ELECTRONICS AND ELECTRICAL ENGINEERING
ISSN 1392 – 1215 2012. No. 3(119)
ELEKTRONIKA IR ELEKTROTECHNIKA

SYSTEM ENGINEERING, COMPUTER TECHNOLOGY
T 120
SISTEMŲ INŽINERIJA, KOMPIUTERINĖS TECHNOLOGIJOS

A Multibackground March Test for Static Neighborhood Pattern-Sensitive Faults in Random-Access Memories

C. Huzum, P. Cascaval

Department of Computer Science and Engineering,"Gheorghe Asachi" Technical University of Iași,
Prof. dr. doc. Dimitrie Mangeron Str., no. 27, Iași, România, phone: +40-232-231343, e-mail: chuzum@cs.tuiasi.ro

crossref http://dx.doi.org/10.5755/j01.eee.119.3.1369

Introduction

Rapid increase of density in the integrated circuits has an immediate effect upon memory testing. On one hand, the capacity of random-access memories chips enhances, thus increasing the test time and cost; on the other hand, the density of memory circuits grows, therefore more failure modes and faults need to be taken into account in order to obtain a good quality product. Accordingly, there are two conflicting constraints that need to be dealt with when considering a test algorithm: reducing the number of memory operations in order to permit large capacity memories to be tested in an appropriate period of time and covering a larger variety of memory faults [1, 2].

As a result of the increasing coupling effect triggered by the growing density of memory circuits, the pattern-sensitive fault (PSF) is becoming an important fault model [3–5]. The PSF model is a type of coupling fault, with several aggressor cells (4, 9 etc.) and only one victim cell. In this work, the neighborhood PSF (NPSF) has been considered. This is a particular PSF, in which the aggressor cells are located in the physical neighborhood of the victim cell. The NPSF model was first defined by Hayes in 1980 [3]. He also devised a memory test for this model [3]. Soon after that, Suk and Reddy have proposed a new memory test [6] based on a bipartite method. This test divides the memory cells into two partitions and applies a sequence of transitions to cover all possible victim-aggressor combinations. Unfortunately, for the memory chips currently used, the test proposed by Suk and Reddy needs a long time to perform. In 2002, other more efficient March tests were given by Cheng, Tsai, and Wu, namely: CM-79N and March-100N [7]. More recently, Julie, Wan Zuha and Sidek use a modified version of March-100N for diagnosis of SRAM [8]. In all these papers the authors have limited themselves only to the class of simple faults. In this work, we have also focused on the problem of testing the linked neighborhood pattern-sensitive faults.

Notations, definitions and fault classifications

An operation sequence that results in a difference between the observed and the expected memory behaviour is called a *sensitizing operation sequence* (*S*). The observed memory behaviour that deviates from the expected one is called *faulty behaviour* (*F*). In order to specify a certain fault, one has to specify the *S*, together with the corresponding faulty behaviour *F*, and the read result (*R*) of *S*, in case it is a read operation. The combination of *S*, *F* and *R* for a given memory failure is called a *Fault Primitive* (FP), and is usually denoted as <*S*/*F*/*R*>. The concept of FPs allows for establishing a complete framework of all memory faults. Some classifications of FPs can be made based on different and independent factors of *S* [9]:

a) Depending on the number of simultaneous operations required in the *S*, FPs are classified into *single-port* and *multi-port* faults:

- *single-ports faults:* These are FPs that require at the most one port in order to sensitize a fault. Note that single-port faults can be sensitized in single-port as well as in multi-port memories,
- *multi-port faults:* These are FPs that can only sensitize a fault by performing two or more operations simultaneously via different ports.

b) Depending on the number of sequential operations required in the *S*, FPs are classified into *static faults* and *dynamic faults*. Let *#O* be the number of different operations carried out sequentially in a *S*:

- *static faults:* These are FPs which sensitize a fault by performing at most one operation in the memory (*#O*=0 or *#O*=1),
- *dynamic faults*: These are FPs that perform more than one operation sequentially in order to sensitize a fault (*#O*>1).

c) Depending on the way FPs manifest themselves, they can be divided into *simple faults* and *linked faults*:

- *simple faults:* These are faults which cannot be influenced by another fault. That means that the behaviour of a simple fault cannot change the behaviour of another one; therefore masking cannot occur,
- *linked faults:* These are faults that do influence the behaviour of each other. That means that the behaviour of a certain fault can change the behaviour of another one such that masking can occur. Note that linked faults consist of two or more simple faults.

In this work, single-port, static faults are considered. From here on, the term ‘fault’ refers to a single-port, static, simple fault and the term ‘linked fault’ means single-port, static, linked fault.

The following notations are usually used to describe operations on RAMs:

- *r*0 (*r*1) denotes a read 0 (1) operation from a cell;
- *w*0 (*w*1) denotes a write 0 (1) operation into a cell;
- ↑ (↓) denotes an up (down) transition due to a certain sensitizing operation.

The neighborhood pattern-sensitive fault model

RAM faults can also be divided into *single-cell* and *multi-cell* faults. Single-cell faults consist of FPs involving a single cell, while multi-cell faults consists of FPs involving more than one cell. In this work, we consider a particular class of multi-cell faults (also called *coupling faults*), namely the pattern sensitive faults (PSF). The PSF is a coupling fault, which affects the content of a memory cell (called the *victim cell* or the *base cell*), or the ability to change its content, when other memory cells (called *aggressor cells*) have certain patterns. It is unnecessary and unrealistic to consider all possible patterns of all the memory cells, therefore simplified models of neighborhood pattern sensitive faults (NPSF) have been introduced. In these models, the aggressor cells are limited to the physical neighborhood of the victim cell. Depending on the number of aggressor cells, NPSF can be divided into several types, but only two of those are more commonly used: Type-1 NPSF and Type-2 NPSF, with four and eight aggressor cells, respectively, as illustrated in Fig. 1 [10].

Like in the most previous works, in this paper only the Type-1 NPSF has been considered.

Due to the features of the NPSF model, the general notation for a FP is particularized, thus in the rest of this paper a FP is denoted as <N W E S; B/B$_f$> [7], where:

- N, W, E, S describes the sensitizing value or operation in the aggressor cells, placed as presented in Fig. 1a;
- B describes the correct value or transition in the base cell;
- B$_f$ shows the faulty value or transition of the base cell.

Note that N, W, E, S, B and B$_f$ ∈{0, 1, ↑, ↓}.

Depending on the behavior of the fault, the NPSFs can be divided into three classes [10], namely:

- Static NPSF (SNPSF): the base cell is forced to a certain value when the aggressor cells have a certain pattern. An example of a static NPSF is FP$_1$=<0100; 0/1>, where the base cell is forced to 1 when the aggressor cells have the pattern 0100;
- Passive NPSF (PNPSF) reflects the impossibility of the base cell to execute a transition due to the existence of a certain pattern in the aggressor cells. An example of a PNPSFs is FP$_2$=<1100; ↓/1>, where the base cell cannot switch from 1 to 0 because the aggressor cells have the pattern 1100;
- Active NPSF (ANPSF): a certain transition in one of the aggressor cells forces the victim cell to change its value when the other aggressor cells (also called *enabling cells*) have a certain pattern. An example of this class of faults is FP$_3$=<10↑0; 0/1>, where a transition in the E cell causes the base cell to flip from 0 to 1 when the N, W and S cells have the pattern 100.

The model of NPSFs we have considered can be entirely described by the set of FPs presented in Table 1. There are 192 fault primitives of which 32 SNPSFs, 32 PNPSFs, and 128 ANPSFs.

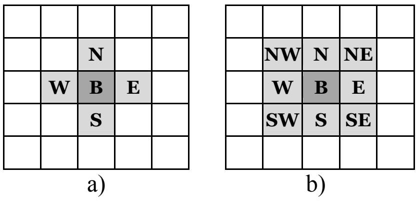


Fig. 1. Common types of neighborhood pattern sensitive faults: a – Type-1 NPSF; b – Type-2 NPSF

The linked neighborhood pattern-sensitive faults are NPSFs that influence the behavior of each other, such that masking can occur. Therefore, they are more difficult to detect.

Table 1. List of NPSF Primitives

Fault primitives		Fault type
<*x y z t*; 0 / 1>	*x*, *y*, *z*, *t* ∈ {0, 1}	SNPSF
<*x y z t*; 1 / 0>		
<*x y z t*; ↑ / 0>	*x*, *y*, *z*, *t* ∈ {0, 1}	PNPSF
<*x y z t*; ↓ / 1>		
<*x y z* ↑; 0 / 1>	*x*, *y*, *z* ∈ {0, 1}	ANPSF
<*x y z* ↓; 0 / 1>		
<*x y z* ↑; 1 / 0>		
<*x y z* ↓; 1 / 0>		
<*x y* ↑ *z*; 0 / 1>		
<*x y* ↓ *z*; 0 / 1>		
<*x y* ↑ *z*; 1 / 0>		
<*x y* ↓ *z*; 1 / 0>		
<*x* ↑ *y z*; 0 / 1>		
<*x* ↓ *y z*; 0 / 1>		
<*x* ↑ *y z*; 1 / 0>		
<*x* ↓ *y z*; 1 / 0>		
<↑ *x y z*; 0 / 1>		
<↓ *x y z*; 0 / 1>		
<↑ *x y z*; 1 / 0>		
<↓ *x y z*; 1 / 0>		

A linked fault consists of two or more FPs (even number) with contrary effects on the same victim (base) cell. For example, take a NPSF fault in which an up transition into cell W changes the state of cell B from 1 to 0, when the enabling cells have the pattern 100, whereas an up transition into cell S changes the state of cell B from 0 to 1, when the enabling cells have the pattern 011. This is a linked fault that can be modeled by two FPs: FP_1=<1↑00; 1/ 0>, and FP_2=<011↑; 0/1>.

Even though there are many papers that deal with the issue of linked faults, such as [11] for two-cell linked faults, or [14] for three-cell linked faults, we do not know any work treating this model of linked NPSFs.

A new memory test – March-76N

In order to describe the memory test, first some notations regarding the March tests are given. Usually, a complete March test is delimited by '{...}' bracket pair, while a March element is delimited by the '(...)' bracket pair. March elements are separated by semicolons, and the operations within a March element are separated by commas. Note that all operations of a March element are performed at a certain address, before proceeding to the next address. The whole memory is checked homogeneously in either one of two orders: ascending address order ($\Uparrow$) or descending address order ($\Downarrow$). When the address order is not relevant, the symbol $\Updownarrow$ is used.

As stated in [7], if a certain March element is applied to a solid background (all ones or all zeros), when we are reading from or writing to a cell C, then all the cells with higher address than C have the same state, while those with lower address than C also have the same state. So, most of the NPSF faults cannot be activated by applying March elements to these solid backgrounds. Therefore, the necessity of a March test that runs under several different backgrounds arises. These are called multibackground March tests [12, 13].

The test we propose, called March-76N, uses sixteen different backgrounds, denoted by BG_0, BG_1, ..., BG_{15}, as presented in Fig. 2. For a certain cell, let *a* be the value within the background, and *b* the complement of *a*. For example, when initializing the memory with BG1, *a* is 0 for the lines 1, 3, 4, 6, 7, 9 etc. of the memory, and 1 for lines 2, 5, 8 etc. These backgrounds were selected for the test in order to create all victim-aggressor combinations necessary for activating all the faults in the NPSF model.

March-76N forms an alternating series of March elements and background changes (as Cockburn proposed in [15]) and has the following structure:

$$\{ \Updownarrow (w0); [\Uparrow(ra, wb); \Uparrow(rb, wa); CBG_i], i = 1, 2, \dots, 16 \}, \quad (1)$$

where CBG_i is a sequence for checking and changing from the current background to the next. More exactly, the sequence CBG_i has the following meaning:

- CBG_i, i =1, 2, ..., 15, changes the background from BG_{i-1} to BG_i. Note that when changing from one background to the next, only the cells that must change states are written. Each write operation is also preceded (for checking) by a read operation. We can observe in Fig. 2 that the backgrounds were ordered such as the difference between every two successive backgrounds consists of three out of the nine cells so that all background changes affect only a trinity of the cells.
- CBG_{16} reads the whole memory for the finally checking ($CBG_{16} = \Updownarrow(ra)$).

Consequently, March-76N comprises the following operations:

- *N* operations for the first initialization;
- [2*N* +2*N* + 2×3*N*/9] ×15 = 70*N* operations, for the first fifteen series of March elements and background changes;
- 2*N*+2*N*+*N*=5*N*, for the sixteenth series of March elements and final checking.

BG_0	BG_1	BG_2	BG_3
0 0 0	0 0 0	1 1 1	0 1 1
0 0 0	1 1 1	1 1 1	0 1 1
0 0 0	0 0 0	0 0 0	1 0 0

BG_4	BG_5	BG_6	BG_7
0 1 1	1 0 0	1 0 0	1 1 0
1 0 0	1 0 0	0 1 1	0 0 1
1 0 0	1 0 0	1 0 0	1 1 0

BG_8	BG_9	BG_{10}	BG_{11}
0 0 1	1 0 1	0 1 0	0 1 0
0 0 1	1 0 1	1 0 1	0 1 0
1 1 0	0 1 0	0 1 0	0 1 0

BG_{12}	BG_{13}	BG_{14}	BG_{15}
1 1 0	0 0 1	1 0 1	1 1 1
1 1 0	1 1 0	0 1 0	0 0 0
1 1 0	1 1 0	0 1 0	0 0 0

Fig. 2. The 16 backgrounds for the test March-76N

Therefore, the length of March-76N is *N* +70*N* +5*N* = 76*N*.

A simulation study has been made in order to determine whether March-76N covers all simple and also all linked NPSFs. Taking into account that the patterns of the backgrounds used by this test are composed of 3×3 cells, for the simulation study, the memory cells have been divided into nine mutually disjoint subsets. These are denoted by S_1, S_2, ..., S_9, depending on their location. Let *r* and *c* be the row address and the column address, respectively, of a memory cell. The cell with the address (*r*, *c*) belongs to a certain subset according to the following rule:

$$\text{the cell } (r,c) \in \begin{cases} S_1,, \text{ if } c \% 3 = 0 \text{ and } r \% 3 = 0, \\ S_2, \text{ if } c \% 3 = 0 \text{ and } r \% 3 = 1, \\ S_3, \text{ if } c \% 3 = 0 \text{ and } r \% 3 = 2, \\ S_4, \text{ if } c \% 3 = 1 \text{ and } r \% 3 = 0, \\ S_5, \text{ if } c \% 3 = 1 \text{ and } r \% 3 = 1, \\ S_6, \text{ if } c \% 3 = 1 \text{ and } r \% 3 = 2, \\ S_7, \text{ if } c \% 3 = 2 \text{ and } r \% 3 = 0, \\ S_8, \text{ if } c \% 3 = 2 \text{ and } r \% 3 = 1, \\ S_9, \text{ if } c \% 3 = 2 \text{ and } r \% 3 = 2, \end{cases} \quad (2)$$

where "%" denotes the modulo operation.

For a memory array with 8 rows and 8 columns, these nine subsets of cells are illustrated in Fig. 3.

	0	1	2	3	4	5	6	7
0	S_1	S_4	S_7	S_1	S_4	S_7	S_1	S_4
1	S_2	S_5	S_8	S_2	S_5	S_8	S_2	S_5
2	S_3	S_6	S_9	S_3	S_6	S_9	S_3	S_6
3	S_1	S_4	S_7	S_1	S_4	S_7	S_1	S_4
4	S_2	S_5	S_8	S_2	S_5	S_8	S_2	S_5
5	S_3	S_6	S_9	S_3	S_6	S_9	S_3	S_6
6	S_1	S_4	S_7	S_1	S_4	S_7	S_1	S_4
7	S_2	S_5	S_8	S_2	S_5	S_8	S_2	S_5

Fig. 3. The cell subsets for an 8×8 memory chip array

Note that every memory cell that belongs to the same cell subset will support the same operations during the test March-76N. Moreover, if two base cells belong to the same subset, their aggressor cells will support, respectively, the same initializations. Hence, for the simulation study, only nine locations (one for each subset) have to be considered for the base cell.

For the linked fault coverage evaluation, the linked faults consisting of two simple faults have been considered. There are 96 NPSFs that flip the base cell from 0 to 1 and 96 that flip it from 1 to 0. Consequently, a total of 96 × 96 = 9216 linked faults have been considered for the simulation study.

The conclusion of the study is that March-76N covers entirely the model of simple and linked NPSFs.

Theorem: March-76N is able to detect all simple and linked NPSFs in our model.

Table 2. Bit sequence for each background and for each subset of base cell

		S_1		**S_2**		**S_3**		**S_4**		**S_5**		**S_6**		**S_7**		**S_8**		**S_9**	
BG$_0$	0 0 0 0 0 0 0 0 0	0 0 0 0 0	I_1	0 0 0 0 0	I_1	0 0 0 0 0	I_1	0 0 0 0 0	I_1	0 0 0 0 0	I_1	0 0 0 0 0	I_1	0 0 0 0 0	I_1	0 0 0 0 0	I_1	0 0 0 0 0	I_1
BG$_1$	0 0 0 1 1 1 0 0 0	0 0 0 0 1	I_2	0 1 1 1 0	I_{16}	1 0 0 0 0	I_3'	0 0 0 0 1	I_2	0 1 1 1 0	I_{16}	1 0 0 0 0	I_3'	0 0 0 0 1	I_2	0 1 1 1 0	I_{16}	1 0 0 0 0	I_3'
BG$_2$	1 1 1 1 1 1 0 0 0	0 1 1 1 1	I_3	1 1 1 1 0	I_2'	1 0 0 0 1	I_{16}'	0 1 1 1 1	I_3	1 1 1 1 0	I_2'	1 0 0 0 1	I_{16}'	0 1 1 1 1	I_3	1 1 1 1 0	I_2'	1 0 0 0 1	I_{16}'
BG$_3$	0 1 1 0 1 1 1 0 0	1 1 0 1 0	I_4	0 1 0 1 1	I_7'	0 0 1 0 0	I_5'	0 0 1 1 1	I_9'	1 0 1 1 0	I_8	1 1 0 0 1	I_{14}	0 1 1 0 1	I_{10}	1 1 1 0 0	I_{11}'	1 0 0 1 1	I_{15}'
BG$_4$	0 1 1 1 0 0 1 0 0	1 1 0 1 1	I_5	0 0 1 0 1	I_4'	1 0 1 0 0	I_7	0 0 1 1 0	I_{14}'	1 1 0 0 0	I_9	0 1 0 0 1	I_8'	0 1 1 0 0	I_{15}	1 0 0 1 0	I_{10}'	0 0 0 1 1	I_{11}
BG$_5$	1 0 0 1 0 0 1 0 0	1 0 1 0 1	I_6	1 0 1 0 1	I_6	1 0 1 0 1	I_6	0 1 0 0 0	I_{13}'	0 1 0 0 0	I_{13}'	0 1 0 0 0	I_{13}'	0 0 0 1 0	I_{12}	0 0 0 1 0	I_{12}	0 0 0 1 0	I_{12}
BG$_6$	1 0 0 0 1 1 1 0 0	1 0 1 0 0	I_7	1 1 0 1 1	I_5	0 0 1 0 1	I_4'	0 1 0 0 1	I_8'	0 0 1 1 0	I_{14}'	1 1 0 0 0	I_9	0 0 0 1 1	I_{11}	0 1 1 0 0	I_{15}	1 0 0 1 0	I_{10}'
BG$_7$	1 1 0 0 0 1 1 1 0	1 0 1 1 0	I_8	1 1 0 0 1	I_{14}	0 0 1 1 1	I_9'	1 1 1 0 0	I_{11}'	1 0 0 1 1	I_{15}'	0 1 1 0 1	I_{10}	0 1 0 1 1	I_7'	0 0 1 0 0	I_5'	1 1 0 1 0	I_4
BG$_8$	0 0 1 0 0 1 1 1 0	1 1 0 0 0	I_9	0 1 0 0 1	I_8'	0 0 1 1 0	I_{14}'	1 0 0 1 0	I_{10}'	0 0 0 1 1	I_{11}	0 1 1 0 0	I_{15}	0 0 1 0 1	I_4'	1 0 1 0 0	I_7	1 1 0 1 1	I_5
B$_9$	1 0 1 1 0 1 0 1 0	0 1 1 0 1	I_{10}	1 1 1 0 0	I_{11}'	1 0 0 1 1	I_{15}'	1 1 0 1 0	I_4	0 1 0 1 1	I_7'	0 0 1 0 0	I_5'	0 0 1 1 1	I_9'	1 0 1 1 0	I_8	1 1 0 0 1	I_{14}
BG$_{10}$	0 1 0 1 0 1 0 1 0	0 0 0 1 1	I_{11}	0 1 1 0 0	I_{15}	1 0 0 1 0	I_{10}'	1 0 1 0 0	I_7	1 1 0 1 1	I_5	0 0 1 0 1	I_4'	0 1 0 0 1	I_8'	0 0 1 1 0	I_{14}'	1 1 0 0 0	I_9
BG$_{11}$	0 1 0 0 1 0 0 1 0	0 0 0 1 0	I_{12}	0 0 0 1 0	I_{12}	0 0 0 1 0	I_{12}	1 0 1 0 1	I_6	1 0 1 0 1	I_6	1 0 1 0 1	I_6	0 1 0 0 0	I_{13}'	0 1 0 0 0	I_{13}'	0 1 0 0 0	I_{13}'
BG$_{12}$	1 1 0 1 1 0 1 1 0	1 0 1 1 1	I_{13}	1 0 1 1 1	I_{13}	1 0 1 1 1	I_{13}	1 1 1 0 1	I_{12}'	1 1 1 0 1	I_{12}'	1 1 1 0 1	I_{12}'	0 1 0 1 0	I_6'	0 1 0 1 0	I_6'	0 1 0 1 0	I_6'
BG$_{13}$	0 0 1 1 1 0 1 1 0	1 1 0 0 1	I_{14}	0 0 1 1 1	I_9'	1 0 1 1 0	I_8	1 0 0 1 1	I_{15}'	0 1 1 0 1	I_{10}	1 1 1 0 0	I_{11}'	0 0 1 0 0	I_5'	1 1 0 1 0	I_4	0 1 0 1 1	I_7'
BG$_{14}$	1 0 1 0 1 0 0 1 0	0 1 1 0 0	I_{15}	1 0 0 1 0	I_{10}'	0 0 0 1 1	I_{11}	1 1 0 1 1	I_5	0 0 1 0 1	I_4'	1 0 1 0 0	I_7	0 0 1 1 0	I_{14}'	1 1 0 0 0	I_9	0 1 0 0 1	I_8'
BG$_{15}$	1 1 1 0 0 0 0 0 0	0 1 1 1 0	I_{16}	1 0 0 0 0	I_3'	0 0 0 0 1	I_2	0 1 1 1 0	I_{16}	1 0 0 0 0	I_3'	0 0 0 0 1	I_2	0 1 1 1 0	I_{16}	1 0 0 0 0	I_3'	0 0 0 0 1	I_2

Proof:

A. March-76N is able to detect all simple NPSFs.

When the memory is initialized with a certain background, this causes the appearance of different bit sequences in neighborhoods that have the base cell in different subsets. For example, when the memory is initialized with BG_2, if the base cell belongs to S_1, its neighborhood contains the bit sequence presented in Fig. 4a, and if the base cell belongs to S_5, its neighborhood consists of the bit sequence presented in Fig. 4b.

Table 2 presents for each background, the bit sequences that appear in the neighborhood of the base cell when it belongs to each subset. Note that, whichever subset the base cell belongs to, the initialization of the memory cells with all backgrounds will determine the appearance of all 16 possible bit sequences or their complements (denoted by I_1, I_2,...,I_{16}, and I_1', I_2', ... I_{16}', respectively, as shown in Table 3) in a different order.

On the other hand, another simulation study has been made in order to determine which faults are detected by applying each March element of the memory test on each of the 16 bit sequences.

Table 3 shows these faults grouped by the test sequence that detects them. Each of the sixteen series of our test is composed of three test sequences (two March elements and a test sequence for changing the background) identified with a superscript (x), where $x \in \{1, 2, 3\}$, as follows:

$$[\Uparrow(ra, wb)^{(1)};\ \Uparrow(rb, wa)^{(2)};\ CBG_i^{(3)}],\ i=1, 2, \ldots, 16. \quad (3)$$

To simplify the writing in Table 3, the '<' and '>' symbols usually used to denote a fault primitive have been neglected. Moreover, a bit sequence from a neighborhood is written as NWBES and it contains, respectively, the values in the north, west, base, east, and south cell.

Table 3 includes all 192 fault primitives of this NPSF model. Note that, by applying the two March elements (1) and (2) on a bit sequence or on their complement, the same operations are carried out in the memory, and consequently, the same memory faults are detected. Therefore, when the three test sequences in March-76N are applied to all of the 16 bit sequences I_1, I_2, ..., I_{16}, or to their complements, all NPSF simple faults are detected.

Note that Table 2 shows that whichever subset the base cell belongs to, applying all of the 16 backgrounds to the memory will determine the appearance in the neighborhood of the base cell of all 16 bit sequences or their complements in a different order. In conclusion, wherever the base cell is located, March-76N covers the whole model of simple NPSFs.

B. March-76N is able to detect all linked NPSFs.

Let us consider the pairs of simple faults that can mask their effects while running the March-76N memory test. The simple faults that belong to such pairs have opposite effect upon the base cell and are both activated between two consecutive read operations of the base cell. For example, when running the test on bit sequence I_1, the simple faults pairs shown in Table 4 can mask their effects, hiding an incorrect behaviour of the memory. All the pairs composed of two active and/or passive simple faults that can be activated between two consecutive reading operations have been considered.

Table 3. Faults detected by each bit sequence within March-76N

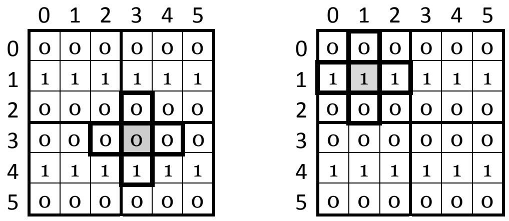
Bit sequence	Notation	NPSF faults (grouped by the test sequences that detect them)			
		(1)	*(2)*		*(3)*
00000	**I_1**	↑000;0/1 1↑00;0/1	1100;↑/0 1100; -/0 11↑0;1/0 111↑;1/0	↓111;1/0 0↓11;1/0	0011;↓/1 0011; -/1 00↓1;0/1 000↓;0/1
00001	**I_2**	↑001;0/1 1↑01;0/1	1101;↑/0 1101; -/0 11↑1;1/0 111↓;1/0	↓110;1/0 0↓10;1/0	0010;↓/1 0010; -/1 00↓0;0/1 000↑;0/1
01111	**I_3**	↑111;1/0 1↓11;1/0	1011;↓/1 1011; -/1 10↓1;0/1 100↓;0/1	↓000;0/1 0↑00;0/1	0100;↑/0 0100; -/0 01↑0;1/0 011↑;1/0
11010	**I_4**	↓110;0/1 0↓10;0/1	0010;↑/0 0010; -/0 00↓0;1/0 000↑;1/0	↑001;1/0 1↑01;1/0	1101;↓/1 1101; -/1 11↑1;0/1 111↓;0/1
11011	**I_5**	↓111;0/1 0↓11;0/1	0011;↑/0 0011; -/0 00↓1;1/0 000↓;1/0	↑000;1/0 1↑00;1/0	1100;↓/1 1100; -/1 11↑0;0/1 111↑;0/1
10101	**I_6**	↓001;1/0 0↑01;1/0	0101;↓/1 0101; -/1 01↑1;0/1 011↓;0/1	↑110;0/1 1↓10;0/1	1010;↑/0 0101; -/0 10↓0;1/0 100↑;1/0
10100	**I_7**	↓000;1/0 0↑00;1/0	0100;↓/1 0100; -/1 01↑0;0/1 011↑;0/1	↑111;0/1 1↓11;0/1	1011;↑/0 1011; -/0 10↓1;1/0 100↓;1/0
10110	**I_8**	↓010;1/0 0↑10;1/0	0110;↓/1 0110; -/1 01↓0;0/1 010↑;0/1	↑101;0/1 1↓01;0/1	1001;↑/0 1001; -/0 10↑0;1/0 101↓;1/0
11000	**I_9**	↓100;0/1 0↓00;0/1	0000;↑/0 0000; -/0 00↑0;1/0 001↑;1/0	↑011;1/0 1↑11;1/0	1111;↓/1 1111; -/1 11↓1;0/1 110↓;0/1
01101	**I_{10}**	↑101;1/0 1↓01;1/0	1001;↓/1 1001; -/1 10↑1;0/1 101↓;0/1	↓010;0/1 0↑10;0/1	0110;↑/0 0110; -/0 01↑0;1/0 010↑;1/0
00011	**I_{11}**	↑011;0/1 1↑11;0/1	1111;↑/0 1111; -/0 11↓1;1/0 110↓;1/0	↓100;1/0 0↓00;1/0	0000;↓/1 0000; -/1 00↑0;0/1 001↑;0/1
00010	**I_{12}**	↑010;0/1 1↑10;0/1	1110;↑/0 1110; -/0 11↓0;1/0 110↑;1/0	↓101;1/0 0↓01;1/0	0001;↓/1 0001; -/1 00↑1;0/1 001↓;0/1
10111	**I_{13}**	↓011;1/0 0↑11;1/0	0111;↓/1 0111; -/1 01↓1;0/1 010↓;0/1	↑100;0/1 1↓00;0/1	1000;↑/0 1000; -/0 10↑0;1/0 101↑;1/0
11001	**I_{14}**	↓101;0/1 0↓01;0/1	0001;↑/0 0001; -/0 00↑1;1/0 001↓;1/0	↑010;1/0 1↑10;1/0	1110;↓/1 1110; -/1 11↓0;0/1 110↑;0/1
01100	**I_{15}**	↑100;1/0 1↓00;1/0	1000;↓/1 1000; -/1 10↑0;0/1 101↑;0/1	↓011;0/1 0↑11;0/1	0111;↑/0 0111; -/0 01↓1;1/0 010↓;1/0
01110	**I_{16}**	↑110;1/0 1↓10;1/0	1010;↓/1 1010;-/1 10↓0;0/1 100↑;0/1	↓001;0/1 0↑01;0/1	0101;↑/0 0101;-/0 01↑1;1/0 011↓;1/0

Table 4. Simple fault pairs that can mask their effect when applying March-76N on bit sequence I_1

↑000;0/1 1↑00;1/0	1100;↑/0 11↑0;0/1	1100;↑/0 111↑;0/1	1100;↑/0 ↓111;0/1	1100;↑/0 0↓11;0/1
11↑0;1/0 111↑;0/1	11↑0;1/0 ↓111;0/1	11↑0;1/0 0↓11;0/1	111↑;1/0 ↓111;0/1	111↑;1/0 0↓11;0/1
↓111;1/0 0↓11;0/1	0011;↓/1 00↓1;1/0	0011;↓/1 000↓;1/0	00↓1;0/1 000↓;1/0	

Note that the second fault in each pair in Table 4 is detected by applying the test on bit sequence I_5 (see Table 3). I_1 and I_5 have the same value for the base cell and complementary values for the aggressor cells (I_1 is 00000, and I_5 is 11011). Thus, for each of the pairs in Table 4, by applying March-76N to the bit sequence I_5, only the second fault in each of the pairs is activated, such that the masking will not occur.

In the same way, the linked NPSFs that can appear while applying March-76N on a certain bit sequence will be detected by applying the test on another bit sequence that has the same value for the base cell and opposite values for the aggressor cells.



a)

	0	1	2	3	4	5
0	0	0	0	0	0	0
1	1	1	1	1	1	1
2	0	0	0	0	0	0
3	0	0	0	0	0	0
4	1	1	1	1	1	1
5	0	0	0	0	0	0

b)

	0	1	2	3	4	5
0	0	0	0	0	0	0
1	1	1	1	1	1	1
2	0	0	0	0	0	0
3	0	0	0	0	0	0
4	1	1	1	1	1	1
5	0	0	0	0	0	0

Fig. 4. Bits sequences that appear in BG_2: a – Base cell belongs to S_1; b – Base cell belongs to S_5

Therefore, every linked NPSF consisting of a pair of simple NPSFs that mask their effect while applying the memory test on a certain background (denoted by BG_m) will be detected when applying the test on another background (denoted by BG_d). Let I_m be the bit sequence that appears in the neighborhood of the base cell when the memory is initialized with BG_m. Then, BG_d is the background that causes the appearance of the pair of bit sequence I_m (as can be found in Table 5) in the neighborhood of the base cell.

Conclusions

The test we propose improves the performance of the test March-76N given by Cheng, Tsai, and Wu [7], which is the shortest published memory test dedicated to this NPSF model. In comparison with CM-79N, March-76N is shorter with 3.33*N*. Moreover, March-76N is able to detect all linked faults whereas, as presented in [16], March-76N cannot cover entirely the set of linked NPSFs.

References

1. **Adams R. D.** High performance memory testing: design principles, fault modelling and self–test – USA: Kluwer Academic Publishers, 2003 – 247 p.
2. **Hamdioui S.** Testing static random access memories: defects, fault models and test patterns – The Netherlands: Kluwer Academic Publishers, 2004. – 240 p.
3. **Hayes J. P**. Testing memories for single–cell pattern–sensitive fault // IEEE Trans. Comput., 1980. – Vol. 29. – P. 249–254.
4. **Kang D. C., Cho S. B.** An efficient build–in self–test algorithm for neighborhood pattern sensitive faults in high–density memories" // Proc. 4th Korea–Russia Int. Symp. Science and Technology, 2000. – Vol. 2. – P. 218–223.
5. **Cockburn B. F.** Deterministic tests for detecting scrambled pattern–sensitive faults in RAMs // Proc. IEEE Int. Workshop Memory Technology, Design and Testing (MTDT), 1995. – P. 117–122.
6. **Suk D. S., Reddy M.** Test procedures for a class of pattern–sensitive faults in semiconductor random–access memories // IEEE Trans. Comput., 1980. – Vol. 29. – P. 419–429.
7. **Cheng K. L., Tsai M. F., Wu C. W.** Neighborhood pattern–sensitive fault testing and diagnostics for random–access memories // IEEE Trans. On CAD, 2002. – Vol. 21. – No. 11. – P. 1328–1336.
8. **Julie R. R., Wan Zuha W. H., Sidek R. M.** 12N test procedure for NPSF testing and diagnosis for SRAMs // Proc. IEEE Int. Conf. on Semiconductor Electronics, 2008. – P. 430–435. DOI: 10.1109/SMELEC.2008.4770357.
9. **Hamdioui S., van de Goor A. J., Rodgers M.** March SS: A test for all static simple RAM faults // Proc. of IEEE Int'l Workshop on Memory Technology, Design and Testing, 2002. – P. 95–100.
10. **Van de Goor A. J.** Testing semiconductor memories: theory and practice – John Wiley & Sons Inc, 1991. – 512 p.
11. **Benso A., Bosio A., Di Carlo S., Di Natale G., Prinetto P**. A 22n March Test for Realistic Static Linked Faults in SRAMs // Proceedings of the Eleventh IEEE European Test Symposium, 2006. – P. 49–54.
12. **Yarmolik V., Klimets Y., Demidenko S.** March PS(23N) test for DRAM pattern–sensitive faults // Proc. Seventh IEEE Asian Test Symp.(ATS), 1998. – P. 354–357.
13. **Cheng K. L., Tsai M. F., Wu C. W.** Efficient neighborhood pattern–sensitive fault test algorithms for semiconductor memories // Proc. IEEE VLSI Test Symp. (VTS), 2001. – P. 225–237.
14. **Caşcaval P., Bennett S., Hutanu C**. Efficient March Tests for a Reduced 3–Coupling and 4–Coupling Faults in Random–Access Memories // J. Electronic Testing: Theory and Applications, 2004. – Vol. 20(3) – P. 227–243.
15. **Cockburn B. F.** Deterministic tests for detecting single V–coupling faults in RAMs // J. Electronic Testing. Theory and Applications, 1994. – Vol. 5(1). – P. 91–113.
16. **Huzum C., Caşcaval P.** Linked Neighborhood Pattern–Sensitive Faults in Random–Access Memories. A Fault Coverage Evaluation // Proc. of 14th ICSTC, 2010. – P. 241–245.

Received 2011 03 10
Accepted after revision 2011 09 15

C. Huzum, P. Cascaval. A Multibackground March Test for Static Neighborhood Pattern-Sensitive Faults in Random-Access Memories // Electronics and Electrical Engineering. – Kaunas: Technologija, 2012. – No. 3(119). – P. 81–86.

A multibackground March test (March-76N) for a model of static neighborhood pattern sensitive faults (NPSFs) in $N \times 1$ random-access memories is presented. March-76N is able to cover both simple and linked NPSFs. As any other test dedicated to the NPSFs, March-76N assumes that the storage cells are arranged in a rectangular grid and the mapping from logical addresses to physical cell locations is known completely. With a length of 76*N*, this March test is more efficient than other published tests dedicated to this model. Ill. 4, bibl. 16, tabl. 4 (in English; abstracts in English and Lithuanian).

C. Huzum, P. Cascaval. Daugiafonis operatyviosios atminties statinių kaimyninių struktūrų klaidų testas // Elektronika ir elektrotechnika. – Kaunas: Technologija, 2012. – Nr. 3(119). – P. 81–86.

Pateikiamas daugiafonis March testas (March-76N), skirtas statinių kaimyninių struktūrų klaidų (SKSK) aptikimo operatyviojoje atmintyje modeliui. March-76N gali aptikti tiek paprastas, tiek susijusias minėto tipo klaidas. Kaip ir kituose testuose, skirtuose SKSK aptikti, March-76N daroma prielaida, kad elementai yra išrikiuoti stačiakampiame tinklelyje ir kreipimasis iš loginių adresų į fizinę elemento vietą yra visiškai žinomas. March testas yra efektyvesnis už kitus skelbtus šiam modeliui skirtus testus. Il. 4, bibl. 16, lent. 4 (anglų kalba; santraukos anglų ir lietuvių k.).