Efficient $O(\sqrt{n})$ BIST Algorithms for DDNPS Faults in Dual Port Memories

A. A. Amin, M. Y. Osman, R. E. Abdel-Aal and H. Al-Muhtaseb[1]
King Fahd University of Petroleum & Minerals,
Dhahran 31261, Saudi Arabia

ABSTRACT

The testability problem of dual port memories is investigated. Architectural modifications to enhance the testability by allowing multiple access of memory cells with minimal overhead on both silicon area and device performance are described. New fault models are proposed and efficient $O(\sqrt{n})$ test algorithms are described for both the memory array and the address decoders. The new fault models account for the simultaneous dual access property of the device. In addition to the classical static neighborhood pattern sensitive faults, the array test algorithm covers a new class of pattern sensitive faults, Duplex Dynamic Neighborhood Pattern Sensitive faults (DDNPSF).

1.0 INTRODUCTION

Dual-port random access memories (DPRAMs) allow simultaneous access of stored data from two ports as compared to access from only one port in conventional single-port RAMs (SPRAMs). Whereas this may be used to speed up the test algorithm, it does complicate the memory array fault model as well as the decoder fault model resulting in more complex test algorithms. Fault modeling for SPRAMs has been thoroughly investigated [15]. However, in spite of the growing use of DPRAMs, limited work on the testability issues of these devices has been reported. An ad-hoc test technique which adopts no specific fault model was described by Raposa [16]. Ad-hoc techniques, however, are only useful for small size memories. A serial test algorithms for an embedded DPRAM was developed in [17] where several simple fault models were adopted. A simple stuck-at fault model was used for the memory array, address decoder and read/write logic. In addition, two types of coupling faults were considered. A static (state) coupling fault model for cells on the same word line, and a dynamic coupling fault model where a read or a write operation to some cell forces a particular state on another cell. Whereas these are mainly single-port faults, a simple bridging fault model was also adopted to accommodate the dual-port nature of the memory. A special shadow write operation was designed to test for bridging faults between bit lines and between word lines of opposite ports. Test algorithms which cover stuck-at faults, bridging faults between adjacent bit lines, and stuck-open as well as stuck-ON faults of some of the memory cell transistors was reported in [13]. These approaches, however, do not take into account complex pattern sensitive failures which become more common as transistor and memory cell sizes get smaller [1]. The notion of a complex coupling fault model for DPRAMs was introduced in [18] where an $O(n^2)$ march test was developed. By imposing some topological restrictions on the relative locations of the coupling and the coupled cells, the test length was reduced to an $O(n)$ complexity [19]. Even though [18] has extended the SPRAM array coupling fault model to account for the special features of DPRAMs, the approach used a single-port decoder fault model which does not account for complexities introduced by the second port.

To properly address the testability problem of DPRAMs, new fault models covering complex pattern sensitive faults should be adopted, the effect of the added complexity of the second port should be considered, and efficient test algorithms should be developed. An efficient test algorithm should not only be of low complexity to maintain reasonable test time for large size memories, but should also be simple enough to implement as on-chip BIST logic without incurring unacceptable chip area overhead or degrading the memory performance. This paper achieves these goals by introducing new array and decoder fault models as well as new circuit modifications to allow parallel access and verification of data in the test mode. This has lead to efficient $O(\sqrt{n})$ test algorithms that are simple enough to implement in BIST logic [22]. The new array fault model covers a new class of pattern sensitive faults, Duplex Dynamic Neighborhood Pattern Sensitive Faults (DDNPSF), which account for failure modes expected in this type of devices. An $O(\sqrt{n})$ array test algorithm covers, in addition to DDNPSF faults, all stuck-at faults, all static neighborhood pattern sensitive faults for a neighborhood of size 5, and a restricted class of complex coupling faults [18]. Moreover, a new decoder fault model is proposed to account for the interaction between the decoders of both ports. An $O(\sqrt{n})$ test algorithm to detect decoder faults is also presented. Both

[1]A. A Amin and M. Y. Osman are with the Computer Engineering Department, R. E. Abdel-Aal is with the Energy Research Laboratory of the Research Institute, and H. Al-Muhtaseb is with the Information and Computer Science Department.

the array and the address decoders test algorithms are suitable for BIST implementation.

The low O($\sqrt{n}$) complexity of the test algorithm is achieved through parallel access of row/word line data. In the test mode, one fourth of the memory cells on any given row can be written into in a single memory cycle. In addition, data from half the memory cells on two different rows are verified simultaneously by the two ports. A similar parallel access approach for SPRAMs has been reported in [2]. Whereas parallel access in [2] has been achieved by modifying the column decoders, we have achieved this through controlling the address inputs of the column decoders instead. In addition to being simpler to implement, this is also more area efficient with an area overhead of only O(*Log n*) as compared to O($\sqrt{n}$) in [2]. Such simplicity and area efficiency are essential requirements for efficient BIST implementation. The number of cells selected in parallel (one fourth of the memory cells on any given row) is independent of the supplied input address as opposed to a similar scheme reported in [20].

The adopted memory array tiling [21] partitions the memory array cells into 8 distinct groups. Such tiling causes the test algorithm complexity to have a higher constant multiplier compared to the 5-group tiling proposed in [2] to obtain minimal test length. Whereas such approach fits the design for testability approach they have adopted, using such tiling would result in a complex BIST implementation because the power-of-two nature of practical memory array sizes does not allow simple division by 5. Thus, in light of the required simplicity of the algorithm and BIST implementation, the 8-group tiling was adopted in spite of the slight increase in the test length. In addition to parallel access of row line data, parallel comparison of data is allowed by two multiple input comparators which flag data errors. An error is flagged whenever non identical data are detected.

In section 2 of the paper, the functional model of DPRAMs is presented. Section 3 illustrates the testability added modifications required for parallel writing and data verification. Details of the array fault model and its test algorithm are given in section 4. In section 5, the decoder fault model is presented and its test algorithm explained. Detailed analysis and BIST implementation of these algorithms can be found in [22].

2.0 FUNCTIONAL MODEL

Typically, a memory chip of size n is organized as a number (p) of sub-arrays of memory cells each of size (r x q) with r rows and q columns ($n = pqr$). To speed up memory testing, the BIST logic is designed to test the p sub-arrays in parallel thus cutting the test time by a factor of p. In the following analysis, only one such sub-array is considered. Each memory cell has two identical access ports; a and b. Each access port is associated with one row/word line and one set of bit/data line(s). Thus, each DPRAM cell has two row lines, one per port, and two sets of bit lines, one per port. Both row lines are identified by the same row address and both sets of bit lines are identified by the same column address.

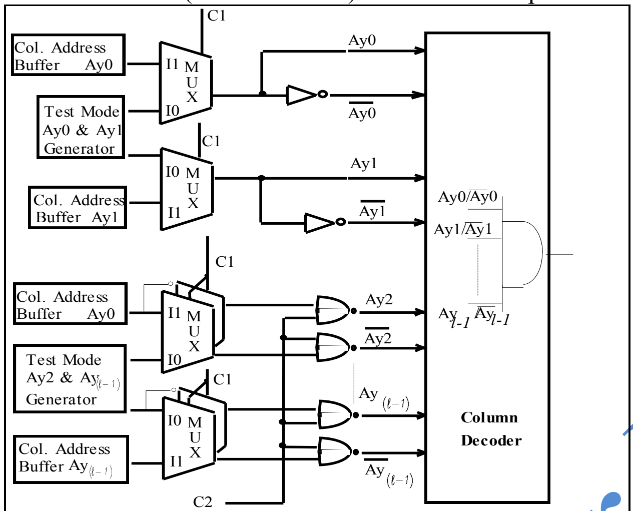
The DPRAM functional model consists of an r x q memory array of DPRAM cells. Each access port has its own row decoder, column decoder, sense amplifiers, and input/output buffers. Control, timing and arbitration circuitry is common to both ports. The model assumes the use of one sense amplifier per bit line.

3.0 TESTABILITY ADDED FEATURES

A number of design modifications are proposed to simplify the BIST implementation logic and allow efficient O($\sqrt{n}$) test algorithms. In the array test mode, the modifications provide parallel access of multiple cells on the two addressed rows. Each row in the memory array is partitioned into exactly four ***sectors***, with each sector having *q/4* bits. Each port may write data into one full sector on any given row, and therefore the column decoder should allow accessing a total of *q/4* bits in parallel. To achieve this, two circuit modifications are necessary. First, the write amplifier should be made powerful enough to drive the accessed bit lines. This, however, should not result in unacceptably high current spikes. If necessary, the write cycle during the test mode can be extended to avoid such spikes. Second, the column decoders of both ports should allow selection of multiple bit lines in the

array test mode. As shown in Figure 2, to implement this feature, only two column addresses (A_{y0} and A_{y1}) are allowed to assume arbitrary values while the true and complement outputs of all other column address buffers will be forced to a logic 1 state (by setting control signal C2 to a logic 0 state).

The array test algorithm includes verification read steps which verify the integrity of stored data in the array. These steps verify that half the array cells (*the base cells*) contain certain identical background data; either all 0's or all 1's. This can be accomplished using *rq/2* single read operations or *rq/4* double (using both ports) read operations. Instead, two coincidence comparators are used to simultaneously verify two sets of data. Each set consists of data stored in half the cells on one of the two accessed row lines (*q/2* cells). The cells of one set (accessed by one port) are the odd-numbered cells, while the cells of the other (accessed by the other port) are the even-numbered ones. One port verifies data of the odd-numbered cells on some row while the other port simultaneously verifies those of the even-numbered cells on another row. Thus, verifying half the array cells (*rq/2* cells) needs only *r/2* such parallel verification steps. According to the memory tiling used, exactly two sectors per row line per port will be involved in such a verifying read operation. Thus, in one verification read step both ports are used to access four sectors on two different rows *in parallel.* Data verification is accomplished by adding one parallel coincidence comparator circuitry for each port. The inputs to the comparator of port a (port b), are the outputs of the sense amplifiers of half the array bit lines corresponding to the even- numbered (odd-numbered) columns of the port.



C1C2	Access Mode
1 1	Normal Mode
0 0	Array Test Mode, (Sector Access)
0 1	Decoder Test Mode, (Single Cell Access)

Figure 2. Modified Column Address Decoder Inputs

Figure 3 shows the logic diagram of the comparator circuitry with inputs from the even-numbered bit lines. The comparator consists of two NOR gates (N1 and N2) the outputs of which (α and β) are inputs to a third NOR gate (N3). The output of N3 is the "Error" flag signal. The inputs to N1 are the true outputs (S_0, S_2, ..., S_{q-2}.) of the even-numbered sense amplifiers connected to half the bit lines of a given port.

The inputs to N2 are the complement outputs of these sense amplifiers ($\bar{S}_0, \bar{S}_2, ..,\bar{S}_{q-2}$.). The "Error" output signal is flagged high only if the data being compared are not identical. If the data being verified are identical, the inputs of either the N1 or the N2 gate will be all 1's, while the inputs of the other will be all 0's. Thus, the output of one of these two NORs will be 0, while the output of the other will be 1 in which case the "Error" output signal will be 0. In case the content of some memory cell is in error, the comparator inputs will not be identical and at least one of the input signals to each of N1 and N2 will be a logic 1. This forces the outputs of both NOR gates to a logic 0 causing the "Error" output signal to be a logic 1 indicating a fault. With their large number of inputs, both the N1 and the N2 gates are best implemented as distributed CMOS domino logic gates. The pull down transistors of both gates span the width of the array *q* columns as shown in Figure 4. The N3 gate, however, is implemented as a static CMOS NOR gate. The output signals of N1 and N2 (α and β) together with a single switched ground signal (γ) are bused in a direction normal to the bit lines. The outputs of N1 and N2 (α and β), are evaluated during the comparator sampling clock Φ only after the contents of memory cells of the sector under consideration are sensed (i.e., after the sense enable signal SE is asserted). In the normal operating

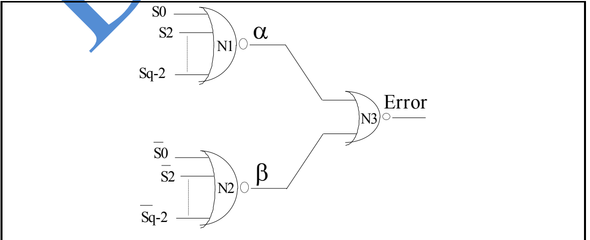


Figure 3. Parallel Coincidence Comparator Logic Circuit (Even Comparator)

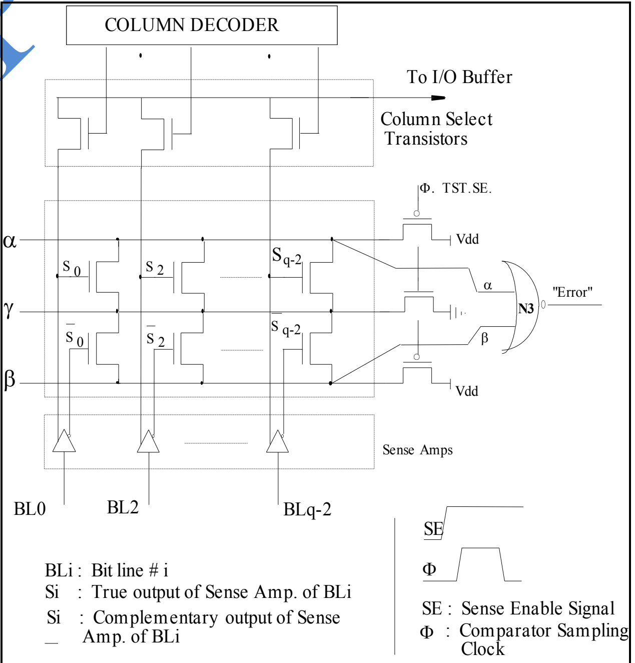


Figure 4. Coincidence Comaparator for Even Bit Lines.

mode, the "Error" signal always assumes a logic 0 state. Furthermore, the gate capacitance load on the sense amplifier outputs is not only balanced but is also minimal since the NMOS load transistors of the domino logic NORs are OFF in the normal mode resulting in negligible overhead on the sensing speed. In a preliminary design of a 256K DRAM using 1.5μ CMOS technology, SPICE simulations indicated an estimated delay overhead of less than 3ns under worst case operating conditions. The delay is mostly due to the added logic at the address buffer output to control the column decoder input signals. The total BIST logic area overhead was less than 2%.

4.0 ARRAY FAULT MODELS and TEST ALGORITHMS

4.1 Dual-Port Pattern Sensitive Fault Model Static Neighborhood Pattern Sensitive Faults (SNPSFs) and Dynamic Neighborhood Pattern Sensitive Faults (DNPSFs) [24] can occur either in single or DPRAMs. However, the presence of two ports and the ability to have two simultaneous write operations in DPRAMs provoke another type of neighborhood pattern sensitive faults. Interference faults between cells, e.g. coupling faults or static and dynamic pattern sensitive faults, are generally attributed to leakage currents and capacitive coupling between cells that are physically adjacent [1, 25, 27]. These two problems become more pronounced as the packing density of memory cells is increased and their geometries are reduced. DNPSFs may be caused by capacitive coupling or transition-induced leakage currents from neighboring cells. A more subtle type of faults, however, would cause degradation in the integrity of the stored data in the base cell due to a single transition in a deleted neighborhood cell but such degradation is not strong enough to show as a DNPSF. Two simultaneous such transitions, however, would have a stronger effect on the base cell resulting in a larger degradation which would be more likely to show as a fault. Such type of duplex dynamic pattern sensitive faults, not possible with SPRAMs, may occur in DPRAMs due to their simultaneous double write capability.

Definition 1: A ***Duplex Dynamic Neighborhood Pattern Sensitive Fault (DDNPSF)*** for DPRAMs is defined as follows: the content of a cell is forced to a certain state due to a duplex change in its deleted neighborhood pattern. This change consists of two simultaneous transitions in two cells of the deleted neighborhood, while the remaining cells of the deleted neighborhood and the base cell contain a certain pattern.

It should be noted that the double simultaneous transitions of the two deleted neighborhood cells can be either in the same direction or in opposite directions. If the transitions are in the same direction, then they can be either ***positive*** (where the two cells change states from 00 to 11), or ***negative*** (where the two cells change states from 11 to 00). If the transitions are in opposite directions (*mixed*), then one cell changes state from 0 to 1 while the other cell changes state from 1 to 0. Therefore, DDNPSF can be further classified as either ***positive***, ***negative,*** or ***mixed***. As stated earlier, while the effect of a single transition in one cell of the deleted neighborhood may not be strong enough to show as a fault in the base cell, the effect of double simultaneous transitions is stronger and is more likely to show as a DDNPSF fault if the effects of both transitions are additive. Such additive effect will occur when the double transitions are in the same direction. It is also reasonable to assume that double transitions in the same direction will sensitize most faults caused by single transitions in any of the two deleted neighborhood cells undergoing the transition. In other words, most DNPSF are detectable by DDNPSF tests. Therefore, we will limit our discussion to positive and negative DDNPSFs. To test for positive (negative) DDNPSF, each base cell must be read in state 0 and in state 1, for all possible positive (negative) duplex changes in the deleted neighborhood patterns. To minimize the total number of ***double write*** operations required to step through all such changes in the deleted neighborhood patterns, we extend the notion of Eulerian sequence [2, 15] to handle DPRAMs as described below.

4.2 Extended Eulerian Sequence Let $X = (x_0, x_1, \ldots, x_{k-1})$, $Y = (y_0, y_1, \ldots, y_{k-1})$ be vertices in a k-dimensional cube that represents the state vector space of a deleted neighborhood of size k. Vertex X is said to be $\geq$ vertex Y if and only if $x_i \geq y_i \;\; \forall \;\; i = 0,1, \ldots, k-1$. Conversely, X is said to be $\leq$ Y if and only if $x_i \leq y_i \; \forall \; i = 0,1, \ldots, k-1$. Otherwise, if for some i , j $x_i > y_i$ and $x_j < y_j$ then X and Y are unordered. We define a DDNPSF graph for a k-bit deleted neighborhood as a graph where there is a vertex for each k-bit pattern. Two vertices X and Y are connected if and only if they satisfy the following two conditions:

1. The Hamming distance between X and Y is exactly 2, i.e., HD(X,Y) = 2
2. X $\geq$ Y, or X $\leq$ Y

When two vertices, X and Y are connected, they are connected by exactly two directed arcs: one from X to Y, and the other from Y to X. Condition 1 ensures that, moving from vertex X to vertex Y corresponds to a double transition write. Condition 2 ensures that the double transition write is either positive (X $\leq$ Y) or negative (X $\geq$ Y). Traversing the set of all directed arcs in the DDNPSF graph sensitizes all of the positive and negative DDNPSF faults for a deleted neighborhood of size k. Thus, a tour through this graph such that each directed arc is traversed at least once constitutes a test sequence for DDNPSFs. The length of such test sequence is minimum if each directed arc is traversed only once.

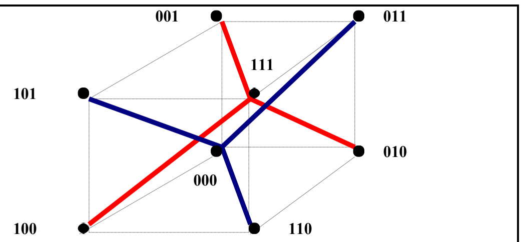


Figure 5. DDNPSF Graph for K = 3 (heavy lines)
DNPSF Eulerian Graph (dotted lines)

Figure 5 shows the DDNPSF graph for a hypothetical neighborhood of size k = 3. The arcs are shown in heavy lines. Each line represents two directed arcs (one for each direction). The dotted lines represent the arcs of a single-port DNPSF Eulerian graph. The arcs of the DDNPSF graph can be easily identified if the vertices of the k-dimensional cube are arranged in levels such that a vertex X appears in level i, i ∈ {0,1,...,k}, if and only if vertex X represents a pattern which has exactly i ones. This is illustrated in Figure 6 for the hypothetical case of k = 3 where arcs emanating from a vertex on level 0 (level 3) can only connect to a vertex on level 2 (level 1) and vice versa. It is clear from Figure 6 that the DDNPSF graph is not connected, but rather consists of two strongly connected subgraphs. Each of the two subgraphs is an Eulerian graph with a possible subtour which traverses each directed arc of the subgraph exactly once [26]. An *extended Eulerian sequence* (EES) for k = 3 is constructed by linking the two Eulerian subtours as shown in Table 1. Each of subtour 1, and subtour 2 has a length of 6. In addition, a link of two double write operations is needed to move from the end of subtour 1 (000) to the beginning of subtour 2 (111). Thus, such a test sequence has a length of 14.

4.2.1 Extended Eulerian Sequence for k = 4: Figure 7 shows the two subgraphs (subgraph 1 and subgraph 2) for the DDNPSF graph for k = 4. A vertex X appears on level i if and only if X has exactly i ones. The following should be noted:

Table 1

SUBTOUR 1	SUBTOUR 2
000	111
101	001
000	111
011	100
000	111
110	010
000 Link	111

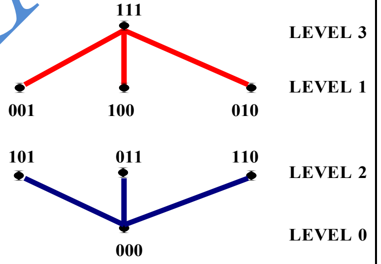
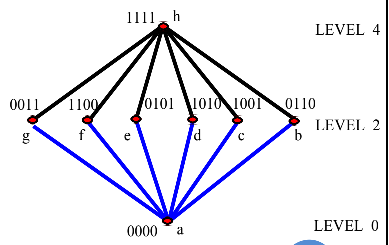


Figure 6.DDNPSF Graph for k = 3

1. No arcs connect vertices X and Y on the same level, e.g. vertices on level 2, even if HD(X,Y) = 2 since X and Y are unordered.

Table 2. An Optimal Subtour 1 (k= 4)

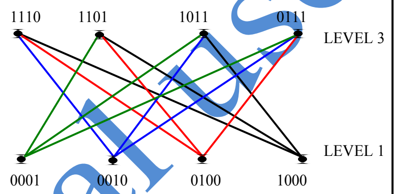
0000		
0110	1010	1100
1111	1111	1111
0110	1010	1100
0000	0000	0000
1001	0101	0011
1111	1111	1111
1001	0101	0011
0000	0000	0000



(a). Subgraph 1 (vertices on levels 0, 2, 4) (k= 4)

Table 3. An Optimal Subtour 2 (k= 4)

1000	0100	0100
1011	0111	1101
0001	0010	1000
1011	1011	1110
0010	1000	0010
0111	1101	1110
0001	0001	0100
0111	1101	1110
		1000



(b). Subgraph 2 (vertices on levels 1, 3) (k = 4)

Figure 7 DDNPSF Graph for k = 4

2. No arcs connect vertices on levels i, and i ± 1 ∀ i, j ∈ {0,1, ..., k} since their hamming distance is not equal to two.
3. If X is a vertex on level 4 and Y is a vertex on level 2, then HD(X,Y) = 2 and X ≥ Y. A similar remark applies if X is a vertex on level 2, and Y is a vertex on level 0. Arcs connecting vertices X and Y constitute the arcs of subtour 1 in Figure 7(a).
4. If X is a vertex on level 3 and Y is a vertex on level 1, an arc of subtour 2 in Figure 7(b) connects vertices X and Y iff X ≥ Y and HD(X,Y) = 2.

Subtour1 is an optimal Eulerian subtour of length 24 and is obtained by traversing each path between 0000 and 1111 up and down, i.e., in both directions. (6 paths × 2 arcs × 2 directions = 24). One such subtour is shown in Table 2. An optimal Eulerian subtour 2 of length 24 for the second subgraph of Figure 7(b) is shown in Table 3. Therefore, an optimal extended Eulerian sequence for k = 4 will have a length of 49. It consists of subtour 1 (length 24), a link (length 1) from the end of subtour 1 to the beginning of subtour 2, and subtour 2 (length 24). It should be noted that the EES covers SNPSF faults as well since all of the graph vertices are visited.

4.2.2 Extended Eulerian Sequence for k > 4 In general, for any deleted neighborhood of size k, the DDNPSF graph is a disconnected graph with two strongly connected components (subgraphs). Graph vertices having even levels belong to one subgraph, while vertices of odd levels belong to the other subgraph. Each of these two subgraphs is Eulerian. An EES to test for DDNPSFs is constructed by linking the two Eulerian subtours of these two subgraphs. The EES will cover both DDNPSF and SNPSF faults.

4.3 Memory Array Tiling: The adopted memory array tiling [21] is shown in Figure 8. The memory cells are partitioned into 8 distinct sets of memory cells. The 8 sets

ROW \ COL	0	1	2	3	4	5	6	7
0	s	N	n	S	s	N	n	S
1	W	w	E	e	W	w	E	e
2	n	S	s	N	n	S	s	N
3	E	e	W	w	E	e	W	w
4	s	N	n	S	s	N	n	S
5	W	w	E	e	W	w	E	e
6	n	S	s	N	n	S	s	N
7	E	e	W	w	E	e	W	w

Figure 8. Memory Tiling

are designated by the letters *N,E,W,S*, *n,e,w,s*. Each row has cells belonging to four of these sets. Cells on even rows belong to the *s*, ***n***, *S*, *N* sets, while cells on odd rows belong to the *w,e,W,E* sets. With this tiling, the eight sets of memory cells are divided into two major groups: The first group consists of all cells designated by *n,e,w,s* (half the array cells), while the second group consists of all cells designated by *N,E,W,S*. Considering *n,e,w,s* as base cells, *N,E,W,S* become the deleted neighborhood for these cells and vice versa. This tiling has the advantage of simple address generation which is an important requirement for built-in testing.

4.4 Array Test Procedure In the array test mode, cells belonging to the same set, on any given row, are accessed in parallel as one ***sector***. Each sector consists of *q/4* memory cells. For example, all *q/4* cells designated ***s*** on row 0 constitute one *sector*. This includes cells at columns 0,4,8,12, etc. In general, the four sectors of a row will be designated by the letters corresponding to cell sets on this row, i.e. {*s,N,n,S*} for even rows and {*W,w,E,e*} for odd rows. Write operations, in the array test mode, are sector oriented rather than individual cell oriented where each port performs write operations into the *q*/4 memory cells of a given sector in parallel through multiple selection of appropriate column decoder outputs as explained in section 3. Thus, a ***double write*** operation refers to the simultaneous writing of data into the two array sectors addressed by the two ports. Furthermore, in this mode, the contents of the *n,e,w,s* (or *N,E,W,S*) base cells are repeatedly verified for the correct background data (all 0's or all 1's) using the two coincidence comparators after each change in the deleted neighborhood pattern. This step is accomplished in *r/2* verification read operations. Odd-numbered cells, e.g. *w,e* (or *N, S*), are verified in parallel by one coincidence comparator, while simultaneously the even-numbered cells, e.g. *n, s* (or *W, E*) are verified in parallel by the other coincidence comparator.

To test for positive (negative) DDNPSF, each base cell (e.g., *n,e,w,s*) must be read in state 0 and in state 1, for all possible positive (negative) duplex changes in the deleted neighborhood patterns (e.g., *N,E,W,S*). To minimize the total number of double write operations required to step through all such changes in the neighborhood patterns, the order specified by an extended Eulerian sequence (EES) of k=4 is followed. This should be performed once with (*n,e,w,s*) considered as base cells and another with (*N,E,W,S*) as the base cells.

The memory array is first initialized to a background of all 0's (vertex **a** in subgraph 1 of Figure 7(a)). Considering *n,e,w,s* as base cells, the arcs of subtours 1 and 2 are traversed and positive/negative duplex changes are applied to the deleted neighborhood (*N,E,W,S*) throughout the whole array. Traversing the arcs of the EES is repeated for an all 1's background data in the *n,e,w,s* base cells. Finally, the whole process is repeated with the role of *N,E,W,S* and *n,e,w,s* interchanged (i.e., by considering *N,E,W,S* as base cells and *n,e,w,s* as the deleted neighborhood). The test procedure can therefore be summarized as follows:

1. Consider *n,e,w,s* as base cells, and *N,E,W,S* as the deleted neighborhood cells
2. Initialize the memory array to a background of all 0's.
3. Traverse the next arc of the EES (k = 4) and verify the base cells.
4. If the EES is not completed, go back to 3.
5. Initialize all base cells to 1's, and all deleted neighborhood cells to 0's.
6. Repeat steps 3 through 4.
7. Consider *N,E,W,S* as base cells, and *n,e,w,s* as the deleted neighborhood cells and repeat steps 2 through 6.

4.4.1 EES Arc Traversal It should be noted that, while following an optimum EES minimizes the number of write operations, it does not test all the base cells for the required duplex transitions. As an example, consider the case where *n,e,w,s* are the base cells and the deleted neighborhood cells (*N,E,W,S*) undergoing the positive duplex transition (0000 $\rightarrow$ 0011), i.e. traversing arc **ag** in subgraph 1 of the DDNPSF graph (Figure 7-a). A series of (00 to 11) double write operations to sectors *S* and *W* on rows (0, 1), (2, 3), (4, 5), ... , (*r*-2, *r*-1) will test base cells *e* and *s* for the ***positive*** duplex transition 0000 to 0011, while base cells *n* and *w* are only tested for two successive ***single*** transitions from 0000 to 0010 then to 0011. To test all base cells for all possible duplex transitions, an ***arc traversal procedure*** is followed. In this procedure, a given bi-directional arc is traversed in both directions (e.g., 00 $\rightarrow$ 11 $\rightarrow$ 00) in a repeated manner such that all base cells are covered. The procedure also includes verification cycles in between global write operations. As an example, consider the case of traversing the bi-directional arc **ag** (0000 $\leftrightarrow$ 0011) in subgraph 1 (Figure 7-a). A series of double write operations to all cells designated *W* and *S* in the memory array need to be performed. For all the *n,e,w,s* base cells to be exposed to the required deleted neighborhood **duplex** transitions (0000 to 0011 and 0011 to 0000), such series of writes is performed in ***two phases*** with ***two passes*** per phase. Depending on the row addresses of the *W* and *S* sectors selected to be written simultaneously, either cells *e* and *s* (phase 1) or cells *n* and *w* (phase 2) will be subjected to the required positive and negative duplex transitions (0000 ↔ 0011) in their deleted neighborhoods. In the *first pass* of the *first phase*, a sequence of double write operations (0000 → 0011) to sectors *S* and *W* on rows (0, 1), (2, 3), (4, 5), ... , (*r*-2, *r*-1) is performed in *r*/2 double write cycles (see Figure 8). This pass will test base cells *e* and *s* for the ***positive duplex*** transition 0000 to 0011, while base cells *n* and *w* are only tested for two successive ***single*** transitions from 0000 to 0010 then to 0011. This is followed by a sequence of verification read cycles (*r*/2 cycles) to verify the data in the base cells. In the *second pass*, a sequence of (0011 to 0000) double write operations is applied to the same set of *W* and *S* sectors (*r*/2 cycles). This will test base cells *e* and *s* for the ***negative*** duplex transition 0011 to 0000, while base cells *n* and *w* are only tested for two successive ***single*** transitions from 0011 to 0001 then to 0000. This is also followed by another sequence of verification read cycles (*r*/2 cycles). Thus, in the first phase, base cells *e* and *s* are tested for the positive and negative duplex transition (0000 ↔ 0011) in their deleted neighborhood. This is accomplished in *r* double write cycles and *r* verification read cycles for a total of 2*r* cycles. The *second phase* is similar to the *first* with the exception that instead of cells *e* and *s*, base cells *n* and *w* are tested for the positive and negative duplex transitions (0000 ↔ 0011) in their deleted neighborhoods. This is accomplished by a different grouping of the two sectors being written simultaneously (*W* and *S*). Thus, in the second phase the double write operations will be performed on sectors *W* and *S* on rows (1, 2), (3, 4), (5, 6), ... (*r*-3, *r*-2), and (*r*-1, 0).

Traversing arcs which require duplex transitions in the (*S*, *E*), the (*E*, *N*), or the (*N*, *W*) sectors, will be handled similarly and each would require 2*r* double write cycles and 2*r* verification read cycles for a total of 4*r* cycles. In contrast, traversing arcs which require duplex transitions in the (*S*, *N*), or the (*E*, *W*) sectors, is performed in *three phases* and requires 3*r* double write cycles and 3*r* verification read cycles for a total of 6*r* cycles [22]. This is due to the fact that sectors of these groups fall on the same row and column which requires slightly different handling. Thus, for duplex transitions applied to the *N, S* (or *E, W*) deleted neighborhood cells, i.e. traversing the arc 0XX0 ↔ 1XX1 (X00X ↔ X11X), ***one phase*** is required to sensitize DDNPSF faults for the *n, s* (*e, w*) base cells, ***a second phase*** for the *w* (*n*) base cells and a ***third phase*** for the *e* (*s*) base cells. Similar argument holds when using *N,E,W,S* as base cells and *n,e,w,s* as deleted neighborhood cells. Thus, the ***arc traversal procedure*** performs all the required positive and negative duplex transitions to the deleted neighborhoods of the base cells throughout the whole array as well as the necessary verification reads of the base cells.

A finite state machine (FSM) is designed to step through the EES and determine the next arc traversal operation that is to be carried out based on the current content of the deleted neighborhood. The complexity of such FSM must be reduced for built-in testing. For a minor increase in the length of the test sequence, the complexity of such FSM can be reduced. To achieve this, the optimal Eulerian subtour 2 shown in Figure 7(b) is replaced with subtour 2* shown in Figure 9 and Table 4. Subtour 2* starts at vertex A by traversing the three bi-directional arcs emanating from A, i.e., AH, AG, and AF. Moving to vertex B (using the added link shown in Figure 9 as a dotted line AB) the bi-directional arcs emanating from B, i.e., BH, BG, and BE are traversed. Then move to C and so on. The extra links added to subtour 2* (AB, BC, and CD) would slightly increase the test sequence length but will lead to a simpler implementation of the FSM as explained below.

To point at the target cells, x, y ∈ {*N,E,W,S*} for the next double write operation, a 4-bit register, *NEWS_REG,* is used. The bits of *NEWS_REG* have a one-to-one correspondence with the *N,E,W,S* sectors such that a given bit is 1 if and only if the corresponding sector is to undergo a transition write operation. For example, *NEWS_REG*=1001 specifies transition double write operations into sectors *N,S.* As illustrated by the following example, the contents of *NEWS_REG* are simply determined using two additional 4-bit registers, R1 and R2. The added cost for this simplicity is the slight increase in the length of subtour 2* as compared to subtour 2.

Table 4 Subtour 2* Traversed Arcs

AH	BE	CF	DG
AG	BH	CE	DF
AF	BG	CH	DE
Link	Link	Link	

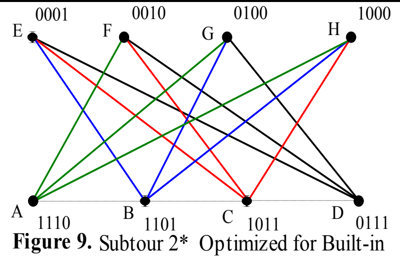


Figure 9. Subtour 2* Optimized for Built-in Test (k = 4)

Example: At vertex A (Figure 9) the pattern of the deleted neighborhood $(N,E,W,S) = (1110)$. To reach vertex H, (N,E,W,S) should be (1000). This requires negative double transition writes to the sectors designated by E,W (i.e., *NEWS_REG* should be 0110). This is achieved by the following simple procedure.

- The deleted neighborhood pattern corresponding to the starting point (A) is complemented, i.e. 0001, and loaded into both of R1 and R2. The pattern in R2 is used as a ***mask***. The following two operations produce the correct required value of *NEWS_REG*:

a) <u>*step 1*</u>: Cyclic Shift Right R1
b) <u>*step 2*</u>: *NEWS_REG* = R1 NOR R2

- At this point *NEWS_REG* contains the right value, 0110, indicating that E,W are the target sectors for the double transition write operations required to traverse the bi-directional arc AH. If steps 1,2 are repeated *NEWS_REG* will contain 1010 providing the locations for the transition writes required for the next bi-directional arc AG. This is true for all the bi-directional arcs emanating from A, and is also true for each of B, C, D provided that the proper value of the starting point is used (i.e., 0010 for B, 0100 for C, and 1000 for D). As shown in Figure 10, after traversing all bi-directional arcs emanating from A, R1 automatically contains the starting point for vertex B (and so on). Moreover, the value of *NEWS_REG* to effect an internal link, e.g. AB, is obtained by simple ORing of the contents of R1 and R2. Figure 11 shows a pseudo code for the partial testing procedure handling subtour 2*.

Starting Point R1=0001, Mask R2 = 0001			
Cyclic Shift Right R1	***R1 NOR R2***	***Target Cells***	***Traversed Arc***
1000	0110	E,W	AH
0100	1010	N,W	AG
0010	1100	N,E	AF
Starting Point R1=0010, Mask R2 = 0010			
0001	1100	N,E	BE
1000	0101	E,S	BH
0100	1001	N,S	BG
Starting Point R1=0100, Mask R2 = 0100			
0010	1001	N,S	CF
0001	1010	N,W	CE
1000	0011	W,S	CH
Starting Point R1=1000, Mask R2 = 1900			
0100	0011	W,S	DG
0010	0101	E,S	DF
0001	0110	E,W	DE

Figure 10 Subtour 2* Bidirectional Arc Traversals

The same hardware can be used in a similar manner to step through subtour 1. Using the *arc traversal* procedure to traverse arcs of subtour 1, however, requires an additional link between vertices **a** (0000) and **h** (1111). The DDNPSF graph for the modified subtour 1, designated subtour 1*, is shown in Figure 12 and the corresponding pseudo code is shown in Figure 13.

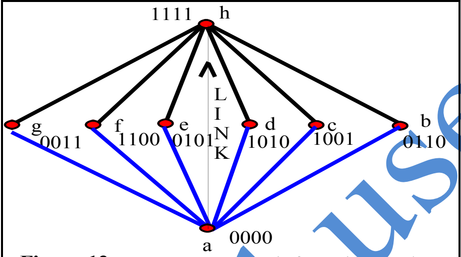


Figure 12. DDNPSF Graph for Subtour 1*

Data written into the two ports in the array test mode are specified using a two-bit register *DATA_REG* (1-bit per port). It is noted that data written to both ports while traversing the bi-directional arcs within subtour 1* and subtour 2* are identical for both ports. This is because the double transitions effected are either both positive or both negative. However, different data inputs to the two ports are required for the following uni-directional links:

i. From the end of subtour 1*, where $(N,E,W,S) = 1111$, to the beginning of subtour 2*, where $(N,E,W,S) = 1110$.
ii. The three internal links of subtour 2* (A to B, B to C, and C to D in Figure 9).

```
procedure subtour2*
LOAD R1, "0001"       /* R1 ← 0001 */
for i = 1 to 4 do
  LOAD R2, R1         /* R2 ← R1 */
  repeat 3 times
    Cyclic-Shift-Right R1
    NEWS_REG = R1 NOR R2
    (x,y) = f (NEWS_REG)
    /* From NEWS_REG determine x,y∈ {N,E,W,S} of the
     target cells to undergo double transition writes. This is
     designated (x,y) = f (NEWS_REG) */
    Traverse_arc(x,y)
  end repeat
  if i < 4 then/* Link steps */
    NEWS_REG = R1 OR R2
    (x,y) = f (NEWS_REG)
    Double Transition Write x,y
  end if
end for
end subtour2*
```

Figure 11. Pseudo Code for the partial testing procedure handling Subtour 2*

```
procedure subtour1*
for i = 1, 2
  LOAD R1, "0001"
  LOAD R2, R1
  repeat 3 times
    Cyclic-Shift-Right R1
    NEWS_REG = R1 NOR R2
    repeat 2 times
      (x,y) = f (NEWS_REG)
      /* From NEWS_REG determine x,y ∈ {N,E,W,S} of the
      target cells to undergo double transition writes.*/
      Traverse_arc(x,y)
      $NEWS\_REG = \overline{NEWS\_REG}$
    end repeat
  end repeat
if i = 1 then Link_ah /* Write 1111 into deleted
                         neighborhood */
end for
end subtour1*
```

Figure 13. Pseudo Code for the partial testing procedure handling Subtour 1*

In subtour 1*, arcs connected to vertices **b** and **c** require 6*r* read/write cycles, while all other arcs require only 4*r* cycles. Thus, including the *r* cycles required for the link from vertex **a** to vertex **h**, procedure subtour 1* requires a total of 57*r* cycles. In procedure subtour 2*, traversing the 3 arcs from each of vertices **A**, **B**, **C**, or **D** requires 14*r* cycles (6*r* + 4*r* + 4*r*). In addition, each of the 3 links **A** to **B**, **B** to **C**, and **C** to ***D*** requires *r*/2 double write cycles. Thus, subtour 2* requires a total of 57.5 r cycles.

4.5 Comparator Test: Since the two coincidence comparators are used throughout the array test in addition to the fact that their layout follows the tight array design rules (same bit line pitch), the test procedure starts by testing the operation of these comparators. Using a simple single stuck-at fault model, each comparator is fully tested using a total of "*q* + *2*" patterns. Each pattern is first written into two specific memory sectors in one double write cycle. That pattern is then applied to the inputs of the comparators (Figure 3) using one verification read cycle. Since both comparators are tested simultaneously using the two ports, a total of " *q* + *2* " double write cycles and *q* +*2* verification read cycles are required for the test.

The pseudo code for the complete memory array test procedure is shown in Figure 14. In the overall array test procedure of Figure 14, each initialization step (e.g., step 1 or step 5) is performed in 2*r* cycles, since two sectors will be written by both ports per cycle. Furthermore, the link operation of step 3, even though it can be performed in *r/4* operations using the two ports, it will be performed in *r/2* double write operations to maintain simple BIST logic [22]. Thus, a total of {468*r* + *q* + *2*} read/write cycles, i.e. O($\sqrt{n}$), are required by the array test algorithm {(2*{2*r* + 2*[57*r* + *r*/2 + 57.5*r*] + 2*r*}) + *q* + *2*}.

```
procedure array_test
  0. Comparator Test
  1. Initialize the memory array M to all 0's.
  2. procedure_subtour1*
  3. link to subtour 2*     /* This step is accomplished by
                               writing 0 to all cells labeled S */
  4. procedure_subtour2*
  5. Initialize M such that (N,E,W,S) = 0000 and
                            (n,e,w,s) = 1111
  6. repeat steps 2 through 4
  7. Interchange the role of N,E,W,S and n,e,w,s and
     repeat steps 1 to 6
end array_test
```

Figure 14. Pseudo Code for the Overall Memory Array Test Procedure

5.0 ADDRESS DECODER FAULTS

According to the DPRAM model, each port has its own row decoder and column decoder. Since multiple cells are accessed during the array test algorithm, some decoder faults are not detectable by this test. Therefore, dedicated tests to verify decoder operation must be used. For fault free decoder operation, each address in the address space should access one and only one memory cell which is not accessed by any other address. In addition, each cell should be uniquely accessed by the same address from either port. To reduce the test algorithm complexity, the row and column decoders are independently tested. It is assumed that a decoder fault does not change the decoder circuit into a sequential one [28]. In addition, decoder faults are assumed to affect both the read and the write operations equally.

5.1 Dual-Port Decoder Fault Model

Definition 2: Let $\mathbf{R}_a$ and $\mathbf{R}_b$ be the sets of all possible row lines of ports *a* and *b* respectively. A row line r_x which is accessed from port *a* (*b*), i.e. $r_x \in \mathbf{R}_a$ ($r_x \in \mathbf{R}_b$), is said to *match* another row line r_y *if and only if* r_y is accessed by the other port *b* (*a*), i.e. $r_y \in \mathbf{R}_b$ ($r_y \in \mathbf{R}_a$), *and* both r_x and r_y access the same set of memory cells. If r_x matches r_y we write ($r_x \Leftrightarrow r_y$).

Let A = {0 .. (*r*-*1*)} be the set of all row addresses. Two mapping functions ϕ_a and ϕ_b map this set of row addresses *onto* the set of row lines for each of the two ports (one to one and onto mapping). Thus; for fault free row decoder operation:

$$\phi_a : \quad A \rightarrow \mathbf{R}_a,$$

$$\phi_b : \quad A \rightarrow \mathbf{R}_b, \text{ and}$$

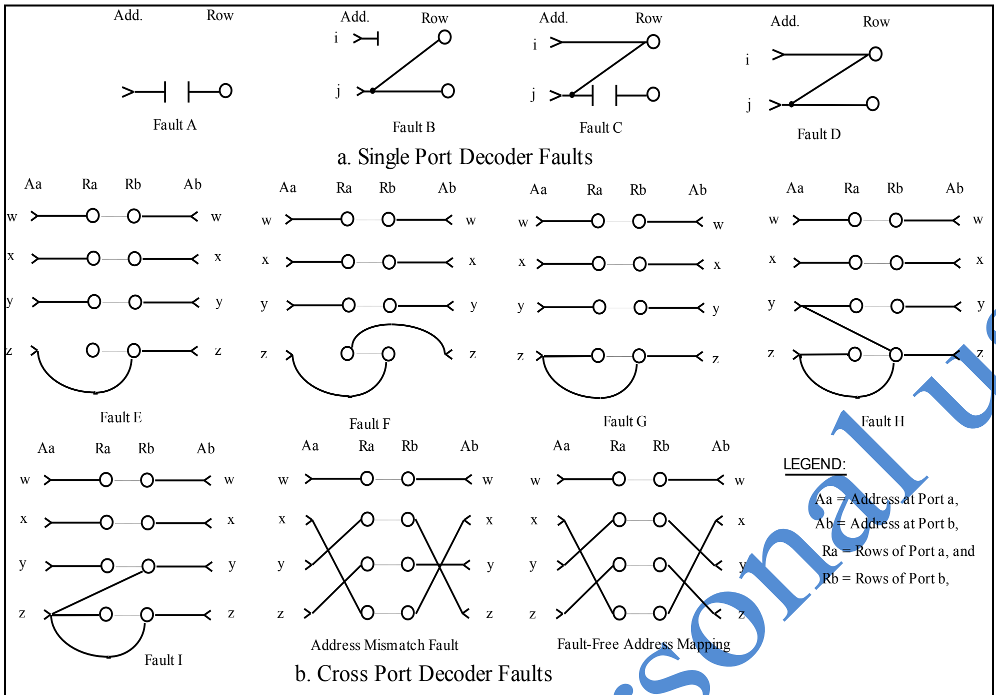


Figure 15. Row Decoder Faults in Dual Port RAMs

$\phi_a(i) \Leftrightarrow \phi_b(i)\ \forall i \in A$,

Thus; $\forall r_x \in \mathbf{R}_a, \exists\ r_y \in \mathbf{R}_b$ such that $r_x \Leftrightarrow r_y\ (\forall x \in A)$.

This implies that each port accesses a total of *r* distinct rows. In addition, one and only one row line is accessible by each valid address, and conversely, each address accesses exactly one row line. Moreover, the above mapping functions guarantee that each row of array memory cells is accessible from both ports by the same unique address whether the accessing port is *a* or *b*. Similar definitions and mapping functions apply for the column address decoder as well.

There are six types of expected DPRAM decoder faults. These can be broadly classified into two main categories: single-port decoder faults (SPDF) and cross port decoder faults (CPDF).

a) Single-port Decoder Faults (SPDF): This category includes four types of faults typically found in single-port RAMs:

1. A row (column) line which is not accessed by any address.
2. A row (column) address that accesses no row (bit) line.
3. One address which accesses more than one row (column) line (one to many).
4. A row (column) line accessed by more than one address (many to one).

b) Cross Port Decoder Faults (CPDF): Faults in this class result from interaction between the access ports. The following two faults belong to this class:

5. An address of one port accesses a row (column) line belonging to the other port.
6. Address mismatch faults, where the same address to both ports selects two row (column) lines that are not matching.

None of the single-port decoder faults can stand alone [15], but rather a combination of such faults will exist together. For cross port decoder faults, only the last fault (address mismatch) can stand alone. Possible fault combinations are shown in Figure 15. Following the same notation of De Goor & Verruijt [15], fault type A combines faults 1 & 2, fault B combines faults 2 & 3, fault C combines faults 1 & 4, and fault type D combines faults 3 & 4. Extending this notation to accommodate the dual-port case, fault combinations E, F, G, H and I are illustrated in Figure 15. Fault type E combines faults 1 & 5, fault F shows a case where fault 5 is combined with itself (5 & 5) from both ports, faults G, H and I show various forms of combining faults 3, 4 & 5.

Fault 5 maps a row address of one port into the row line belonging to the other port. This, however, does not mean that cells on the erroneously selected row are accessible from the first port. This is due to the fact that to access a cell, both its row line and column line should be properly selected by the same accessing port. In case of fault 5, only the row line is erroneously accessed by the first port, while the column line is accessed by the second port. As an example, let row address *i* of port *a* actually selects row line *i* of port *b*. Consider the case where port *a* is reading some cell (*i*,*k*) on row *i* and column *k* (whose content is *x*) while port *b* is writing ($\overline{x}$) data into another cell (*l*,*k*) with a different row address *l* but on the same column address k. Port b will write ($\overline{x}$) not only into cell (l,k) but also into cell (i,k) accessed for reading by port a. Data read by port a will depend on the memory type. In case of DRAMs, the data read will be some stuck-at value. For SRAMs, data read will be noise and layout dependent. A similar argument holds for fault 5 on column decoders.

5.2 Decoder Test Procedures: To reduce the complexity of the test algorithm as well as simplify the BIST logic, row and column decoders are tested separately. The algorithm consists mainly of two test procedures which utilize the capability of simultaneous access by both ports to minimize the test time. To test row decoders, two march tests are performed on the memory cells of particular columns. The first test scans two different columns, each accessed by one port, in two opposite directions, while the second scans a single column, accessed by both ports, in the same direction.

Similar test procedures are used for column decoders with the march tests applied to the memory cells of specific rows rather than columns. The test algorithm is of O(r) for row decoders and O(q) for column decoders, i.e. O($\sqrt{n}$) decoder test algorithms.

5.2.1 Row Decoder Test Procedure 1 Figure 16 shows the pseudo code for this test procedure. The notation *Read_x_α(i,j)* (*Write_x_α(i,j)*) is used to indicate a read (write) operation with expected (input) data x ($x \in \{0,1\}$) through port α ($\alpha \in \{a,b\}$) from (to) the memory cell whose row and column addresses are i and j respectively.

```
Procedure Decoder_1
/* Initialize the memory cells of two distinct columns l and m to all 0's.*/
initialize_columns (l,m,0);
    for i = 0 to r-1 do
        Read_0_a(i,l), Read_0_b(r-i-1, m);
         /* Double Read by both Ports */
        Write_1_a(i,l), Write_1_b(r-i-1, m);
         /* Double Write by both Ports */
    end_for
    for i = 0 to r-1 do
        Read_1_a(r-i-1, l), Read_1_b(i,m);
         /* Double Read by both Ports */
        Write_0_a(r-i-1, l), Write_0_b(i,m);
        /* Double Write by both Ports */
    end_for
end Decoder_1
```

Figure 16. Row-decoder Test Procedure Decoder_1.

The procedure initializes the memory cells of two distinct columns by the two ports to some background data (all 0's or all 1's). After initialization, the memory cells of both columns are scanned simultaneously by both ports in opposite row address directions where the background data are verified and the complement data are written. This amounts to performing two independent march tests of a total length $5r$ on these two columns simultaneously. Thus, the test procedure fully detects fault types A, B, C, and D [15]. This procedure also detects fault types E, F, G, H and I except for the case where fault 5 corresponds to some address x of one port mapping into row *r-x-1 (or column q-x-1)* on the other port. In addition, address mismatch faults are not detected by this procedure. Such faults will be detected by the second test procedure.

5.2.2 Row Decoder Test Procedure 2: In this procedure, the test is performed only on a single column which is accessed by both ports. This column is first initialized to some background data (all 0's). The column cells are then scanned in an ascending order of row addresses, verifying their contents through read operations by both ports. Then, one port is used to write complementary data which is then also verified by a read operation from both ports. The test procedure verifies that data written to some row address by any port is also readable by both ports using the same address. For test regularity, this test procedure uses the double write operation used in the decoder_1 procedure. Thus, in addition to the test column (l), some other dummy column (m) is written into. This regularity leads to a simpler BIST logic implementation. The pseudo code of this procedure is shown in Figure 17. In addition to detecting address mismatch faults, this procedure detects faults of type E, F, G, H and I which escape detection by the previous procedure.

For the row decoder, the first procedure takes 5r cycles while the second procedure takes 6r cycles for a total of 11r cycles. Similarly column decoder test of length 11q

```
Procedure Decoder_2;
/* Initialize the memory cells of the test column "l" to all 0's for test regularity, a dummy column "m" is also initialized to all 0's.*/
        initialize_columns (l,m,0);
        for i = 0 to r-1 do
            Read_0_a(i,l), Read_0_b(i,l);
            /* Double Read Operation by both Ports */
            Write_1_a(i,l), Write_1_b(i,m);
             /* A Write Operation on Port a*/
            Read_1_a(i,l), Read_1_b(i,l);
            /* Double Read Operation by both Ports */
            Write_0_b(i,l), Write_0_a(i,m);
             /* A Write Operation on Port b*/
            Read_0_a(i,l), Read_0_b(i,l);
             /* Double Read Operation by both Ports */
        end_for
end Decoder_2
```

Figure 17. Row-decoder Test Procedure Decoder_2.

will be required for a total of 11(r+q) cycles for both decoders which is of order O($\sqrt{n}$).

Since the array test algorithm requires (468r + q + 1) read and write cycles, the total number of read and write operations required for both the array and decoder tests is (479 r + 12 q + 1). For p = 1 and r = q = $\sqrt{n}$, this amounts to (491$\sqrt{n}$ + 1) operations.

6.0 CONCLUSION

The testability problem of dual port memories has been investigated. A functional model is defined and architectural modifications to enhance the testability of such chips are described. The modifications allow multiple access of memory cells for increased test speed with minimal overhead on both silicon area and device performance. New fault models are proposed and efficient O($\sqrt{n}$) test algorithms are described for both the memory array and the address decoders. In addition to the classical static neighborhood pattern sensitive faults, the array test algorithm covers a new class of pattern sensitive faults, Duplex Dynamic Neighborhood Pattern Sensitive faults (DDNPSF) which accounts for the simultaneous dual access property of the device. The efficiency of the proposed scheme together with the reduced area and performance overhead makes it a viable and promising approach for future DPRAM designs.

ACKNOWLEDGMENTS

Support of King Fahd University of Petroleum & Minerals is greatly acknowledged. Thanks are also due to A. Abdul-Wahab and A. Hamzah for their help, and to S. Ghanta for helpful discussions.

REFERENCES

1. A. Tuszynski, "Memory Testing," *in "Advances in CAD for VLSI,", series editor T. Ohtsuki*, Vol. 5 on "VLSI Testing" (edited by T. W. Williams), Elsvier Science (North Holland), 1986, pp. 161-228.
2. P. Mazumder, and J. K. Patel, "Parallel Testing for Pattern-Sensitive Faults in Semiconductor Random-Access Memories", *IEEE Trans. Computers,* Vol. 38, No. 3, March 1989, pp.394-407.
12. H. Shinohara, et. al., " Flexible Multiport RAM Compiler for Data Path," *IEEE J. of solid state circuits*, vol. 26, no. 3, Mar 1991
13. J. V. Sas, et. al. "Testability Strategy and Test Pattern Generation for Register Files and Customized Memories," *Microprocessors and Microsystems*, vol. 14., No. 7, pp. 444-456, Sept. 1990.
15. A. J. Van De Goor, "Testing Semiconductor Memories, Theory and Practice," John Wiley, 1991.
16. M. J. Raposa, "Dual-port Static RAM Testing," *IEEE International Test Conf.*, 1988 , pp. 362-368.
17. B. Nadeau-Dostie, A. Silburt and V. K. Agrawal, "Serial Interfacing for Embedded-Memory Testing," *IEEE D & T of Computers*, Apr 90, pp. 52-64.
18. V. C. Alves and M. Nicolaidis "Detecting Complex Coupling Faults in Multi-Port RAMs," *IMAG Research Report No RR978*, Feb. 1991.
19. V. C. Alves, M. Nicolaidis,, P. Lestrat, and B. Courtois, "Built-In Self-Test for Multi-Port RAMs," *IEEE* ICCAD-91, November 1991, pp.248-251.
20. K. T. Le and Kewal K. Saluja, "A Novel Approach for Testing Memories Using a Built-In Self-Testing Technique," *IEEE* ITC-86, pp. 830-839.
21. K. Kinoshita, and K. A. Saluja, "Built-In Testing of Memory Using an On-Chip Compact Testing Scheme," *IEEE Trans. on Computers*, Vol. C-35, No. 10, October 1986, pp. 862-870.
22. A. Amin, M. Y. Osman, R. E. Abdel-Aal, and H. Al-Muhtaseb "An O($\sqrt{n}$) BIST Algorithm for Detection of Duplex Dynamic Pattern Sensitive Faults in Dual Port Memories," KFUPM CCSE Tech Report 015, July, 1993.
24. K. K. Saluja, and K. Kinoshita, "Test Pattern Generation for API Faults in RAM," *IEEE Trans. on Computers, Vol. C-34*, No. 3, March 1985, pp.284-287.
25. Y. You and J. P. Hayes, "A self-testing dynamic RAM chip," *IEEE Journal of Solid State Circuits,* Feb.1985, pp. 428 - 435.
26. A. Gibbons, "Algorithmic Graph Theory," Cambridge Univ Press, U. K. 1989.
27. H. D. Oberleet. al., "Enahnced Fault Modeling for DRAM Test and Analysis," *1991 VLSI Test Symp.,* April 1991, pp. 149-154.
28. S. M. Thatte and J. A. Abraham, "Testing of Semiconductor Random Memory," *Proc. 7th Annual Int'l Conf. on Fault Tolerant Computing*, 1977, pp. 81-87.