Physical Design Oriented DRAM Neighborhood Pattern Sensitive Fault Testing

Yiorgos Sfikas and Yiorgos Tsiatouhas
University of Ioannina
Department of Computer Science,
P.O. Box 1186, 45110 Ioannina, Greece (Hellas)
{gsfikas, tsiatouhas}@cs.uoi.gr

***Abstract*–Although the Neighborhood Pattern Sensitive Fault (NPSF) model is recognized as a high quality fault model for memory arrays, the excessive test application time cost associated with it, compared to other fault models, restricts its wide adoption for memory testing. In this work we exploit the physical design (layout) of folded DRAM memory arrays to introduce a new neighborhood type for NPSF testing and a pertinent test and locate algorithm. This algorithm reduces drastically the test application time (about 58% with respect to the well known Type-1 neighborhood) aiming to make the NPSF model also a cost attractive choice. In addition, we introduce the Neighborhood Word-Line Sensitive Fault model and the corresponding test algorithm to cover those faults along with NPSFs, achieving test application time cost reduction from 33% to 41%, depending on various assumptions, with respect to the Type-1 neighborhood.**

Keywords: Memory Testing, DRAM Testing, Neighborhood Pattern Sensitive Fault (NPSF) model, Neighborhood Word-Line Sensitive Fault (NWSF) model, Δ-Type neighborhood.

I. INTRODUCTION

The *Pattern Sensitive Fault (PSF)* model can be considered as the most general case of coupling faults where all memory cells (n the number) are involved. PSF is modeling the susceptibility of a cell in a memory array to the contents and transitions of all other cells [1-3]. This susceptibility is due to the high densities of nanometer technology DRAMs as well as various fault mechanisms (static and dynamic leakage currents [4] like the field-inversion current between two adjacent storage cells [3], [5]) that are present in memory arrays. However, testing DRAMs for PSFs is almost infeasible due to the prohibitive test application time as it requires a test set of $(3n^2+2n)2^n$ patterns [3].

Alternatively, a more realistic and well established memory fault model is the *Neighborhood Pattern Sensitive Fault (NPSF)* model [2]. According to this, the content of a cell or the ability to apply a desired value at that cell, is affected by the values or transitions on the values of k neighbor cells in the memory array. In practice the cells (called *deleted neighborhood*) affecting the operation of a cell (called *base cell*) are those with physical proximity to that particular cell. The combination of the base cell and the deleted neighborhood is called *neighborhood* while the corresponding faults are called *Neighborhood Pattern Sensitive Faults* (*NPSFs*). The NPSF model is distinguished in three categories:

- *Active NPSF (ANPSF) or Dynamic NPSF*, where the base cell changes its contents due to a change in the deleted neighborhood pattern.
- *Passive NPSF (PNPSF)*, where the contents of a cell cannot be changed due to a certain neighborhood pattern.
- *Static NPSF (SNPSF)*, where the contents of a base cell are forced to a certain state due to certain deleted neighborhood pattern.

Various types of neighborhoods have been proposed in the open literature. In [6] the *Row/Column Pattern Sensitive Faults* have been proposed where the contents of a cell become sensitive to the contents of the row and column containing the cell. The *Disturb Neighborhood Pattern Sensitive Faults* have been introduced in [7], which involves k cells in the memory array. According to this, the base cell (*victim cell*) makes an up (↑) or a down (↓) transition due to a r_y (read with expected value y) or w_y (write value y) operation ($y \in \{0, 1\}$) applied to one deleted neighborhood cell (*aggressor cell*), while the remaining k-2 deleted neighborhood cells (*enabling cells*) contain a certain pattern (*enabling pattern*). In [3], [8] a four-cell neighborhood (the *T-Type*) has been proposed targeting the detection of *Bit-Line Neighborhood Pattern Sensitive Faults (NBLSFs)* along with NPSFs.

However, the most common neighborhoods are the Type-1 and Type-2 neighborhoods [1, 2]. The Type-1 neighborhood is consisting of the four adjacent cells to a base cell, these on the same row and the same column, which form the deleted neighborhood. Thus, this is a five cells neighborhood, as it is shown in Fig. 1(a). The Type-2 neighborhood consists of cells within m_1 columns to the west, m_2 rows to the north, m_3 columns to the east and m_4 rows to the south of a base cell. Commonly $m_1=m_2=m_3=m_4=1$ and the neighborhood contains nine cells as it is shown in Fig. 1(b).

Aiming to detect in common and at an optimum test application time, active, passive and static NPSFs (APSNPSFs) with respect to the Type-1 and Type-2

IEEE International Symposium on Design and Diagnostics of Electronic Circuits and Systems

978-1-4244-3339-1/09/$25.00 ©2009 IEEE

neighborhoods, every possible neighborhood with base cell every cell of the memory array should be written with the patterns of an Eulerian sequence [2]. This sequence consists of 161 5-bit patterns in the case of Type-1 neighborhood and 4609 9-bit patterns in the case of Type-2 neighborhood (in a k-bit Eulerian sequence the number of patterns is $k2^k+1$). Moreover, two methods to further accelerate the test application time have been adopted for the write operation of the test sequence in the memory array, the *tiling* and the *two-groups* methods [2], [5].

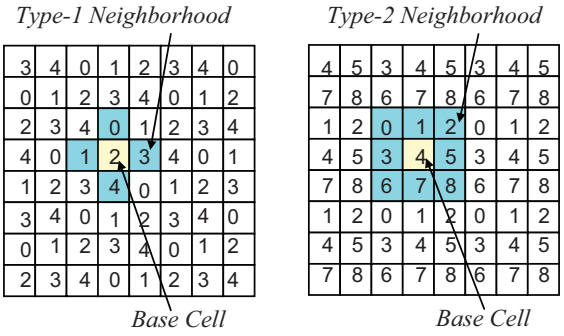

Fig. 1: The Type-1 and Type-2 neighborhoods

With the tiling method, the memory is totally covered by a group of neighborhoods which do not overlap. In Fig. 1 the Type-1 and Type-2 tiling neighborhoods are shown respectively. The cells of each neighborhood are numbered from 0 to 4 and from 0 to 8 respectively.

The two-group method is based on the duality of the cells and can only be used for Type-1 neighborhoods. Using this method the cells are divided into two groups by a checkerboard pattern. A cell is a base cell in one group and a deleted neighborhood cell in the other group, and vice versa. This approach reduces the number of write operations by a factor of 4.

As the memory density increases with CMOS technology evolution in the nanometer era, NPSFs become more and more important especially for DRAMs [7], [9-13]. However, although the NPSF model is preferred since it provides better fault modeling in memory arrays [10-11], the associated test application time cost still makes it unattractive for memory testing. Earlier efforts to reduce this cost [14] failed to provide full fault location capabilities. Moreover, since none of the above NPSF types consider the physical design (layout) of the memory array, a reduced fault coverage may result at an additive cost of a large number of redundant operations.

In this work we discuss a four-cells neighborhood type for NPSFs in DRAMs that is based on the physical design of the memory array. In addition test and locate algorithms are proposed that provide high fault coverage at reduced test application time cost compared to previous approaches. The paper is organized as follows. In Section II the common layout design of folded DRAM memory arrays is presented. Next, in Section III inabilities of present neighborhood types are discussed and the new type (called Δ-Type) is proposed. Finally, in Section IV the conclusions are drawn.

II. DRAM Memory Array Physical Design

The main advantage of DRAM is the low area cost due to the simplicity of its one transistor – one capacitor memory cell. Fig. 2 presents the general layout of a folded DRAM memory array [15-17]. The memory cell (mcell) size is 4F long (2 lines + 2 spaces) and 2F wide (a line and a space) resulting in a cell area of $8F^2$, where F is the minimum lithographic feature size of the technology defined as one-half of the word-line or the bit-line pitch. The cells are arranged back-to-back and share a common bit-line. The distance between back-to-back storage capacitors is equal to 5F. In this work there is not any assumption on the capacitor type; either trench capacitor or stacked capacitor DRAMs can be considered.

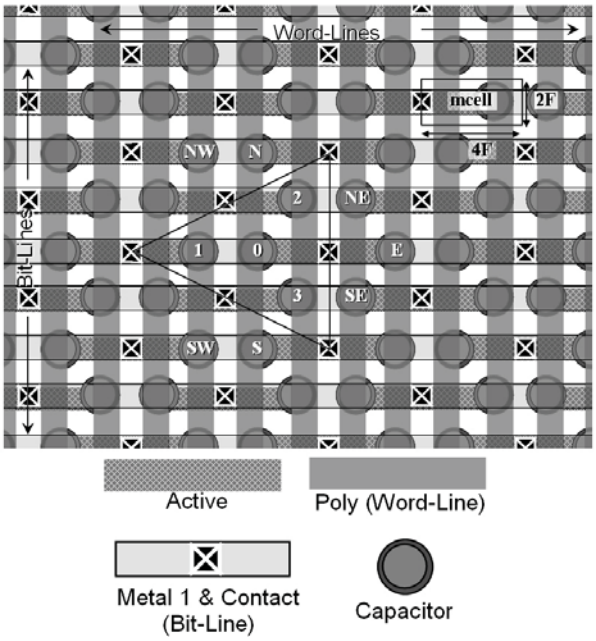

Fig. 2: DRAM memory array layout

The $8F^2$ cell layout provides superior signal-to-noise performance (bit-line noise rejection) with respect to sub-$8F^2$ layouts, because of its folded bit-line architecture capability [4], [15]. This architecture uses adjacent data and reference bit-lines providing excellent matching and noise rejection. Sub-$8F^2$ cell layouts require two levels of bit-line wiring to achieve equivalent matching and noise rejection. Moreover, $8F^2$ layouts with folded bit-line architecture pose no problem on sense amplifier layout and area. Each sense amplifier serves a total of four bit-lines in contrast to the open bit-line architectures used in sub-$8F^2$ layouts where each sense amplifier serves only two bit-lines. In those cases the number of sense amplifiers is duplicated and the required silicon area is increased resulting in a significant drawback since sense amplifiers occupy about 10% of the total chip area in modern DRAMs [4].

III. THE Δ-TYPE NEIGHBORHOOD FOR NPSFS TESTING

A. Δ-Type neighborhood

Focusing at the physical design of the memory array in Fig. 2 it is easy to realize that none of the neighbor cells (the immediate adjacent cells with physical proximity) of any cell in the memory array (e.g. cell-0) belongs to the same word-line with this cell, as it is supposed in the various NPSF types in the open literature [2].

According to the Type-1 neighborhood and considering cell-0 as the base cell, cells N, S and E belong in the deleted neighborhood, although the distance of their storage nodes from the storage node of cell-0 is large. Cells N and S share the same word-line with cell-0 and their distance is equal to 3F. Cell-E shares the same bit-line with cell-0 and lies on an adjacent word-line, but the distance of its storage node from the corresponding node of cell-0 (equal to 5F) does not sustain the assumption of physical proximity with it. Moreover, the common bit-line contact comes in between the two storage nodes so that in case of an interaction of cell-0 towards the direction of cell-E this will be with the common bit-line and not with cell-E. Consequently, it is not expected that cell-E may have any realistic contribution to the pattern of the pertinent deleted neighborhood affecting the base cell. In addition, cells 1, 2 and 3 only partially fulfill the criteria to form the fourth cell of the Type-1 deleted neighborhood shown in Fig. 1(a). Cells 2 and 3 also belong to an adjacent word-line to the word-line of cell-0 but do not lie on the same bit-line with it, as the Type-1 neighborhood assumes, while there is not any criterion to select only one of them as the fourth cell. The storage nodes of cell-1 and cell-0 are adjacent (with the smallest distance of 1F) but this is not the case for their word-lines as the Type-1 neighborhood claims. A solution may be to extend the neighborhood to a six-cells one in order to add both cells 2 and 3 in the deleted neighborhood or possibly a seven-cells neighborhood to take into account also cell-1.

The same questioning arises trying to form the Type-2 neighborhood. Cells 1, 2, 3, N, NW, S, SW may be considered as the seven of the eight cells of the Type-2 deleted neighborhood, although many of them do not fulfill the criteria of Fig. 1(b) and their distance from the base cell is questionable. Moreover, as we mentioned earlier cell-E lies far away from cell-0, while there is not any good criterion to select only one of cells NE and SE as the eighth cell. Once again we may have to extend the neighborhood to a ten-cells or an eleven-cells neighborhood in order to include both cells WE and SE as well as cell-E respectively in the deleted neighborhood. However, these extensions in the Type-1 and Type-2 neighborhoods increase considerably the complexity of testing and the test application time without offering significant fault coverage advantages since many cells under consideration in these deleted neighborhoods (cells N, S, E, NW, SW, NE and SE) are far away from the base cell. Finally, according to the T-Type neighborhood its deleted neighborhood consists of cells 1, N and S that are not the best selection with respect to their physical proximity to cell-0. Cells 2 and 3 should be added in the T-Type deleted neighborhood for better fault coverage, extending the neighborhood to a six-cells one, with the same complexity and test application time drawbacks as in the previous cases.

From Fig. 2 it is easy to realize that the neighbor cells of cell-0 are the three cells numbered 1, 2 and 3 [18-19]. Their distance from cell-0 is less than $\sqrt{2}F$. This observation motivated us to consider a new neighborhood consisting of four-cells, those inside the triangle of Fig. 2, where the cell-0 is the base cell while cells 1, 2 and 3 are the deleted neighborhood. We call this the *Triangle-Type* or *Δ-Type* neighborhood. The layout based definition of the Δ-Type neighborhood and the reduced number of deleted neighborhood cells implies higher fault coverage at a reduced test complexity and test application time with respect to earlier neighborhood types.

The number of cells in the Δ-Type neighborhood is equal to four (k=4). Thus, the Eulerian sequence for testing APSNPSF in it consists of $k2^k+1=65$ test patterns. Such a 4-bit Eulerian sequence is shown in Table I.

TABLE I
A 4-BIT EULERIAN SEQUENCE FOR Δ-TYPE NPSF TESTING

$P_4P_3P_2P_1$	$P_4P_3P_2P_1$	$P_4P_3P_2P_1$	$P_4P_3P_2P_1$
0000	1001	0101	0011
0001	0001	0001	0001
0011	0000	1001	0101
0010	1000	1101	0111
0110	1010	1100	1111
0111	0010	1000	1101
0101	0011	0000	1001
0100	1011	0100	1011
1100	1111	0110	1010
1101	0111	0010	1000
1111	0110	1010	1100
1110	1110	1110	1110
1010	1100	1111	0110
1011	0100	1011	0100
1001	0101	0011	0000
1000	1101	0111	0010
			0000

Furthermore, in order to reduce the complexity of the write operation, during the application of the Eulerian test sequence for NPSF testing, the tiling method has been adopted. In Fig. 3 the tiling method for the Δ-Type neighborhood is illustrated. According to this, the memory is tiled by non-overlapping triangle neighborhoods. There are two kinds of triangle neighborhoods, right oriented triangles where the “top” cell-1 lies at the right side of the neighborhood, and left oriented triangles where the “top” cell-1 lies at the left side of the neighborhood. In addition, note that every word-line contains cells assigned with only two numbers either 0 and 1 or 2 and 3. The tiling method applies each test pattern simultaneously on all tiling neighborhoods. In general, it is assumed in the open literature that the memory read and write operations are of equal time cost. Thus, applying a test to locate APSNPSF algorithm, like the TLAPNPSF1T and TLAPNPSF2T algorithms in [2], we can detect and locate all active, passive and static NPSFs related to the Δ-Type neighborhood with a cost of 82n operations, where n is the number of cells in the memory array. This algorithm is shown in Fig. 4 and we will

similarly call it the TLAPNPSFΔT algorithm. The *write(j)* ($1 \le j \le k2^k$) procedure in Fig. 4 writes the j-th pattern of the 4-bit Eulerian sequence to the neighborhoods of the memory array. For each pattern only half of the word-lines are involved in the write operations and in each of these word-lines half of the cells are written. This is due to the fact that each word-line contains only two cell number assignments and only one bit changes between subsequent patterns in the Eulerian sequence. The algorithm locates all ANPSFs, PNPSFs and SNPSFs because each execution of *write(j)* applies one active or passive neighborhood pattern in each neighborhood, while finally all possible 4-bit patterns are written in it for SNPSF testing. Moreover, the Eulerian sequence ensures that the active and passive patterns are generated optimally (i.e. single bit transitions without repeating previously generated subsequences). The algorithm cost is analyzed as follows: a) in step (1) there are n write and n read operations and b) in step (2) there are $nk2^k/k=n2^k$ write operations to apply the patterns of the Eulerian sequence and $nk2^k$ read operations, where k is the number of cells in the neighborhood. Thus, there is a total of $n[2+(k+1)2^k]$ operations and since k=4 for the Δ-Type neighborhood, the cost turns to be 82n operations. This is a significant test application time reduction with respect to TLAPNPSF1T and TLAPNPSF2T algorithms for the classic Type-1 and Type-2 neighborhoods, where the corresponding cost is equal to 194n and 5122n operations respectively. The reduction of the test cost with respect to the Type-1 neighborhood, which is of greater importance, is 57.7%.

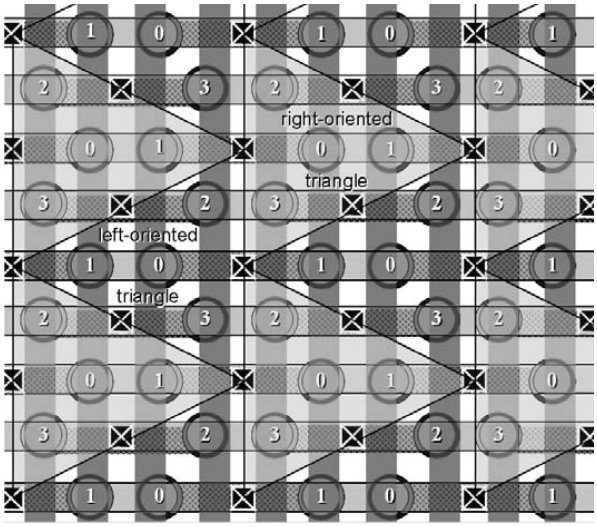

Fig. 3: The tiling method for the Δ-Type neighborhood

(1) Initialize all cells with 0; read 0 from all cells;
(2) For j:=1 to $k2^k$ do
begin
write(j);
read all cells;
end;

Fig. 4: The TLAPNPSFΔT algorithm (k=4)

B. Neighborhood word-line sensitive faults (NWSFs)

In subsection III.A we have shown that cell-E in Fig. 2 is not expected to have any realistic contribution to the pattern of the deleted neighborhood affecting the base cell-0, due to its distance (5F) from this base cell (no physical proximity of their storage nodes). However, the word-line that activates cell-E is adjacent to the word-line of cell-0, while in addition both cells share the same bit-line. Consequently, read or write operations on cell-E may disturb cell-0 leading it to an erroneous logic state. For example, consider the strong capacitive coupling between two adjacent word-lines, which are located on the same conducting layer and routed side by side for a long distance. The activation of a word-line may raise the voltage level of its adjacent word-line. Thus, depending on the value of cells' common bit-line, the leakage current of cell-0 may be increased when read/write operations are performed on cell-E [20-21]. We will call this kind of faults *Neighborhood Word-Line Sensitive Faults* (*NWSF*). From this point of view a five-cells "cross" type neighborhood involving cells 0, 1, 2, 3 and E (like Type-1 although not identical) seems to be more attractive since it covers a possible influence on cell-0 from read/write operations on cell-E. On the other hand, the NPSF model assumption that all cells in a deleted neighborhood affect the base cell with their data pattern, implies that this cross type neighborhood, which includes cell-E, will induce a lot of redundant operations increasing the test application time cost.

A better approach is to keep the Δ-Type neighborhood and enhance the TLAPNPSFΔT algorithm with a limited number of proper additional operations (whenever is needed) to ensure the detection of NWSFs along with NPSFs. This way we can achieve the same fault coverage as the five-cells "cross" type neighborhood but at a considerably lower cost. Initially, note that the base cell-0 and cell-E in Fig. 2, share a common drain and a common bit-line. Moreover, cell-E is the top cell-1 of an adjacent neighbourhood. Next, we will call cell-E the *adjoining* cell of the neighbourhood under consideration. The TLAPNPSFΔT algorithm ensures all possible conditions required for NWSF testing since the base cell is assigned as cell-0 and the adjoining cell is assigned as cell-1. Consequently, every possible transition-write operation on the adjoining cell will take place for every possible value on the base cell during the TLAPNPSFΔT algorithm application. Thus, the cost of testing remains 82n operations.

However, in order to cover extreme conditions we may wish to test NWSFs for every possible combination in the deleted neighborhood of the base cell. This implies that for NWSFs detection we must perform both a 0→1 transition-write and a 1→0 transition-write on the adjoining cell, for every possible pattern in the neighborhood of the base cell. Using the TLAPNPSFΔT algorithm, the assumption "for every possible pattern in the deleted neighborhood" is not true since both the top cell of a neighbourhood and the adjoining cell are assigned as cell-1 in their neighbourhoods. Thus, at least one combination related to the top cell value is not applicable. For the rest cells 0, 2 and 3 of the neighborhood every combination is feasible.

Next, we will provide a solution to the above problem. Initially, with $\updownarrow$A|BT> we denote a right-oriented triangle where a transition-write to the logic value A is performed on the adjoining cell with the base cell carrying the logic value B and top cell carrying the logic value T {A, B, T∈[0, 1]}. Similarly with <TB|A$\updownarrow$ we refer to a left oriented triangle. During the application of the TLAPNPSFΔT algorithm we observe the following cases. Choosing the left-to-right direction for the algorithm application (that is the write operations start from the leftmost word-line), when a transition-write operation on the adjoining cell of a right-oriented triangle is performed, the only possible condition is $\updownarrow$X|Y$\overline{X}$>, where X, Y∈[0, 1]. The condition $\updownarrow$X|YX> is not feasible for right-oriented triangles. This is true since: a) prior to the transition-write operation on the adjoining cell both the top cell and the adjoining cell always have the same value due to the previous pattern application and b) the top cell is written after the adjoining cell. On the other side, in left-oriented triangles the top cell is written before the adjoining cell so that a transition-write operation on the latter always result in the <XY|X$\updownarrow$ condition. In those triangles, the condition <$\overline{X}$Y|X$\updownarrow$ is not feasible. Choosing the opposite direction (right-to-left) for the algorithm application, the complementary situations stand for the right and left oriented triangles respectively. This observation provides a first solution to the NWSF testing problem since applying twice the TLAPNPSFΔT algorithm, in both directions, every possible pattern in the neighborhood will be present for both transition-write operations on the adjoining cell (i.e. all conditions $\updownarrow$X|Y$\overline{X}$>, $\updownarrow$X|YX>, <XY|X$\updownarrow$ and <$\overline{X}$Y|X$\updownarrow$ are feasible for every test pattern application). Note that cells 2 and 3 are not considered in the previous discussion since the required combinations in the neighbourhood, with respect to these cells, are feasible for every transition-write on the adjoining cell.

Obviously, a double application of the algorithm increases undesirably the test cost to 162n operations, although it remains less than the 194n operations for the Type-1 neighborhood. However, many of these operations are redundant and unnecessary since during the first algorithm application all NPSFs of the Δ-Type neighborhood are detected and located and the same stands for half of the NWSFs. Indeed, in the left-to-right oriented application the conditions $\updownarrow$X|Y$\overline{X}$> and <XY|X$\updownarrow$ are already covered. In the second application (pass), with the opposite direction, the only target is the detection and location of the rest half NWSFs (i.e. conditions $\updownarrow$X|YX> and <$\overline{X}$Y|X$\updownarrow$). To attain this, in the second application only the cells that play the role of the base cell in triplets $\updownarrow$A|BT> and <TB|A$\updownarrow$ are read for each test pattern that is written in the memory array and a transition-write operation is performed on the adjoining cell. Thus, the rest two conditions $\updownarrow$X|YX> and <$\overline{X}$Y|X$\updownarrow$ are also covered. An equivalent description for the second pass procedure is as follows: after each test pattern application and for every word-line where write operations were performed, read the cells that have not been written. This approach is based on the following observation: only the word-lines that contain cells with the number assignment of the base cell also contain cells with the number assignment of the top/adjoining cell; consequently only these word-lines will be accessed for write operations on the top/adjoining cells and the rest cells (the base cells) are those that must be read.

Consequently, the total cost is reduced to 114n operations and is analyzed as follows: 82n is the cost for the initial application of the TLAPNPSFΔT algorithm and 16n write operations plus 16n read operations is the cost for the second pass to cover the rest half NWSFs. Note that in the second pass we do not have to initialize the memory array since it is already initialized by the last pattern of the first pass. The reduction of the test cost with respect to the Type-1 neighborhood is 41.2%. Next, in Fig. 5 the above algorithm is presented under the name TLAPNPWSFΔT.

(1) Initialize all cells with 0; read 0 from all cells;
(2) For j:=1 to $k2^k$ do
begin
write(j); (left-to-right)
read all cells;
end;
(3) For j:= 1 to $k2^k$ do
begin
write(j); (right-to-left)
read the cells that were not written in the word-lines involved in the write operations;
end;

Fig. 5: The TLAPNPWSFΔT algorithm (k=4)

C. The worst case scenario

In the above discussion we treated NPSFs and NWSFs as independent fault models and this seems to be a realistic approach. Consequently, a possible worst case scenario is the simultaneous occurrence of both NWSF and NPSF faults in a neighborhood. We will call this case *Neighborhood Word-Line and Pattern Sensitive Faults* (*NWPSFs*). In that situation and considering right-oriented triangles, the activation of a NWSF due to a transition-write operation on the adjoining cell may be masked by the activation of an ANPSF in the neighborhood due the subsequent transition-write on the top cell. The opposite fault activation sequence may happen in left-oriented triangles also resulting in fault masking.

To overcome this masking problem, it is required to perform a number of extra read operations. Considering the above paradigm, a read operation on the base cell between the write operation on its adjoining cell and the write operation on its top cell will solve the problem since the effect of the first activated fault mechanism will be detected before the activation of the second one. Note that these extra read operations are performed during the first pass of the memory array and do not release us from reading the whole memory array after each test pattern application, as the TLAPNPSFΔT algorithm requests. The second pass is also performed exactly as it has been described in sub-section III.B, without extra readings, just to cover the rest NWSF conditions. In this second pass masking problems do not bother us since it is guaranteed that any possible NPSF has been already detected during the first pass.

An equivalent description for the first pass procedure is as follows: for every pattern that is written by the *write(j)* procedure, after the write operations on a word-line, read the cells of this word line that have not be written. This approach is based on: a) the earlier observation that only the word-lines accessed for write operations on the top/adjoining cells also contain the base cells that must be read and b) accessing sequentially the word-lines for the write operations, the word-line of the base cell in a neighborhood is always between the word-lines of its top and its adjoining cell.

The extra cost, for all patterns, is 16n read operations during the first pass of the memory array. Consequently, the total cost is increased to 114n+16n=130n operations. The test cost reduction with respect to the Type-1 neighborhood still remains substantial and is equal to 33%. This third algorithm is presented in Fig. 6 under the name TLAPNCPWSFΔT.

(1) Initialize all cells with 0; read 0 from all cells;
(2) For j:=1 to $k2^k$ do
begin
write(j); (left-to-right)
{during write(j), after the write operations on a word-line, read the cells of this word-line that have not be written}
read all cells;
end;
(3) For j:= 1 to $k2^k$ do
begin
write(j); (right-to-left)
read the cells that were not written in the word-lines involved in the write operations;
end;

Fig. 6: The TLAPNCPWSFΔT algorithm (k=4)

Finally, note that NWSFs may be also related to the second adjacent word-line to the word-line of cell-0. However, since cells 2 and 3 belong to this word-line as well as in the neighborhood of cell-0, these NWSFs are covered along with NPSFs during the application of any of the above three algorithms.

IV. Conclusions

A neighborhood based on the physical design of folded DRAM memory arrays is proposed in this work for NPSF testing. This neighborhood consists of four memory cells in a triangle fashion and is called the Δ-Type neighborhood. In addition, the neighborhood word-line sensitive fault model (NWSF) is introduced and the ability to cover NWSFs along with NPSFs is discussed. Three new testing algorithms have been derived for NPSFs and NWSFs detection and location with test application time cost equal to 82n, 114n and 130n respectively. Correspondingly the cost reduction, with respect to the pertinent testing algorithm that is used for the Type-1 neighborhood, is 57.7%, 41.2% and 33%. Considering also the cost of recently proposed March algorithms that rise up to 70n and 100n [22-23], the proposed in this work algorithms turn to be a viable test solution for nanometer DRAMs.

References

[1] J. P. Hayes, "Testing Memories for Single-Cell Pattern-Sensitive Faults," IEEE Transactions on Computers, vol. 29, no. 3, pp. 249-254, 1980.
[2] A. J. van de Goor, "Testing Semiconductor Memories-Theory and Practice," John Wiley & Sons Ltd., 1991.
[3] D-C. Kang, S.M. Park and S-B Cho, "An Efficient Built-In Self Test Algorithm for Neighborhood Pattern and Bit-Line-Sensitive Faults in High Density Memories," ETRI Journal, vol. 26, no. 6, pp. 520-534, 2004.
[4] J.A. Mandelman, R.H. Bennard, G.B Bronner, J.K. DeBrosse, R. Divakaruni, Y. Li and C.J. Radens, "Challenges and Future Directions for the Scaling of Dynamic Random-Access Memory (DRAM)," IBM Journal of Research and Development, vol. 46, no. 2/3, pp. 187-212, 2002.
[5] P. Mazumder and K. Chakraborty, "Testing and Testable Design of High-Density Random-Access Memories," Kluwer Academic Publishers, 1996.
[6] M. Franklin, K. Saluja and K. Kinoshita, "A Built In Self Test Algorithm for Row/Column Pattern Sensitive Faults in RAMs," IEEE Journal of Solid-State Circuits, vol. 25, no. 2, pp. 514-524, 1990.
[7] A.J. van de Goor and I.B.S. Tlili, "Disturb Neighborhood Pattern Sensitive Fault," IEEE Int. VLSI Test Symposium, pp. 37-45, 1997.
[8] D-C. Kang and S-B Cho, "An Efficient Built-In Self Test Algorithm for Neighborhood Pattern and Bit-Line-Sensitive Faults in High Density Memories," IEEE KORUS, pp. 218-223, 2000.
[9] S. Banerjee, D.R. Chowdhury and B.B. Bhattacharya, "A Programmable Built-In Self-Test for Embedded DRAMs," IEEE Int. Workshop on Memory Technology Design and Testing, pp. 58-63, 2005.
[10] S. Boutobza, M. Nicolaidis, K. M. Lamara and A. Costa, "A Transparent Based Programmable Memory BIST," IEEE European Test Symposium, pp. 89-94, 2006.
[11] S. Banerjee, D.R. Chowdhury and B.B. Bhattacharya, "A Programmable Built-In Self-Test for Embedded Memory Cores," IETE Technical Review, vol. 24, no. 4, pp. 287-311, 2007.
[12] J.Y. Kim, S.J. Hong and j. Kim, "Parallely Testable Design for Detection of Neighborhood Pattern Sensitive Faults in High Density DRAMs," IEEE Int. Symposium on Circuits and Systems, pp. 5854-5857, 2005.
[13] Y-J. Huang and J-F. Li, "Testing Active Neighborhood Pattern-Sensitive Faults of Ternary Content Addressable Memories," IEEE European Test Symposium, pp. 55-60, 2006.
[14] K-L. Cheng, M-F. Tsai and C-W Wu, "Efficient Neighborhood Pattern-Sensitive Fault Test Algorithms for Semiconductor Memories," IEEE VLSI Test Symposium, pp. 225-230, 2001.
[15] B. Keeth and R.J. Baker, "DRAM Circuit Design: A Tutorial," IEEE Press, Series on Microelectronic Systems, 2001.
[16] Y. Matsubara, et al, "Fully Compatible Integration of High Density Embedded DRAM with 65nm CMOS Technology (CMOS5)," IEEE Electron Devices Meeting, pp. 423-426, 2003.
[17] C-H. Chung and J-W. Chien, "Memories Having Charge Storage Node at Least Partially Located in a Trench in a Semiconductor Substrate and Electrically Coupled to a Source/Drain Region Formed in the Substrate," US Patent 7,348,622 B2, 2008.
[18] H.D. Oberle and P. Muhmenthaler, "Test Patten Development and Evaluation for DRAMs with Fault Simulator RAMSIM," IEEE Int. Test Conference, pp. 548-555, 1991.
[19] Z. Al-Ars, S. Hamdioui, G. Gaydadjiev, "Optimizing Test Length for Soft Faults in DRAM Devices," IEEE VLSI Test Sym., pp. 59-66, 2007.
[20] F. Karimi, S. Irrinki, T. Crosby, N. Park and F. Lombardi, "Parallel Testing of Multi-Port Static Random Access Memories," Microelectronics Journal, vol. 34, pp. 3-21, 2003.
[21] D-S. Min and D. Langer, "Multiple Twisted Data Line Techniques for Coupling Noise Reduction in Embedded DRAMS," IEEE Custom Integrated Circuits Conference, pp. 231-234, 1999.
[22] G. Harutunyan, V. Vardanian and Y. Zorian, "Minimal March Tests for Dynamic Faults in Random Access Memories," IEEE European Test Symposium, pp. 43-48, 2006.
[23] A. Benso, A. Boisio, S. Carlo, G. Natale, P. Prinetto, "Automatic March Test Generation for Static and Dynamic Faults in SRAMS," IEEE European Test Symposium, pp. 122-127, 2005.