the problem, this assumption still allows to propose a framework and a set of results comparable with previous hand performed memory defect analyses such as the one proposed in [19], [20], [10].

By performing a preliminary and extensive campaign of simulations with different defects/technologies, and by looking at the defect analyses available in the literature ([19], [20], [10]) we observed that typically, for a given defect (e.g., resistive defects), the memory behavior can be classified according to the distribution proposed in Fig. 2.

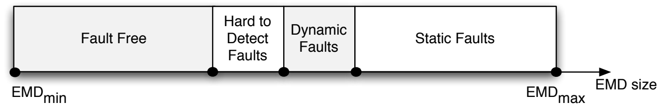

Fig. 2. Faults distribution for a single defect

Fig. 2 splits the defect range in four main defect areas:

- *Fault free:* the defect size is too small to cause any faulty behavior;
- *Hard to detect faults:* represent a very particular defective conditions already analyzed in [13]. The defects do not produce any faulty behavior at the output of the memory. Nevertheless, there is a strong degradation of the memory internal signals (e.g., the differential value of the bit lines almost equal to zero), that may in turn lead to a faulty behavior with a small variation of the operational conditions (e.g., voltage, temperature, etc.). The ability of identifying these situations could be very useful to better characterize the reliability of the target memory;
- *Dynamic faults:* with the increase of the defect size, faulty behaviors start to appear. In many situations, with a small defect size, the fault assumes a dynamic behavior, i.e., it requires more than one operation to be sensitized;
- *Static faults:* when the defect size reaches a certain level, the memory starts to show a static faulty behavior, i.e., a single memory operation is enough to sensitize the fault.

Depending on the memory architecture, technology, and defect type, some of the four areas may be missing. In particular, dynamic faults and hard to detect faults are usually observed in very specific conditions for very specific types of defects. Moreover, depending on the defect type and location, the order of the four defect areas may be specular.

Considering this fault distribution, the FME has to efficiently identify the existing defect areas, and for each of them provide the identified fault models. Being a complete exploration of the defect range impossible under all possible sensitizing sequences the FME explores the defect range using the following heuristic:

- Each area is analyzed separately starting from the static fault area;
- In order to efficiently identify the border between the target area and the remaining ones a binary search is applied. The two limits of the defect range are first analyzed and characterized followed by the analysis of the middle point of the range. The analysis of the two limits is required to identify whether the target area actually exists, and also to identify the order of the areas. For example, considering the static faults area, at the first step the two limits EMD_{min} and EMD_{max} of the defect range are analyzed to check whether they actually fall in the no fault area and in the static faults area respectively;
- Based on the result of the middle point analysis the search space is restricted. Considering the previous example, if the middle point does not fall in the static fault area, the search space is restricted to the upper half of the defect range;
- The process is repeated until the border of target area is identified;
- Each intermediate analysis is also used to restrict the search space of the remaining defect areas.

This approach allows to reduce the number of required simulations while still allowing to explore the full defect range. For each analyzed defect point, the FME relies on the Defect Injector to actually place the target defect in the defective memory and to the SSG to analyze the behavior of the memory in the defective condition. The output of the FME is the characterization of the target memory in terms of identified fault models in the four defect areas.

B. Defect Injector

The defect injector is a very simple block in charge of injecting defects into the memory model in order to build the target defective memory. The implementation of this block is strictly dependent on the type of memory model and simulator used to implement the framework. In the proposed Spice based implementation this block, based on the information about defect types, locations and values, modifies the Spice model of the fault free model by adding the components modeling the target defects. This is an easy task mainly involving the elaboration of text files. For this reason no additional details will be provided in the paper.

C. Simulation Sequences Generator

The Simulation Sequences Generator (SSG) is in charge of analyzing a defective memory by performing a set of electrical simulations in order to identify the occurrence of faulty behaviors.

The complexity of this activity stems in the identification of appropriate sequences of memory operations able to sensitize and to observe faulty behaviors caused by the insertion of memory defects. Besides considering a predefined set of fault primitives and related test sequences such as the ones available in the literature [3], we would like to be able to analyze a wider space of test sequences comprising those not yet identified as candidates for sensitizing faulty behaviors. This is particularly important to possibly identify new fault models arising in emerging technologies and memory architectures.

It is clear that, working with this assumption, the space of candidate test sequences is too large to be explored exhaustively, especially when working in the dynamic faults area (see Section II-A) where more than one operation is required to sensitize the fault.

To efficiently explore this search space the SSG resorts to an evolutionary approach based on a genetic algorithm that tries to drive the simulation toward those test sequences (*solutions*) with higher probability of identifying faulty behaviors. Test sequences are generated in the form of march tests fully resorting to all degrees of freedom provided by the march test definition [18]. Each element composing the march test (i.e., memory operations, and addressing orders) represents a *gene*. The genetic algorithm represents each solution of the problem as a string (*chromosome*) of *genes*. The problem is thus reduced to the identification of a chromosome (sequence of memory operations) able to sensitize and to observe a faulty behavior into the target defective memory.

A fitness value is assigned to each chromosome, based on the value given by an evaluation function (*fitness function*). The fitness function should somehow measure how close the individual is to the optimum solution. A set of individuals constitutes a *population* that evolves from one generation to the next through a *reproduction* process that acts mixing the genetic material that comes from chosen individuals (*parents*), forming new individuals called *children* and deleting old ones. The process starts with an initial population created in some way (e.g., randomly).

The evolution is usually based on two different mechanisms:

- *Cross-over*: chromosomes of two individuals are combined to obtain new individuals inserted in the population to eventually replace existing ones, e.g., the ones with the lowest fitness (elitism);
- *Mutation*: one or more genes of a selected individual are randomly changed. This provides additional chances of entering unexplored regions.

The fitness function is the key element used to drive the evolution process and in particular to select those chromosomes that most likely lead to valid solutions of the problem. In our specific problem, the idea is to identify a function that privileges the ability of a chromosome of sensitizing faulty behaviors, i.e., the ability of producing different electrical signals at the nodes of the fault-free and of the defective memory.

The proposed fitness function is based on the concept of *probe nodes*, i.e., internal nodes of a memory cell, and output nodes of a sense amplifier. The electrical signals (i.e., voltage, current) produced at each probe node of the target memory during the electrical simulation of the test sequence associated with a chromosome are traced. These values are then analyzed and combined to compute a fitness value.

The fitness, assigned with a chromosome x at the simulation time t is expressed by Eq. 1 where N_{probe} represents the number of analyzed probe nodes:

$$f_t(x) = \sum_{i=0}^{N_{probe}-1} D_{i,t} \tag{1}$$

It is computed as the sum of the differential values $D_{i,t}$ (Eq. 2) between the signal at probe node i in the fault-free memory ($Ng_{i,t}$) and the signal at the probe node i in the defective memory ($Nf_{i,t}$).

$$D_{i,t} = |\, \mathrm{Ng}_{i,t} - \mathrm{Nf}_{i,t} \,| \tag{2}$$

The values $f_t(x)$ obtained at each simulation time are then combined to obtain the final fitness according to Eq. 3:

$$f(x) = \sum_{t=0}^{T_{max}} f_t(x) \tag{3}$$

where T_{max} is the total simulation time.

The proposed function has two main drawbacks:

- It can easily lead to populations with very small differences between the individuals. This actually turns the evolution process into a random selection among the chromosomes reducing the efficiency of the genetic approach;
- It can produce among a high number of similar individuals, a single chromosome (*super chromosome*) with fitness much higher than all remaining ones. This is again negative since the evolution will be completely polarized by this chromosome and the space of solutions will not be correctly explored.

To avoid these problems, we introduce a *linear normalization* able to correctly distribute the fitness values. The population is sorted by decreasing fitness. Chromosomes in the sorted list receive a new scaled fitness $f_s(x)$ according to Eq. 4.

$$f_s(x) = C - n \cdot L \tag{4}$$

where C is a constant value, L represents the linear normalization step (a parameter of the method), and n is the position of the chromosome in the sorted list.

The generation process works by mutating the population of chromosomes, and trying to maximize the fitness of the solution. The mutation consists in modifying the genes composing each chromosome, as well as adding additional genes. This in turn means modifying the march elements composing the march tests represented by the chromosomes. The search process ends when either a chromosome able to sensitize and detect a faulty behavior appears in the population, or a maximum computation effort is reached. Several optimizations are applied during the mutation process to guarantee that each chromosome always represents a valid test sequence.

Every time a new chromosome is generated, the electrical simulations of the fault-free and of the target defective memory are compared in order to identify if the given test sequence is able to identify a new erroneous behavior. In particular, the analysis focuses on the portions of the simulation (samples) corresponding to genes encoding read operations. The logic value returned by the observations is calculated (taking into account the electrical parameters of the target memory) for the fault-free and the defective memory. When the two values differ, a faulty behavior is detected and the generation ends. Starting from a solution, and by also analyzing the internal state of the memory cells during the application of the test sequence, the fault primitive corresponding to the faulty behavior is finally generated.

III. Experimental results

To demonstrate the effectiveness of the proposed framework, we performed a set of experiments on a simple SRAM architecture, considering different technologies and using a well established set of defect locations.

A. Memory model and defect locations

The experiments have been performed considering a Spice model of a Static Random Access Memory (SRAM) block. Our reference architecture includes a cell array organized as a 512x512 matrix including, the pre-charge circuits for bit and I/O data lines, the sense amplifier, the write circuitry, and the address decoder. Due to the amount of required simulations, to keep the simulation time into a reasonable level, the generated test sequences have been applied to a reduced portion of the cell array only. This simplification still allows us to obtain realistic results since it has been demonstrated in [21] that defects are usually localized in a range of a few cells.

We considered two implementations of the memory core, using two different technologies with feature size of 130 nm and 65 nm respectively. Both technologies are from Predictive Technology Models [22].

Each memory cell consists of a standard 6-Transistors architecture. Each bit line has a capacitance used to model the parasitic capacitance associated with line metallization. Finally the read path and the write path are enabled by independent control signals.

Figure 3 proposes the architecture of a single memory cell including our target collection of defect locations. DFR1-DFR6 identify six typical resistive defects deeply analyzed in literature [20], [13], whereas DFS1-DFS5 represent the set of short defects analyzed in [23].

We considered a defect range between $1K\Omega$ and $500M\Omega$ for resistive defects, while shorts have been modeled by a resistor of 1.0Ω.

B. Experimental conditions

To correctly validate the results obtained by the experiments, we considered similar working conditions for both technologies. The two technologies differ for the timing required for the correct execution of each operation, and for the supply voltage required by the circuitry, only. Considering the 130nm technology, the operation cycle is 10 ns. The first 3.3 ns are used by the internal precharge circuits to pre-charge the core cell bit lines and the I/O data bit lines to Vdd. The next 3.3 ns are dedicated to drive the write signals in the case of write operations, while the last 3.3 ns can be used to enable the read

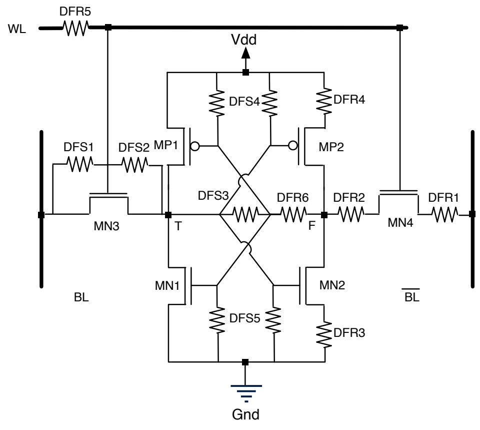

Fig. 3. SRAM cell with defects

path of the memory. The 65 nm technology allows a faster operation of 1.5 ns. The cycle is divided into three regions of 0.5 ns each corresponding to the three regions already defined for the 130 nm technology.

The typical voltage supply is equal to 1.5 V for the 130 nm memory, and 1.1 V for the 65 nm memory. For both technologies we considered the typical process, and three possible operational temperatures of $-40°$C, $27°$C, and $125°$C to cover both normal working conditions and more extreme once. Finally, we also considered the effect of the voltage degradation by analyzing the 65 nm memory using a supply voltage of 0.8 V .

C. Results

We implemented the proposed framework using the commercial HSPICE^{TM}simulator and about 3,500 lines of C code. All experiments have been performed on a dual socket, 3.4 GHz Intel Pentium 4^{TM}desktop equipped with 2 Gb of RAM and running the Linux Operating System.

Table I summarizes the results obtained by considering resistive and short defects with the 130 nm technology based memory. The table is divided in two parts, the left side refers to the analysis of resistive defects while the right side refers to the analysis of short defects (see Fig. 3).

Considering resistive defects, the first column identifies the target defect, the second one expresses the minimum resistance able to create a faulty behavior, the third and fourth columns define the operational conditions in terms of temperature and supply voltage applied to the memory circuit, and the last column gives the fault primitive extracted by the framework. In a similar way, for short defects Table I provides the same information except for the defect value that in this case is fixed to 1.0Ω.

Table I shows that, for this particular technology, all defects lead to static faulty behaviors where R_{min} represents the border between the no fault and the static faults area (see Fig. 2). No dynamic or hard to detect faults have been identified. Moreover, DFR4 leads to a faulty behavior only for very extreme working conditions, i.e. $125°C$. These results are in accordance with similar analyses proposed in [20], [13] for resistive defects and [23] for shorts. The only difference we highlighted concerns DFR4 where we were able to observe a data retention fault not identified in previous studies. Nevertheless, this defect was already considered critical in the literature, and the different technology and memory architecture used in our experiments may lead to this difference.

Table II shows the main results obtained analyzing the $65\,\mathrm{nm}$ memory block in presence of resistive defects. Being this technology more critical, we performed a deeper analysis considering different working conditions. For each working condition the table reports the characterization of the memory behavior showing how, changing the defect value, the memory behavior changes according to the four defect areas proposed in Fig. 2.

As expected DFR2, DFR3 and DFR4 lead to faulty behaviors that are more difficult to detect, in particular when considering defects in the dynamic range ([20], [13]). DFR2 and DFR3 show both static and dynamic faulty behaviors while DFR1, DFR5 and DFR6 only lead to static faults. The fault primitives extracted for DFR3 and DFR4 in the dynamic defect area have a correspondence with the faulty dynamic behaviors presented in the literature for the same defects and similar technologies [20], [13]. It is interesting to note how changing the working conditions the number of operations required to sensitize dynamic faults changes.

The experiments also lead to the identification of an hard to detect fault for DFR4 operating in normal conditions. When the defect value is higher than $300M\Omega$ the internal core cell array signals have a very high degradation, even if the entity of the degradation is not enough to lead the cell to flip. This phenomena is depicted in Fig. 4 where the voltage of the internal node F runs very close to the threshold switching voltage without crossing it, regardless of the number of applied read operations. Hard to detect faults requires a few considerations. We defined hard to detect faults those conditions in which there is a strong degradation of the internal memory signals. There are different conditions that may fall in this class. In the current implementation hard to detect faults are identified looking at the internal nodes of the cell (i.e., nodes F and T of Fig. 3). When one of these nodes is close to the threshold switching voltage less than a certain value δ, configurable by the user, an hard to detect fault is extracted.

Finally, Table III shows the main results concerning the injection of short defects in the 65nm technology based memory. As in the case of 130nm based memory, all defects led to a static faulty behavior, regardless the test conditions applied to the circuit.

The generation time to perform a complete characterization, e.g., the set of resistive defects for a given working condition, was of about 6 hours of computation time. This includes the complete exploration of the defect range of all defects.

To conclude, in order to show the effectiveness of the proposed genetic algorithm in efficiently choosing the test sequences to simulate, and in particular to assess the effectiveness of the proposed fitness function, Fig. 5 depicts the fitness value for the candidate solution during the analysis of DFR4 with a temperature of $125°C$ in the dynamic defects area. The x axis reports the number of generations while the y axis shows the fitness. The figure clearly shows how the fitness correctly increases toward the generation process reaching the maximum value at the end of the generation when the solution is identified. The different steps are connected to mutations inserting new operations in the test sequences, while the flat areas identify the exploration with a fixed test sequence length. Being the target fault dynamic, adding new operations allows to better approach the solution.

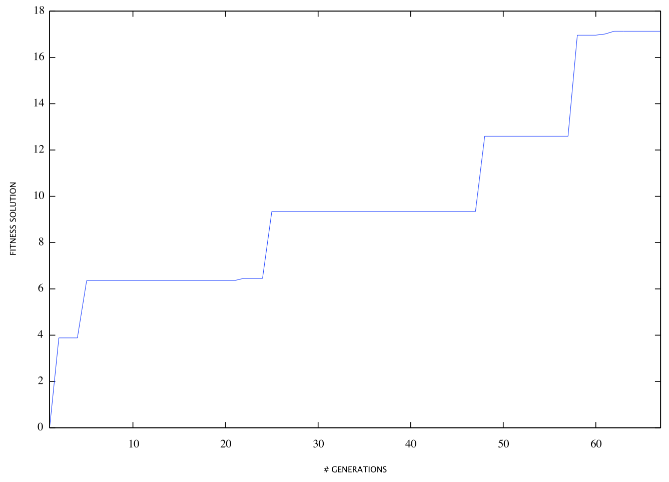

Fig. 5. Fitness trend for DFR4 with a temperature of $125°C$ in the dynamic defects area

TABLE III
$65\,\mathrm{nm}$ TECHNOLOGY ANALYSIS FOR SHORT DEFECTS

Short defects			
#DFS	**Temp.**	**Volt. supply**	**Fault primitive**
$DFS1$	27°C	1.1V	$\langle 1W0/1/-\rangle$
$DFS2$	27°C	1.1V	$\langle 1W0/1/-\rangle$
$DFS3$	27°C	1.1V	$\langle 1R1/0/0\rangle$
$DFS4$	27°C	1.1V	$\langle 1R1/0/0\rangle$
$DFS5$	27°C	1.1V	$\langle 1R1/0/0\rangle$

IV. Conclusions

This paper proposed a set of preliminary results toward the definition of an efficient framework for automatic memory defects simulation and fault model extraction. The target memory is modeled at the electrical level using Spice, and defects are directly inserted in this model. A set of heuristics, and the use of evolutionary techniques allow to keep the time required to perform the overall analysis under control.

The effectiveness of the proposed solution has been validated by performing an extensive set of experiments on well known memory defects, using different technologies. The results showed that we have been able to automatically extract, with a reasonable execution time, most of the results manually

TABLE I
130 nm TECHNOLOGY ANALYSIS

Resistive defects					short defects			
#DFR	**Rmin**	**Temp.**	**Volt. supply**	**Fault primitive**	**#DFS**	**Temp.**	**Volt. supply**	**Fault primitive**
$DFR1$	25KΩ	27°C	1.5V	$\langle 0W1/0/-\rangle$	$DFS1$	27°C	1.5V	$\langle 0W1/0/-\rangle$
$DFR2$	20KΩ	27°C	1.5V	$\langle 1W0/1/-\rangle$	$DFS2$	27°C	1.5V	$\langle 0W1/0/-\rangle$
$DFR3$	7KΩ	27°C	1.5V	$\langle 1R1/0/0\rangle$	$DFS3$	27°C	1.5V	$\langle 0W1/0/-\rangle$
$DFR4$	64MΩ	125°C	1.5V	$\langle 0_T/\uparrow/?\rangle$	$DFS4$	27°C	1.5V	$\langle 0W1/0/-\rangle$
$DFR5$	2MΩ	27°C	1.5V	$\langle 0W1/0/-\rangle$	$DFS5$	27°C	1.5V	$\langle 0W1/0/-\rangle$
$DFR6$	2MΩ	27°C	1.5V	$\langle 1W0/1/-\rangle$				

TABLE II
65 nm TECHNOLOGY ANALYSIS FOR RESISTIVE DEFECTS

Condition: Temp = 27°C / V_{dd} = 1.1V

#DFR	**No Fault**	**Dynamic**	**Fault Primitive**	**Static**	**Fault Primitive**
$DFR1$	1.00KΩ - 184.99KΩ	-	-	185.00KΩ - 500.00MΩ	$\langle 0W1/0/-\rangle$
$DFR2$	1.00KΩ - 28.76KΩ	28.77KΩ - 28.79KΩ	$\langle 0R0R0/1/1\rangle$	28.80KΩ - 500.00MΩ	$\langle 0R0/1/1\rangle$
$DFR3$	1.00KΩ - 14.37KΩ	14.38KΩ	$\langle 1R1R1/0/0\rangle$	14.39KΩ - 500.00MΩ	$\langle 1R1/0/0\rangle$
$DFR4$	1.00KΩ - 500.00MΩ	-	-	-	-
$DFR5$	1.00KΩ - 589.99KΩ	-	-	590.00KΩ - 500.00MΩ	$\langle 0W1/0/-\rangle$
$DFR6$	1.00KΩ - 549.99KΩ	-	-	550.00KΩ- 500.00MΩ	$\langle 0R0/1/1\rangle$

Condition: Temp = -40°C / V_{dd} = 1.1V

#DFR	**No Fault**	**Dynamic**	**Fault Primitive**	**Static**	**Fault Primitive**
$DFR1$	1.00KΩ - 294.99KΩ	-	-	295.00KΩ - 500.00MΩ	$\langle 0W1/0/-\rangle$
$DFR2$	1.00KΩ - 2.16KΩ	-	-	2.17KΩ - 500.00MΩ	$\langle 0R0/1/-\rangle$
$DFR3$	1.00KΩ - 1.33KΩ	1.34KΩ - 1.45KΩ	$\langle 1R1R1/0/0\rangle$	1.46KΩ - 500.00MΩ	$\langle 1R1/0/0\rangle$
$DFR4$	-	-	-	1.00KΩ - 500.00MΩ	$\langle 1W0/1/-\rangle$
$DFR5$	1.00KΩ - 500.00MΩ	-	-	-	-
$DFR6$	1.00KΩ - 524.99KΩ	-	-	525.00KΩ - 500.00MΩ	$\langle 0R0/1/1\rangle$

Condition: Temp = 125°C / V_{dd} = 1.1V

#DFR	**No Fault**	**Dynamic**	**Fault Primitive**	**Static**	**Fault Primitive**
$DFR1$	1.00KΩ - 154.99KΩ	-	-	155.00KΩ - 500.00MΩ	$\langle 0W1/0/-\rangle$
$DFR2$	1.00KΩ - 30.20KΩ	30.21KΩ - 30.24KΩ	$\langle 0R0R0/1/1\rangle$	30.25KΩ - 500.00MΩ	$\langle 0R0/1/1\rangle$
$DFR3$	1.00KΩ - 14.21KΩ	14.22KΩ - 14.23KΩ	$\langle 1R1R1/0/0\rangle$	14.24KΩ - 500.00MΩ	$\langle 1R1/0/0\rangle$
$DFR4$	1.00KΩ - 9.99MΩ	10.00MΩ - 500.00MΩ	$\langle 1W0R0R0R0R0/1/1\rangle$	-	-
$DFR5$	1.00KΩ - 974.99KΩ	-	-	975.00KΩ - 500.00MΩ	$\langle 0W1/0/-\rangle$
$DFR6$	1.00KΩ - 349.99KΩ	-	-	350.00KΩ - 500.00MΩ	$\langle 0R0/1/1\rangle$

Condition: Temp = 27°C / V_{dd} = 0.8V

#DFR	**No Fault**	**Dynamic**	**Fault Primitive**	**Static**	**Fault Primitive**
$DFR1$	1.00KΩ - 154.99KΩ	-	-	155.00KΩ - 500.00MΩ	$\langle 0W1/0/-\rangle$
$DFR2$	1.00KΩ - 154.50KΩ	154.51KΩ - 228.90KΩ	$\langle 0R0R0/1/1\rangle$	228.91KΩ - 500.00MΩ	$\langle 0R0/1/1\rangle$
$DFR3$	1.00KΩ - 79.71KΩ	79.72KΩ	$\langle 1R1R1R1R1/0/0\rangle$	79.73KΩ - 500.00MΩ	$\langle 1R1/0/0\rangle$
$DFR4$	1.00KΩ - 2.46MΩ	-	-	2.47MΩ - 500.00MΩ	$\langle 1W0/1/-\rangle$
$DFR5$	1.00KΩ - 499.99KΩ	-	-	500.00KΩ - 500.00MΩ	$\langle 0W1/0/-\rangle$
$DFR6$	1.00KΩ - 374.99KΩ	-	-	375.00KΩ - 500.00MΩ	$\langle 0R0/1/1\rangle$

obtained and published in previous works. The proposed experiments mainly focused on memory cells defects, the application to defects in the surrounding circuitry such as slow write driver faults proposed in [24] is still under investigation. While the proposed framework does not directly addresses test generation, the identified list of fault models can be directly used as starting fault list for march test generation algorithms such as the ones proposed in [4], [5], [6], [7], [8], [9] in order to generate test sequences customized for realistic defects in the target memory/technology, and thus optimizing

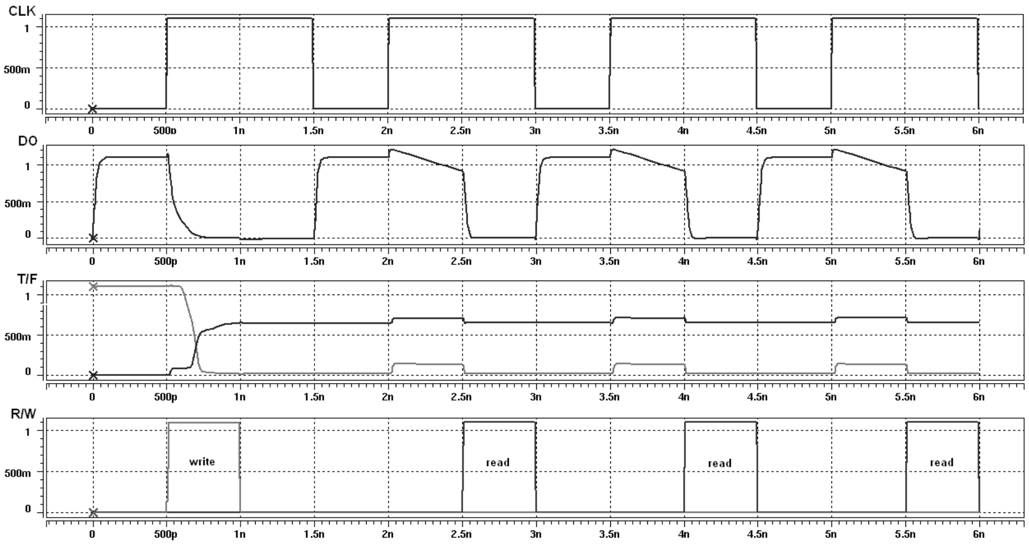

Fig. 4. DFR4 hard to detect faulty behavior

the resulting test sequences. On going activities to enhance the effectiveness of the approach concerns the extension to the analysis of multiple defects, and the improvement of the genetic approach in order to better explore the space of possible test sequences, and to additionally reduce the computation time.

REFERENCES

[1] International technology roadmap for semiconductors. [Online]. Available: http://www.itrs.net/

[2] A. J. van de Goor, "Using march tests to test srams," *IEEE Des. Test. Comput.*, vol. 10, no. 1, pp. 8–14, Mar. 2004.

[3] A. J. van de Goor and Z. Al-Ars, "Functional memory faults: A formal notation and a taxonomy," in *Proc. 18th IEEE VLSI Test Symposium (VTS'00)*, Montreal, Canada, Apr.30–May3, 2000, pp. 281–289.

[4] B. Smit and A. J. van de Goor, "The automatic generation of march tests," in *Proc. IEEE International Workshop on Memory Technology, Design and Testing (MTDT'94)*, San Jose (CA), USA, Aug.8–9, 1994, pp. 86–91.

[5] K. Zarrineh, S. Upadhyaya, and S. Chakravarty, "Automatic generation and compaction of march tests for memory arrays," *IEEE Trans. VLSI Syst.*, vol. 9, no. 6, pp. 845–857, Dec. 2001.

[6] K.-L. Cheng, C.-W. Wang, J.-N. Lee, Y.-F. Chou, C.-T. Huang, and C.-W. Wu, "Fault simulation and test algorithm generation for random access memories," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 21, no. 4, pp. 480–490, Apr. 2002.

[7] S. M. Al-Harbi and S. Gupta, "Generating complete and optimal march tests for linked faults in memories," in *Proc. 21st IEEE VLSI Test Symposium (VTS'03).* Napa Valley (CA), USA: IEEE Computer Society, Apr.27–May1, 2003, pp. 254–261.

[8] D. Niggemeyer and E. M. Rudnick, "Automatic generation of diagnostic memory tests based on fault decomposition and output tracing," *IEEE Trans. Comput.*, vol. 53, no. 9, pp. 1134–1146, Sep. 2004.

[9] A. Benso, A. Bosio, S. Di Carlo, G. Di Natale, and P. Prinetto, "Automatic march tests generation for static and dynamic faults in srams," in *Proc. 10th IEEE European Test Symposium (ETS'05)*, Tallinn, Estonia, May 22–25, 2005, pp. 122–127.

[10] L. Dilillo, P. Girard, S. Pravossoudovitch, A. Virazel, and S. Borri, "Comparison of open and resistive-open defect test conditions in sram address decoders," in *12th IEEE Asian Test Symposium (ATS 2003)*, I. Press, Ed., November 16–9 2003, pp. 250–255.

[11] L. Dilillo, P. Girard, S. Pravossoudovitch, A. Virazel, and M. Bastian Hage-Hassan, "Resistive-open defect influence in sram pre-charge circuits: analysis and characterization," in *IEEE European Test Symposium (ETS 2005)*, I. Press, Ed., May, 22-25 2005, pp. 116–121.

[12] S. Hamdioui, Z. Al-Ars, and A. van de Goor, "Opens and delay faults in cmos ram address decoders," *IEEE Trans. Comput.*, vol. 55, no. 12, pp. 1630–1639, Dec 2006.

[13] L. Dilillo, P. Girard, S. Pravossoudovitch, A. Virazel, and M. Hage-Hassan, "Data retention fault in sram memories: analysis and detection procedures," in *23rd IEEE VLSI Test Symposium (VTS 2005)*, May, 1–5 2005, pp. 183–188.

[14] J. Rajski and K. Thapar, "Nanometer design: what are the requirements for manufacturing test?" in *Proceedings Design, Automation and Test in Europe Conference and Exhibition, DATE 2004*, vol. 2, Feb. 2004, pp. 930–935.

[15] K.-L. Cheng, W. Chih-Wea, L. Jih-Nung, C. Yung-Fa, H. Chih-Tsun, and W. Cheng-Wen, "Fame: a fault-pattern based memory failure analysis framework," in *International Conference on Computer Aided Design, 2003. ICCAD-2003.*, 9-13 Nov. 2003, pp. 595–598.

[16] Z. Al-Ars, S. Hamdioui, G. Mueller, and A. van de Goor, "Framework for fault analysis and test generation in drams," in *Proceedings Design, Automation and Test in Europe, 2005.*, 2005, pp. 1020–1021.

[17] A. Jee and F. Ferguson, "Carafe: an inductive fault analysis tool for cmos vlsi circuits," in *11th IEEE VLSI Test Symposium*, Apr 1993, pp. 92–98.

[18] N. Niggemeyer, M. Redeker, and J. Ottersted, "Integration of non-classical faults in standard march tests," in *IEEE International Workshop on Memory Technology, Design and Testing, MTDT'98*, Aug 24–26 1998, pp. 91–96.

[19] L. Dilillo, P. Girard, S. Pravossoudovitch, A. Virazel, and M. Bastian, "Resistive-open defect injiection in sram core-cell: analysis and comparison between 0.13μm and 90nm technologies," in *Proceedings 42nd Design Automation Conference, DAC'05*, IEEE/ACM, Ed., 2005.

[20] L. Dilillo, P. Girard, S. Pravossoudovitch, A. Virazel, S. Borri, and M. Hage-Hassan, "Resistive-open defects in embedded-sram core cells: analysis and march test solution," in *IEEE 13th Asian Test Symposium (ATS 2004)*, November 15–17 2004, pp. 266–271.

[21] Z. Al-Ars and A. van de Goor, "Static and dynamic behavior of memory cell array spot defects in embedded drams," *IEEE Trans. on Comp.*, vol. 52, no. 3, pp. 293–309, 2003.

[22] P. T. M. (PTM). http://www.eas.asu.edu/ ptm.

[23] R.-F. Huang, Y.-F. Chou, and C.-W. We, "Defect oriented fault analysis for sram," in *Proceedings of the 12th Asian Test Symposium*, 2003, pp. 1–6.

[24] A. Ney, P. Girard, C. Landrault, S. Pravossoudovitch, A. Virazel, and M. Bastian, "Slow write driver faults in 65nm sram technology: Analysis and march test solution," in *Design, Automation and Test in Europe Conference and Exhibition, 2007. DATE07*, April 2007, pp. 1–6.