Genetic Defect Based March Test Generation for SRAM

Genetic Defect Based March Test Generation for SRAM

Authors: Di Carlo S., Politano G., Prinetto P., Savino A., Scionti A.,

Author's version of the manuscript published in the LECTURE NOTES IN COMPUTER SCIENCE, Vol. 6625/2011 pp. 141-150.

The final publication is available at www.springerlink.com:

URL: http://www.springerlink.com/content/6l37213060502224/fulltext.pdf

DOI: 10.1007/978-3-642-20520-0_15

Genetic Defect Based March Test Generation for SRAM

Stefano Di Carlo, Gianfranco Politano, Paolo Prinetto, Alessandro Savino, and Alberto Scionti

Politecnico di Torino, Control and Computer Engineering Department, Torino I-10129, Italy,
stefano.dicarlo,gianfranco.politano,paolo.prinetto,alessandro.savino,
alberto.scionti@polito.it
http://www.testgroup.polito.it

Abstract. The continuos shrinking of semiconductor's nodes makes semiconductor memories increasingly prone to electrical defects tightly related to the internal structure of the memory. Exploring the effect of fabrication defects in future technologies, and identifying new classes of 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 exploiting a dedicated genetic algorithm.

Keywords: Memory testing, march test generation, defect based testing

1 Introduction

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

Functional Fault Models (FFMs) coupled with efficient test algorithms such as march tests (the reader may refer to [1] to understand the concept of march test) have been so far enough to deal with emerging classes of memory defects [1]. FFMs do not depend on the specific memory technology and allow automation of test sequences generation [3]. Exploring the effect of fabrication defects in future technologies, and identifying new classes of FFMs with their relevant test sequences, is a time consuming task up to now mainly performed by hand. However, the continuos shrinking of semiconductor's nodes makes semiconductor memories increasingly prone to electrical defects tightly related to the internal structure of the memory [8, 7]. Automating the analysis of these defects is mandatory to guarantee the quality of next generation memory devices.

[1] http://www.itrs.net/

Automatic defect level memory test generation is an area of test generation and diagnosis that still needs to be fully explored. Cheng et al. [4] presented FAME, a fault-pattern based memory failure analysis framework that applies diagnosis-based fault analysis to narrow down potential causes of failure for a specific memory architecture. Results of the analysis are then used to optimize a given test sequence (march test) considering only the observed faults. Similarly, Al-Ars at al.[2] proposed a framework for fault analysis and test generation in DRAMs that uses Spice to model the memory under test and the target defects. Spice simulations are used to perform fault analysis starting from well known test algorithms available in literature. The main drawback of these methods is that they allow the optimization of existing test sequences rather than the generation of new and optimized set of stimuli.

This paper tries to overcome these problems proposing a software framework for defect level march test generation . The generation process exploits a genetic algorithm able to highlight faulty behaviors in a defective memory and to generate the related test sequences by means of electrical simulations. The use of a genetic algorithm allows an efficient exploration of a huge space of march test alternatives guaranteeing high defect coverage.

2 Generation Framework Architecture

Figure 1 overviews the architecture of the proposed framework. The core block of the system is the Genetic March Test Generator (GMTG), a genetic algorithm used to drive the march test generation process.

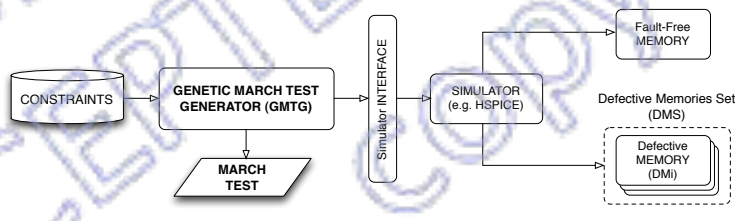

Fig. 1. March test generation framework

We use electrical Spice models to precisely model the memory behavior and the characteristics of the fabrication process (fault-free memory of Figure 1). In a similar way, memory defects are modeled as electrical components (e.g., resistors, parameter changes, etc.) on the fault-free memory obtaining a collection of defective memories (DM_i) named Defective Memory Set (DMS). Each defective memory is characterized by a single defect.

The GMTG operates trying to highlight erroneous behaviors caused by the inserted defects, and providing a march test for their detection. The comparison

between the fault-free and the defective memory models is performed by analyzing their electrical simulations (simulator block of Figure 1) when the target test sequence is applied. To allow adaptation to different types of memories, description levels, description languages, and simulators, an interface layer is placed between the GMTG and the simulator to virtualize the specific commands and results format.

3 Genetic March Test Generator

The following pseudocode describes the way the GMTG algorithm works while generating a march test for a specific set of defects.

```
GMTG (): begin
 1:     solution = "";
 2:     foreach (DM_i in DMS) {
 3:       generation=0; generate initial population based on current solution;
 4:       simulate (DM_i , population);
 5:       while (generation < MAX_GEN) {
 6:         coverage = check_coverage(solution,DM_i);
 7:         if (coverage) break;
 8:         evolve (population, offspringsize)
 9:         validate (population);
10:         simulate (DM_i , population);
11:         evaluate_fitness(population);
12:         solution = update (population,solution);
13:         generation ++;}
14:       if (generation==MAX_GEN) exit_without_solution();}
15:     show_solution (solution);
end
```

The algorithm starts with an empty solution (row 1). Each defect DM_i is examined separately with an iterative process (row 2). This slightly modifies the algorithm structure w.r.t. traditional genetic algorithms. For each defect a random population of individuals is generated. Individuals of the population (chromosomes) represents candidate march tests, and the individual with higher fitness represents the candidate solution. When generating the population for a new defect, all new individuals will contain the genes of the current solution plus new additional genes to guarantee the coverage of the already analyzed defects. Rows 5 to 13 are the actual genetic process with each iteration representing the evolution of the population from a generation to the next one. First the coverage of the current solution w.r.t. the target defect is evaluated (row 6). If the current solution already detects a faulty behavior in the defective model the generation stops and moves to the next model (row 7). If not, the population is evolved applying different genetic operators explained later in this section (row 8). `offspring_size` represents the number of individuals to substitute in the current population. The new population is validated to guarantee that each chromosome correctly represents a march test (row 9). Chromosomes that do

not pass the validation can be either discarded or genetically modified to fit the constraints. The population is then simulated w.r.t. the target defect (row 10), the fitness of each individual is computed (row 11), the new candidate solution is selected (row 12) and the process continues. If the evolution reaches a maximum number of iterations without identifying a suitable solution the generation fails and the algorithm ends (row 14).

3.1 Chromosome Encoding

The proposed genetic algorithm works with chromosomes representing candidate march tests. According to [10] there are six Degrees Of Freedom (DOF) that can be exploited to increase the detection capabilities of march tests. These DOFs are considered in our chromosome encoding to enhance the efficiency of the GMTG. Each chromosome is composed of a sequence of genes representing basic memory operations used to build a march test. Each gene is encoded using a binary string including the following basic fields:

- *start marker* (1 bit): when asserted, it denotes the beginning of a new march element within the current gene;
- *addressing order* (1 bit): defined in correspondence of the beginning of a march element to identify its addressing order (1: direct addressing order ⇑, 0: inverse addressing order ⇓);
- *stop marker* (1 bit): when asserted denotes that the current gene concludes a march element;
- *operation*: a sequence of bits encoding the memory operation to apply. The list of available memory operations depends on the target memory (basic operations we considered are write, read, and idle). During the generation process, the simulator interface (see Figure 1) translates each operation into the correct sequence of signals for the memory;
- *data*: defined in case of write operations, it represents the value to write into the memory (0 or 1 in case of single bit memories);
- *addressing sequence*: is the sequence of addresses associated with the direct and reverse addressing order. This field exploits the first two degrees of freedom proposed in [10]:(i) the addressing sequence can be freely chosen as long as all addresses occur exactly once and the sequence is reversible (DOF1); (ii) the addressing sequence for initialization can be freely chosen as long as all addresses occur at least once (DOF2). In this work we consider two possible sequences: (i)
  - *column mode* ⟨c⟩:each memory cell in a given row of the memory cell array is scanned before moving to the next row;
  - *row mode* ⟨r⟩: each cell in a given column of the memory cell array is scanned before moving to the next column.

  Additional and more complex addressing sequences can be defined and added to the encoding schema to increase the space of possible solutions and to enhance the detection capabilities of the generated algorithms;

– *data pattern*: this field allows to exploits another degree of freedom defined in [10]: the data within a read/write operation does not need to be the same for all memory addresses as long as the detection probabilities for basic faults are not affected (DOF4). Basically this DOF allows to change the data written into the memory while moving to the different cells of the array. We consider four possible data patterns (solid ⟨sol⟩: all cells are written with the same value, checkboard ⟨ckb⟩: cells are written with alternate values, alternate row ⟨alr⟩: rows of the memory are filled with alternate values, and alternate column ⟨alc⟩: columns of the memory are filled with alternate values).

Each chromosome encodes at least a write operation needed to initialize the memory array with a well known value and possibly sensitize a faulty behavior and a read operation to observe the faulty behavior. The number of sensitizing operations can be then incremented to deal with more complex defects or combination of defects.

3.2 Population Validation

During the evolution from a generation to the next one, the application of the genetic transformations may lead to chromosomes with undesired properties, i.e., chromosomes that do not represent valid march tests. To avoid this situation, when new individuals are generated their structure must be validated. Incorrect individuals are not killed but whenever possible their structure is healed applying a set of transformations. These transformations work on the start and stop markers of g es based on the following rules:

– if a gene has the stop marker asserted, the start marker of the next gene of the chromosome must be asserted;
– if a gene has a start marker asserted, the stop marker of the previous gene of the chromosome must be asserted;
– the start marker of the first gene and the stop marker of the last gene of a chromosome must always be asserted.

We decided to introduce these transformations instead of simply discarding individuals with erroneous genetic content to avoid discarding genetic material that may contain interesting characteristics for the final solution.

3.3 Fitness Function

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. The idea is to identify a function that privileges the ability of an individual of sensitizing faulty behaviors, i.e., the ability of producing different electrical signals at the nodes of the fault-free and defective memories. This imposes to define a fitness function able to evaluate analog differences among signals.

The computation of the fitness is based on the concept of *probe nodes*, i.e., I/O nodes of a memory cell (i.e., bit lines) or output nodes of a sense amplifier. The electrical signals (i.e., voltage) produced at each probe node of the target memory during the electrical simulation of the test sequence associated to a chromosome are traced. These values are then analyzed and combined into a value of fitness according to Eq. 1:

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

where x is the target chromosome, t the simulation time and N_{probe} the number of analyzed probe nodes. $D_{i,t}$ represents the absolute value of the difference between the voltage at probe node i in the fault-free memory and the voltage at the probe node i in the defective memory, at simulation time t. These differential values are combined together considering all probe nodes and simulation times by the two sums of Eq. 1. The proposed function has two main drawbacks:

- it can easily lead to populations with very small differences between the individuals (premature convergence). This actually turns the evolution process into a random selection among 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 the 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 leverage these problems, we introduced a linear normalization able to correctly distribute the fitness values. The population is sorted by decreasing fitness values. Chromosomes in the sorted list receive a new scaled fitness $f_s(x)$ according to Eq. 2.

$$f_s(x) = C - n \cdot L \tag{2}$$

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.

3.4 Evolution

During the generation process, when the current solution does not provide the required defect coverage the current population is evolved substituting a set of individuals with new ones with different characteristics. In our framework, in addition to traditional genetic operators (e.g., crossover [5]) we apply additional rules during the evolution. First of all, if a certain sequence of genes in the chromosome has been used to detect a certain defect, it cannot be modified while analyzing a new defect. This in turns requires to add new genes to the sequence in order to have a certain degree of freedom in the modification

of the individuals. This is performed introducing a new genetic operator named *increase_chromosome_length* able to increase by one the number of genes composing the chromosomes of the population. The problem in this case is the selection of the type of gene to insert and, in particular, the type of memory operation the new gene has to encode. Based on the fact that some fault models (i.e., dynamic faults) are sensitized by long sequences of identical operations, the approach we adopted inserts new genes repeating the last operation in the sensitizing sequence of each chromosome. Moreover, the new gene is always inserted as part of the last march element.

3.5 Coverage Conditions

For each defective memory model DM_i the GMTG ends the generation process when either a chromosome able to sensitize and detect a faulty behavior appears in the population, or a maximum computation effort is reached. Every time a new solution is identified the electrical simulations of the fault-free and target defective memories are compared to identify if the given test sequence is able to detect a new erroneous behavior. In particular, the analysis focuses on the portions of the simulation (samples) corresponding to the 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.

4 Experimental Results

The capability of the proposed framework has been validated on a 3×3 SRAM consisting of an array of 9 classical 6 transistors cells implemented using the 130nm Predictive Technology Model[2]. The use of a reduced 3×3 matrix allows to maintain the simulation time under control. Nevertheless, this simplification still allows to obtain realistic results since it has been demonstrated in [1] that defects are usually localized in a range of a few cells. Electrical simulations have been performed using the commercial HSPICE™ simulator while the GMTG has been implemented in about 3,500 lines of C code executed on a 1.8GHz AMD TURION™ laptop equipped with 1GB of RAM and running the Linux operating system. A preliminary set of experiments allowed us to derive the set of tuning parameters for the GMTG[3].

Figure 2 (a) and Figure 2 (b) propose the architecture of a single memory cell including our target collection of defects. Figure 2 (a) proposes seven typical resistive defect locations deeply analyzed in literature [6, 7], whereas Figure 2 (b) proposes the set of short defects analyzed in [9]. During our experiments these defects have been injected in the first cell of the memory matrix.

[2] http://www.eas.asu.edu/ ptm
[3] MAX_GEN=200, population=10, offspring_size=5, C=250, L=25.

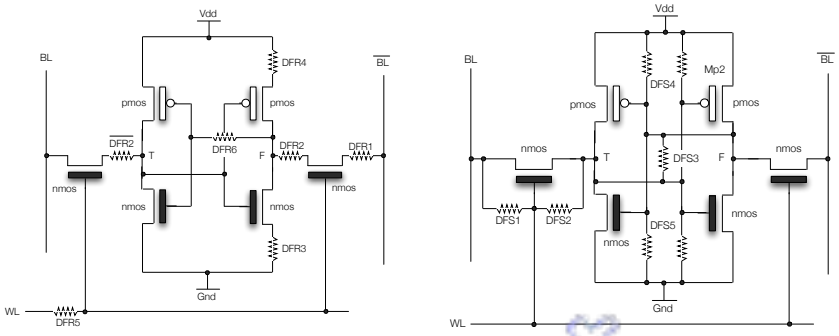

Fig. 2. Memory cell with target resistive defects (a) and short defects (b)

Table 1 shows the results obtained by considering both resistive defects (DFT) and shorts (DFS) in isolation. For resistive defects, the minimum value of resistance able to produce a faulty behavior (Rmin) is reported while for shorts, we considered a resistive value of 1.0Ω. All simulations have been performed at a temperature of 27°C. For each defect the generated test sequence and the generation time are also provided. The obtained results show that we have been able to automatically generate dedicated march tests for resistive defects DFR1, DFR2, DFR3 DFR5, DFR6, $\overline{DFR2}$ and for all short defects with a very low effort in terms of execution time. The generated test sequences are consistent with the studies performed in [6] for resistive defects and in [9] for shorts, thus confirming the effectiveness of the proposed approach.

Table 1. March tests for single defects. Simulations have been performed using $T = 27°C$. Defects are measured in MΩ, while the execution time is expressed in s.

DFR	Rmin	Time	Test Sequence	DFS	Time	Test Sequence
DFR1	0.025	178	$\langle c \rangle \langle ckb \rangle \{\Uparrow (W0, W1, R1);\}$	DFS1	180	$\langle c \rangle \langle ckb \rangle \{\Uparrow (W0, W1, R1);\}$
DFR2	0.020	175	$\langle r \rangle \langle sol \rangle \{\Uparrow (W0, W1, R1);\}$	DFS2	180	$\langle r \rangle \langle sol \rangle \{\Uparrow (W0, W1, R1);\}$
DFR3	0.007	171	$\langle c \rangle \langle alc \rangle \{\Uparrow (W1, R1, R1);\}$	DFS3	180	$\langle c \rangle \langle sol \rangle \{\Uparrow (W0, W1, R1);\}$
DFR4[4]	64.0	416	$\langle c \rangle \langle ckb \rangle \{\Uparrow (W0, R0); \Uparrow (R0);\}$	DFS4	180	$\langle r \rangle \langle sol \rangle \{\Uparrow (W0, W1, R1);\}$
DFR5	2.0	177	$\langle r \rangle \langle alr \rangle \{\Uparrow (W1, W0, R0);\}$	DFS5	180	$\langle r \rangle \langle alr \rangle \{\Uparrow (W0, W1, R1);\}$
DFR6	2.0	168	$\langle c \rangle \langle alc \rangle \{\Uparrow (W1, W0, R0);\}$	-	-	
$\overline{DFR2}$	2.0	150	$\langle c \rangle \langle alc \rangle \{\Uparrow (W1, W0, R0);\}$	-	-	

Slightly more complex is the situation for DFR4. Experiments performed at simulation temperature of 27°C were not able to identify any faulty behavior. This is again coherent with the results of [6]. We thus performed different experiments changing the operational temperature. By setting the temperature to

[4] Defect DFR4 has been simulated using $T = 125°C$.

Table 2. March tests for multiple defects. Execution time is expressed in s.

#Exp	Time	Test sequence
EXP1	510	$\langle r \rangle \langle sol \rangle$ $\{\Uparrow (W0, W1, R1); \Downarrow (W0, R0);\}$
EXP2	2050	$\langle c \rangle \langle sol \rangle$ $\{\Uparrow (W0, W1, R1, W0, R0);\}$
EXP3	2062	$\langle c \rangle \langle ckb \rangle$ $\{\Uparrow (W0, W1, R1); \Downarrow (W0, W0); \Uparrow (R0); \Uparrow (W1); \Uparrow (R1); \Downarrow (R1);\}$

125°C and the defect size to 64.0MΩ, we have been able to produce a defective behavior and to obtain a corresponding test sequence as shown in Figure 3.

The result obtained with DFR4 is particularly interesting. Looking at the results proposed in [6], the authors identified for the same type of defect, injected in their target memory, a dynamic faulty behavior instead of a data retention fault. Once again, this result stresses the importance of resorting to an automatic tool able to automatically generate test sequences customized on the target memory.

In addition to the previous experiments we performed a set of three experiments with groups of defects summarized by the results of Table 2. EXP1 considers a target defect list composed of DFR2 and $\overline{DFR2}$. According to the results of Table 1 these two defects introduce a Transition Fault into the memory. Looking at the generated test sequence in Table 2 we exactly obtained a march test able to detect the Transition Fault. The second experiment (EXP2) considers the same collection of defects of EXP1 plus DFS3. Again the result of the three defects can be modeled as a Transition Fault and the generated march test is able to detect this type of fault. Finally, EXP3 adds DFR3 to the defects considered in EXP2. The generated sequence is able to detect the transition fault introduced by the first three defects and also the read fault introduced by DFR3. However, in this case, it is clear that the generated sequence contains redundant operations. This situation is a direct consequence of the use of a genetic approach to generate the test. Nevertheless, a post elaboration can be applied in order to optimize the generated sequences.

5 Conclusion

This paper proposed a set of preliminary results toward the solution of the problem of defect based automatic march test generation. The proposed approach is based on a genetic algorithm able to identify faulty behaviors in a defective memory and to generate the corresponding test sequences. The use of a genetic approach allows an efficient exploration of a huge space of march test alternatives, guaranteeing high defect coverage and thereby reducing the time test engineers need to explore test alternatives. Experimental results show the effectiveness of the approach that proved to be able to reproduce results of previous studies with an acceptable execution time and without human intervention.

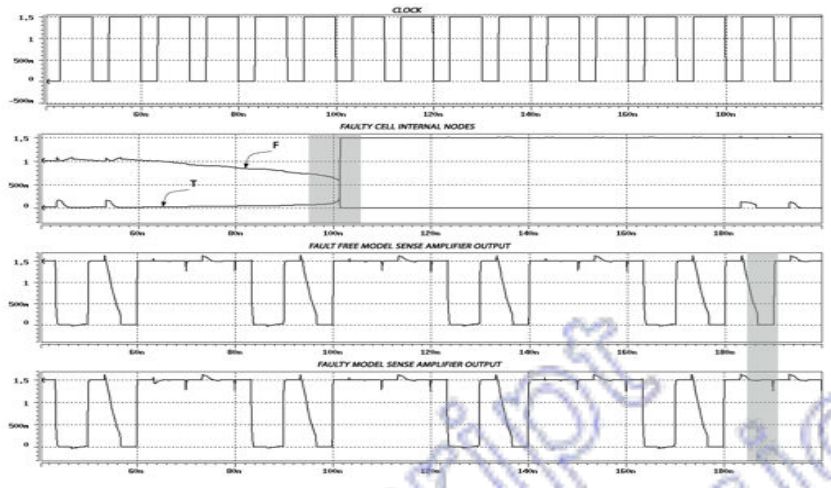

Fig. 3. Electrical simulation for DFR4 with $T = 125°C$ and $R = 64.0M\Omega$

References

1. Al-Ars, Z., van de Goor, A.: Static and dynamic behavior of memory cell array spot defects in embedded drams. IEEE Trans. on Comp. 52(3), 293–309 (2003)
2. Al-Ars, Z., Hamdioui, S., Mueller, G., van de Goor, A.: Framework for fault analysis and test generation in drams. In: Proceedings Design, Automation and Test in Europe, 2005. pp. 1020–1021 (2005)
3. Benso, A., Bosio, A., Di Carlo, S., Di Natale, G., Prinetto, P.: March test generation revealed. IEEE Trans. Comput. 57(12), 1704 –1713 (Dec 2008)
4. Cheng, K.L., Chih-Wea, W., Jih-Nung, L., Yung-Fa, C., Chih-Tsun, H., Cheng-Wen, W.: Fame: a fault-pattern based memory failure analysis framework. In: International Conference on Computer Aided Design, 2003. ICCAD-2003. pp. 595–598 (9-13 Nov 2003)
5. Davis, L.: Handbook of genetic algorithms. Van Nostrand Reinhold (1991)
6. Dilillo, L., Girard, P., Pravossoudovitch, S., Virazel, A., Borri, S., Hage-Hassan, M.: Resistive-open defects in embedded-sram core cells: analysis and march test solution. In: IEEE 13th Asian Test Symposium ATS 2004. pp. 266–271 (November 15–17 2004)
7. Dilillo, L., Girard, P., Pravossoudovitch, S., Virazel, A., Hage-Hassan, M.: Data retention fault in sram memories: analysis and detection procedures. In: 23rd IEEE VLSI Test Symposium (VTS 2005). pp. 183–188 (May, 1–5 2005)
8. Hamdioui, S., Al-Ars, Z., van de Goor, A.: Opens and delay faults in cmos ram address decoders. IEEE Trans. Comput. 55(12), 1630–1639 (Dec 2006)
9. Huang, R.F., Chou, Y.F., We, C.W.: Defect oriented fault analysis for sram. In: Proceedings of the 12th Asian Test Symposium. pp. 1–6 (2003)
10. Niggemeyer, N., Redeker, M., Ottersted, J.: Integration of non-classical faults in standard march tests. In: IEEE International Workshop on Memory Technology, Design and Testing, MTDT'98. pp. 91–96 (Aug 24–26 1998)