register (MAR) and the memory data register (MDR) form the bottleneck in speeding up the test algorithms. In this paper, a new RAM organization with parallel testing technique has been proposed, and two $O(\sqrt{n})$ algorithms have been designed to detect the pattern-sensitive faults. In the test mode, the decoder makes multiple selection of bit lines allowing one to write the same data simultaneously at many storage locations on the same word line. In the read mode, a multibit comparator concurrently detects the error, if any. The additional hardware is minimal and has been designed such that it can fit within the limits of 3λ to 6λ intercell pitch width, where λ is the technological feature width.

The technique of parallel testing was at first proposed by You and Hayes [8], by reconfiguring the memory cells into an n-bit shift register and using a built-in test generator to test many bit lines concurrently. This test procedure has $O(\sqrt{n})$ complexity. It detects limited types of pattern-sensitive faults where a write operation becomes faulty in the presence of a few specific patterns in its adjacent cells. It does not detect the faults caused by transitions in the neighborhood. Moreover, it relies on comparing only two cells in memory arrays having two partitions. Thus, if two cells are identically faulty, the test fails to detect the faults. In contrast to their technique, a multibit comparator which concurrently detects simultaneous 0's and 1's has been used in this paper. You and Hayes reconfigured the memory subarray of size s bit into an s-bit cyclic shift register, where the data recirculated whenever a read operation was performed. The reconfiguration was accomplished by introducing pass transistors on the bit lines which deteriorated both the sensitivity of the sense amplifiers by V_T (threshold voltage in MOS devices) and the access time of RAM in normal mode of operation.

Kinoshita and Saluja [9] analyzed the overhead involved in microcoded, ROM-based built-in test generator for testing the pattern-sensitive faults, using test algorithms of linear time complexity. In a subsequent paper, Le and Saluja [10] utilized the concept of parallel access by modifying the bit lines, and improved the complexity of their algorithms. A potential problem in their technique was the reliability of parallel read operation, particularly in multimegabit DRAM where hundreds of cells are accessed simultaneously for reading and writing.

Sridhar [11] proposed an alternate scheme that used the parallel signature analyzer (PSA) to access many bit lines simultaneously. The PSA operates in three distinct modes: in the scan mode, it is loaded with a specific pattern from outside the chip by sequentially scanning in the bits; in the write mode, it writes its stored value to many bit lines in parallel; finally, in the signature mode, it reads the content of the memory cells written earlier, and generates an error-quotient bit if there is any error. Even though signature analyzers are frequently used for testing, the scheme using PSA has intrinsic problems in memory testing. At first, the scan mode sometimes reduces the parallelism in testing. Han and Malek [12] demonstrated that by using PSA, the popular marching test algorithms [13] could be speeded up at most by 1.2 times. Second, the PSA introduces an aliasing error in testing. An error in output of the jth stage flip-flop at time t_i followed by an error in output of the $j + h$th stage at time t_{i+h} has no effect on the signature, if there is no feedback tap in the PSA between the jth and $j + h$th stage flip-flops. Finally, the overhead in employing a PSA can be fairly high. A b-bit PSA can be implemented by approximately $15b$ transistors. As opposed to this, the proposed technique uses only $2b + 2 \log b + 12$ extra transistors that can be easily laid out even within the constraints of 3λ intercell pitch width in the high-density DRAM.

II. RAM Organization and Operation

Generally, an $n \times 1$-bit RAM, denoted by M_n, consists of p identical two-dimensional submatrices, each of size $b \times w$ ($= r$, say) such that $n = pbw$. A submatrix, M_r, consists of b bit lines and w word lines. Since in two-layer VLSI technology, either the bit line or the word line consists of diffusion/polysilicon wire which has quadratic signal propagation delay as opposed to the linear delay of the metal wire, it is mandatory to partition the M_n into p submatrices so that the access time does not deteriorate for large values of n. Also in order to drive long bit lines, larger sense amplifiers are required to maintain their sensitivity. The partitioning has the intrinsic advantage, since all the submatrices can be tested concurrently, and thereby a testing speedup of p is easily achieved by incorporating built-in testing circuits, or by modifying the addressing scheme in the test mode. But p partitions are achieved at the expense of proliferating the sense amplifiers by a factor of $\sqrt{p}$, and also physically redistributing the decoder logic that introduces additional routing complexity and delay. For practical large DRAM's of size 64K bit or more, the value of p is usually chosen between 2 and 16. The ratio $e = b/w$ is called the *eccentricity of* M_r and is chosen such that the access time is minimized. Usually, $e \geq 1$ if the word line is made of metal, and $e \leq 1$ if the bit line is made of metal.

Let $B = \{0, 1, \cdots, b - 1\}$ denote the set of bit lines in M_r and $W = \{0, 1, \cdots, w - 1\}$ denote the set of word lines. The ordered pair (i, j) described by the Cartesian product $B \times W$ denotes the address of the storage cell that occurs at the crosspoint of the ith bit line and jth word line. Such a cell is denoted by C_{ij}. The state of the cell C_{ij} is denoted by s_{ij}. An operation ψ on cell C_{ij} is denoted by $\psi(C_{ij})$ and its new state by s_{ij}^{+}. Valid operations are to write a 1, or a 0 on a cell, and to read the content of the cell. These operations are denoted by W_1, W_0, and R, respectively. In addition to the above notations, the operation of writing a transition (i.e., complement of the present state) will be denoted by W_x, and the operation of writing $y \in \{0, 1\}$ will be denoted by W_y. Depending on the previous state of the cell $s_{ij} \in \{0, 1\}$, the effect of the application of $\psi \in \{W_1, W_0, R\}$ on C_{ij} can be further classified by the ordered pair in the Cartesian product $s_{ij} \times \psi$.

1) Ordered pair $(0, W_1)$ is to write 1 erasing a previous 0. It is called *transition* write 1, and is denoted by $\uparrow$.

2) Ordered pair $(0, W_0)$ is to write 0 over a previous 0, and is called *nontransition* write 0.

3) Ordered pair $(1, W_1)$ is to write 1 over a previous 1, and is called *nontransition* write 1.

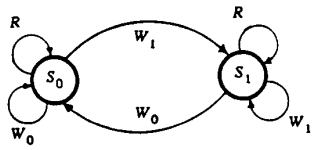

Fig. 1. State diagram of a memory cell and its operations.

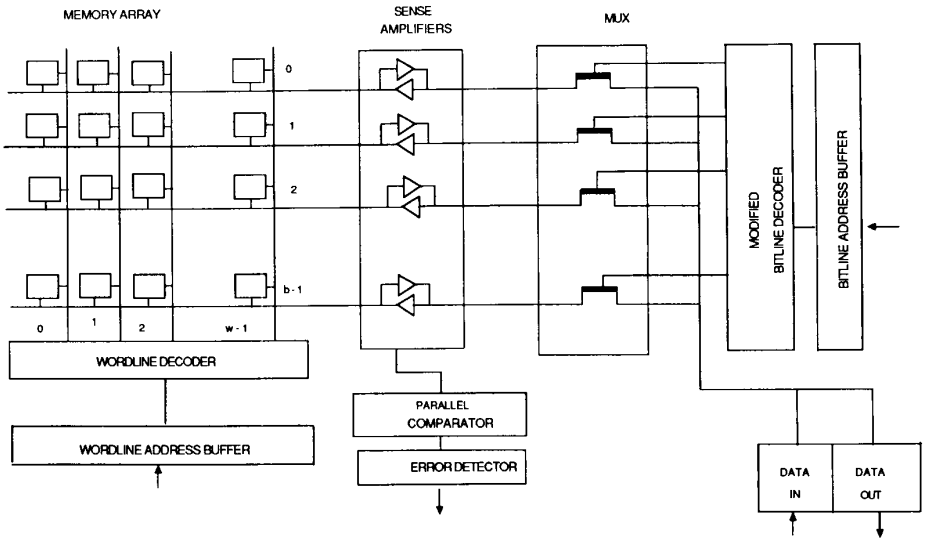

Fig. 2. Organization of a testable RAM.

4) Ordered pair (1, W_0) is to write 0 over a previous 1. It is called *transition* write 0, and is denoted by ↓.

5) Ordered pairs (0, R) and (1, R) denote the operations of reading a cell whose previous contents are 0 and 1, respectively.

The operations W_0, W_1, and R can be represented by the arcs in the state diagram shown in Fig. 1. Since a DRAM cell stores a 0 or a 1, the state diagram consists of two states, S_0 and S_1, corresponding to the storage value of 0 and 1, respectively. In a fault-free DRAM, the action of a read operation, denoted by R, is to output the content of the cell, but the cell stays in its original state, as shown by the self-loops marked R in Fig. 1. If the cell is in state S_0 and a 0 is written by the operation W_0, it stays in the same state. Similarly, if the cell is in state S_1 and a 1 is written by the operation W_1, it does not change the state. Hence, these two writing operations are called the *nontransition* writes. As opposed to these writes, an operation W_1 will result in a change of state if the memory cell is at S_0 before the write operation. Similarly, an operation W_0 will result in a change of state if the memory cell is at S_1 before the write operation. These write operations are called the *transition writes*. In this paper, it will be assumed that only the transition writes and read operations may be faulty. The nontransition writes generally do not result in fault and will not be tested by the proposed test algorithms. In memories where nontransition writes can be faulty, they can be easily tested by following up every transition write in the test procedure by an additional nontransition write. In memories with *nondestructive* read, the operations (0, R) and (1, R) are such that $s_{ij}^{+} = s_{ij}$. But in memories with *destructive* read, s_{ij}^{+} may be different from s_{ij}. The read operations in this paper will be assumed to be *destructive* like in a single-cell DRAM and they will be tested.

III. Design for Parallel Testing of Ram's

The organization of the testable RAM with augmented hardware is shown in Fig. 2. The memory is organized as a $b \times w$ matrix, where b is the number of bit lines and w is the number of word lines. The normal 1-out-of-b, bit-line decoder is modified to select multiple bit lines during testing. In general, b bit lines can be divided into g groups such that the bit line i belongs to group j, where $j = i \pmod g$. In the test mode, all the bit lines in a group j are selected in parallel to execute a read or a write operation. Thus, a write operation in test mode results in writing the content of the data-in buffer on all cells at the crosspoints of the selected word line and the selected bit lines in group j. In read mode, the content of the cells located at the crosspoints of the selected word line, and the selected bit lines group (say j), are read in parallel. Thus, if all the multiple-accessed cells contain 0 (1), a 0 (1) is read out. If all the contents of multiple-accessed cells are not identical, then whether a 0 or 1 is read out into the data-out buffer cannot be predetermined. It may be noted that it is not entirely correct to assume that the resulting operation will be strictly a wired-OR or a wired-AND. The outcome depends on the number of 0's and 1's in the multiple-accessed cells. Thus, for example, if all the cells except one contain 1 and only one cell contains 0, then most likely a 1 will be read out of the data-out buffer. However, if the reverse situation existed, i.e., one 1 and the rest 0's, then a 0 is likely to be read. Thus, the resulting effect is sometimes an OR operation and sometimes an AND operation. Because of this anomaly, in read operation it is

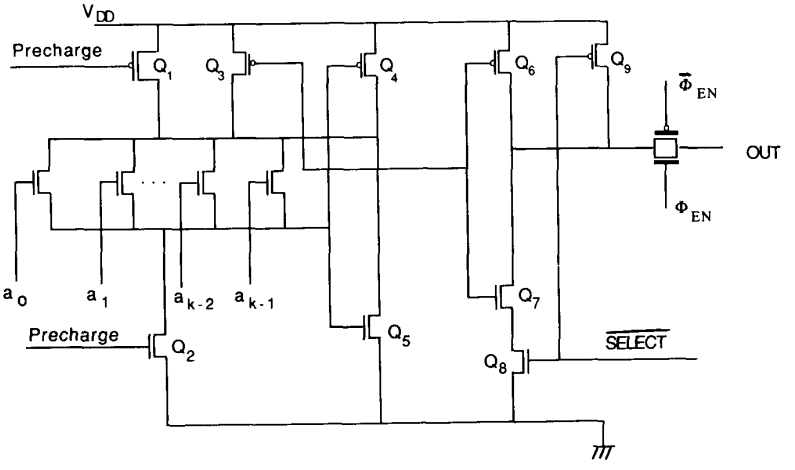

Fig. 3. Modified CMOS bit-line decoder.

not possible to determine by monitoring the data-out buffer whether all the multiple-accessed cells have identical content. This limitation has been circumvented by incorporating a parallel comparator and an error detector in the RAM. The parallel comparator determines whether the content of all the multiple-accessed cells are either all 0 or all 1. If, due to some fault, a write operation on a cell fails or the content of some cells change, all the inputs to the comparator are not identical. The comparator detects this anomalous input and triggers the error latch to indicate the occurrence of a fault.

It should be stressed that only the bit-line decoder is modified to allow multiple access of cells on the selected word line. The word-line decoder is not modified, and word lines are accessed only one at a time. The multiple cells accessed through multiple word lines are also possible. But this needs the sense amplifier to drive many cells at a time. For a moderately large size RAM, this introduces a very high access time delay. By increasing the physical size of the sense amplifier driver, delay can be decreased to a certain extent. But as the sense amplifier driver size is increased, it consumes more power, and also because of its large gate capacitance, the sense amplifier slew rate decreases.

A. Modified Decoder Circuit

A typical CMOS implementation of a decoder circuit is shown in Fig. 3. Transistors $Q_1, \cdots, Q_7$ constitute a normal decoder circuit. Transistors Q_8 and Q_9 have been added so that in the test mode the decoder output can be selected by applying $\overline{\text{SELECT}} = 0$ independent of the input address. In the normal mode of operation, $\overline{\text{SELECT}} = 1$ and the decoder output is selected by the address input $a_0, \cdots, a_{k-1}$. The modified decoder has been simulated using SPICE and the degradation in performance due to additional elements has been found to be an additional 0.1 ns delay (which is 1 percent of the normal decoding delay).

B. Parallel Comparator and Error Detector

The parallel comparator and error detector monitors the output of sense amplifiers connected to bit lines which are selected in parallel. The circuit detects the occurrence of either all 0's or all 1's. In case any selected bit line is different from other bit lines in the same group, it triggers the error latch indicating the occurrence of a fault. Fig. 4 shows the parallel comparator and error detector which compare the contents of cells on the even (odd) bit lines together. Transistors $Q_0, \cdots, Q_{b-1}$ are connected to the sense amplifiers and they are selected by L_1 and L_2. In a normal mode of operation, $L_1 = L_2 = 1$ and the p-type pass transistors isolate the comparator from the sense amplifiers. In the test mode, if $L_1 = 1$ and $L_2 = 0$, all the p-type pass transistors connected to the set of odd bit lines conduct and provide the input to the parallel comparator while even bit lines remain disconnected. If $L_1 = 0$ and $L_2 = 1$, all the p-type transistors connected to the set of even bit lines provide the input to the parallel comparator. Transistors $T_1, \cdots, T_{m-1}$ are connected in parallel and detect a concurrent occurrence of 1 in bit lines set. Transistors $P_1, \cdots, P_{m-1}$ are connected in parallel and detect concurrent occurrence of 0 in bit lines set. Transistors T_0 and P_0 are the precharge transistors which are turned on by precharge clock phase $\phi 1$. Transistor P_m is the discharge transistor which remains cut off during the precharge phase and turns on during the discharge clock phase $\phi 2$. Transistors S_0, S_1, and S_2 form a coincidence detector. If all the selected bit lines are not 0 or 1, then both S_1 and S_2 remain in cutoff and the output of the detector is 1. The output of the detector is connected to an error latch through the pass transistor S_4 that isolates the error latch during phase $\phi 1$. It may be noted that during the precharge phase, the transistor S_0 will be directly shorted through the error amplifier if S_4 does not isolate the coincidence detector from the error amplifier. During phase $\phi 2$, the output of the coincidence detector is connected to the error amplifier through S_4. The error amplifier consists of transistors $V_0, \cdots, V_3$. The error latch output is ERROR = 1, when the selected bit lines are not identical. If the bit lines are all 1's, then S_1 conducts and S_2 remains cut off and the detector output is 0. This holds the error latch output to ERROR = 0. If the bit lines are all 0's then S_2 conducts and S_1 remains cut off, setting the error latch output to ERROR = 0. During

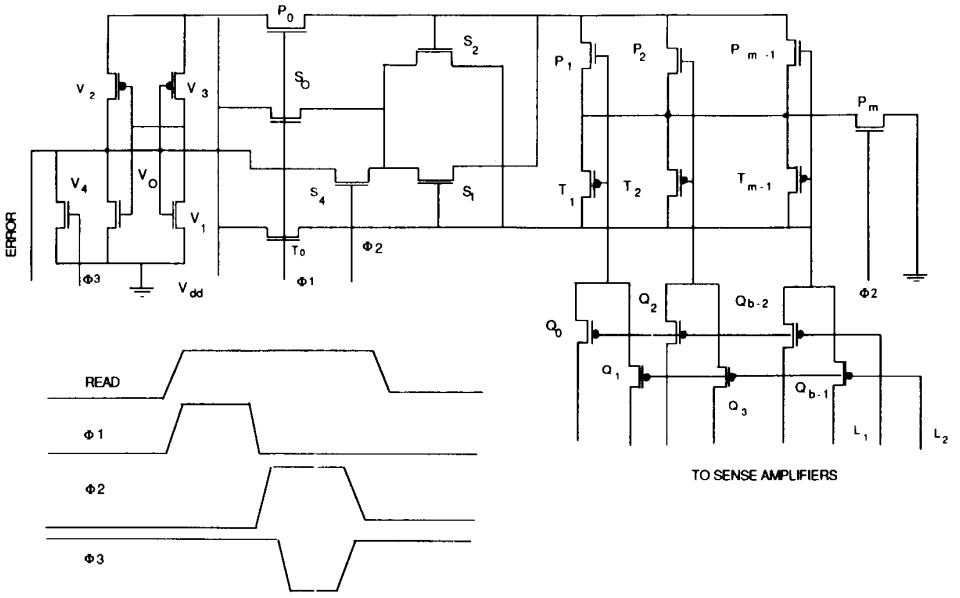

Fig. 4. Parallel comparator and error detector.

write phase and normal mode of operation, the error latch is clamped to zero by V_4. The error detector is inhibited by the discharge transistor P_m during the start of the read phase, when the sense amplifiers' outputs are not identical because of sluggish changes in some of the sense amplifiers.

IV. Algorithms for Testing Pattern-Sensitive Faults

The test algorithms discussed in this paper have $O(\sqrt{n/(pe)})$ complexity, and they can detect the restricted pattern-sensitive fault (PSF) [14]. An unrestricted PSF has been shown by Hayes [15] to have $(3n^2 + 2n)2^n$ test complexity and is impractical in a large size DRAM. In a restricted PSF, an operation $\psi(C_{ij})$ is faulty due to the presence of a specific data pattern in a set of physically adjacent cells (called its neighbors), or due to a specific operation on one or more cells in the neighborhood. Fig. 5 shows the physically adjacent cells of C_{ij} and their neighbors are defined as follows.

Definition 1: A base cell C_{ij} and its four rookwise adjacent neighboring cells $C_{i\pm1,j}$ and $C_{i,j\pm1}$ are called a *von Neumann neighborhood* (or Type-1 neighborhood) of C_{ij}. If the base cell is deleted, the set of remaining cells in the neighborhood is denoted by N_1. Cells $C_{i\pm1,j}$ that share the same word line with the base cell are denoted by N_w, while the cells $C_{i,j\pm1}$ that share the same bit line with the base cell are denoted by N_b, such that $N_1 = N_b \cup N_w$.

Definition 2: A base cell C_{ij} and its eight adjacent neighboring cells $C_{i\pm1,j\pm1}$ are called a *Moore neighborhood* (or Type-2 neighborhood) of C_{ij}. If the Type-1 neighborhood is deleted from the Type-2 neighborhood, the set of remaining cells in the neighborhood is denoted by N_2.

Definition 3: A *static pattern-sensitive fault* (SPSF) is said to occur if an operation $\psi(C_{ij})$ is faulty in the presence of a fixed pattern in a predefined neighborhood of cell C_{ij}.

Definition 4: A *dynamic pattern-sensitive fault* (DPSF) is said to occur if the state of the cell C_{ij} changes because of an operation ψ on one cell in its predefined neighborhood in the presence of a specific pattern.

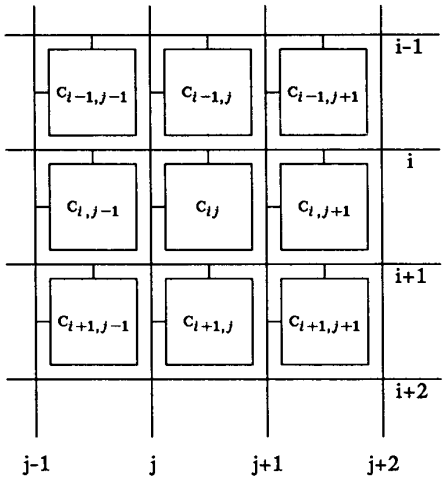

Fig. 5. Base cell C_{ij} and its neighbors.

The above definitions apply to all memory operations, namely, read, transition writes, and nontransition writes. Unless specifically stated in the following discussion, we will assume that only the transition writes and reads are faulty.

Let the cells in the neighborhood be assigned the k distinct positive number in the set $\{0, 1, \cdots, k-1\}$ such that the number i is assigned to the base cell. The content of the neighborhood can be denoted by a state vector $\langle s_0 s_1 \cdots s_i \cdots s_{k-1} \rangle$, where s_j is the state of the cell which has been assigned the number $j \in \{0, 1, \cdots, k-1\}$. Clearly, the state vector space of a neighborhood of size k describes a k-dimensional symmetric Boolean hypercube in which the nodes represent all different possible patterns which can be stored in the neighbor-

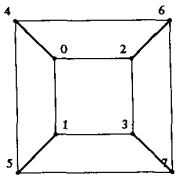

Fig. 6. State space graph for neighborhood with $k = 3$.

hood. A node in the state vector space graph is numbered m, where $m = \Sigma_{j=0}^{k-1} s_j 2^j$ and, evidently, $0 \leq m \leq 2^k - 1$. An edge corresponds to the state transition due to a transition-write operation W_x on a cell in the neighborhood. Let $\psi(j)$ denote an operation of $\psi \in \{W_1, W_0, W_x, R\}$ on the cell numbered j. Thus, $W_x(i)$ changes the state vector from $\langle s_0 s_1 \cdots s_i \cdots s_{k-1} \rangle$ to $\langle s_0 s_1 \cdots \overline{s_i} \cdots s_{k-1} \rangle$. All such $W_x(i)$ operations on 2^k nodes sensitize the transition-write SPSF, and hence at least $2k$ transition-write operations are required to detect all the SPSF's in a neighborhood of k cells. All other operations $W_x(j \neq i)$ sensitize the transition-write DPSF's, and the minimum number of transition writes is equal to $(k - 1)2^k$ (note that each undirected edge in the symmetric k-cube corresponds to two transitions). In Fig. 6, a three-dimensional symmetric Boolean cube represents the state vector space of a hypothetical neighborhood of size $k = 3$. Assuming that $i = 2$ is the base cell, all the diagonal edges between the outer and inner squares sensitize the SPSF's while the rest of the edges sensitize the DPSF's. A naive approach to test these SPSF's and DPSF's will be to perform both the transition writes ($\uparrow$ and $\downarrow$) on each cell of the memory, and to follow each transition write on a cell by read operations over all the cells in its neighborhood. Since there are $k2^k$ transition writes per cell, there will be altogether $(k + 1)k2^k$ operations per cell. Thus, for an n-bit memory organized into a $\sqrt{n} \times \sqrt{n}$ array, the naive approach requires $n(k + 1)k2^k$ operations. This complexity can be reduced by a factor of k if the write operations are cleverly done so as to sensitize several PSF's simultaneously in the overlapping neighborhoods. Notions of graph coloring and tessellations [16], [17] are useful in deriving these test patterns.

A valid test is a tour on the directed Boolean cube that traverses all edges. A tour, which traverses each edge once and only once, is a minimum length test set. Such a tour is called an Eulerian tour on the graph. An example of an Eulerian tour in Fig. 6 is $\langle 0,2,3,1,0,4,6,7,5,4,5,7,6,4, 0,1,3,2,0 \rangle$ recalling the fact that each edge in the Fig. 6 represents two directed edges in opposite directions. Similarly, an Eulerian tour on the Boolean 5-cube gives a minimum length test set for the SPSF's and DPSF's in the Type-1 neighborhood of Fig. 5. Algorithms for deriving an Eulerian tour over a symmetric n-cube are well-known [3], [4], [18].

A. Algorithm for Testing PSF over Type-1 Neighborhood

In order to obtain a test size having an optimal number of transition writes, each memory cell C_{ij} in the RAM is assigned a positive integer $k = (2j + i) \pmod 5$ such that $k \in \{0,1,2,3,4\}$. The above assignment function ensures that each base cell C_{ij}, which is assigned the value k, is surrounded by four cells (N_1) whose assigned values are distinct and not k. Fig. 7 shows the cell assignment values from which it can be seen that every cell that is assigned the value 1 has cells numbered 2,3,4, and 0 in its Type-1 neighborhood. By assigning the cell numbers in this manner, the average number of transition writes per cell can be minimized to 32, when the sequence of writes is derived from an Eulerian tour. This assignment also ensures that the cell values form a periodic pattern that repeats after five successive word lines. A number of other neighborhoods consisting of five cells can be described by the above cell numbering scheme [19]. These neighborhoods, which are identified later on, can be shown to tessellate the memory plane like the Type-1 neighborhood. By superimposing these different neighborhoods, the actual effective physical size of the neighborhood spans over 25 cells.

j \ i	0	1	2	3	4	5	6
0	0	1	2	3	4	0	1
1	2	3	4	0	1	2	3
2	4	0	1	2	3	4	0
3	1	2	3	4	0	1	2
4	3	4	0	1	2	3	4
5	0	1	2	3	4	0	1

Fig. 7. Cell number assignment.

Table II shows the successive bit patterns obtained from an Eulerian cycle over a 5-cube. This constitutes the optimal transition writes necessary to sensitize all the SPSF's and DPSF's. It may be noted that the Hamming distance between any two successive patterns in Table II is always 1. Thus, to obtain a successive pattern, only a single transition write is required on a cell in the neighborhood. The transition write involved in obtaining the mth test pattern from the $(m - 1)$th test pattern can be defined to be an operation $\#m$. Thus, for example, operation #1 in Table II describes the operation of writing a transition 1 ($\uparrow$) on the cell numbered 3 in the neighborhood, and operation #6 implies writing transition 0 ($\downarrow$) on the cell number 4. Let the state of the neighborhood S_{m-1} change to S_m (where $0 \leq S_m, S_{m-1} \leq 31$) due to operation $\#m$ in Table II. Then, in general, the operation is $\uparrow$ on cell numbered $p \in \{0,1,2,3,4\}$ in the neighborhood if $S_m = S_{m-1} + 2^p$ or the operation is $\downarrow$ on the cell numbered p if $S_m = S_{m-1} - 2^p$.

The above set of operations can be applied to the memory locations as described in Algorithm 1. All the memory cells numbered p on a word line are written in parallel using the proposed modified decoder. The word lines are written sequentially so that in w writes (lines e–g of Algorithm 1) all the cells numbered p are applied the operation m and all the neighborhoods in the memory contain the pattern m in Table II.

In one read operation, all bits on a word line are read and compared simultaneously using the parallel comparators. Thus, to read the whole memory in line i of Algorithm 1, w operations are needed. It may be noted that the effects of executing the lines e–k for a particular value of m, a

TABLE II
BIT PATTERNS IN A TYPE-1 NEIGHBORHOOD REPRESENTING AN EULERIAN TOUR

Op #	$s_4 \cdots s_0$	Op #	$s_4 \cdots s_0$	Op #	$s_4 \cdots s_0$	Op #	$s_4 \cdots s_0$
0	00000	40	01001	80	10010	120	01001
1	01000	41	01000	81	10110	121	11001
2	11000	42	01010	82	11110	122	11000
3	10000	43	01011	83	11010	123	01000
4	10001	44	01111	84	01010	124	01010
5	11001	45	01110	85	01110	125	11010
6	01001	46	01100	86	00110	126	11011
7	00001	47	01101	87	00010	127	01011
8	00011	48	11101	88	00011	128	01001
9	01011	49	11100	89	00111	129	01011
10	11011	50	11110	90	01111	130	01111
11	10011	51	11111	91	01011	131	01101
12	10010	52	11011	92	11011	132	00101
13	11010	53	11010	93	11111	133	00111
14	01010	54	11000	94	10111	134	00011
15	00010	55	11001	95	10011	135	00001
16	00110	56	10001	96	00011	136	10001
17	01110	57	10000	97	10011	137	10011
18	11110	58	10010	98	10010	138	10111
19	10110	59	10011	99	00010	139	10101
20	10111	60	10111	100	00000	140	11101
21	11111	61	10110	101	10000	141	11111
22	01111	62	10100	102	10001	142	11011
23	00111	63	10101	103	00001	143	11001
24	00101	64	10001	104	00101	144	11000
25	01101	65	10101	105	10101	145	11010
26	11101	66	11101	106	10100	146	11110
27	10101	67	11001	107	00100	147	11100
28	10100	68	01001	108	00110	148	10100
29	11100	69	01101	109	10110	149	10110
30	01100	70	00101	110	10111	150	10010
31	00100	71	00001	111	00111	151	10000
32	00101	72	00000	112	01111	152	00000
33	00100	73	00100	113	11111	153	00010
34	00110	74	01100	114	11110	154	00110
35	00111	75	01000	115	01110	155	00100
36	00011	76	11000	116	01100	156	01100
37	00010	77	11100	117	11100	157	01110
38	00000	78	10100	118	11101	158	01010
39	00001	79	10000	119	01101	159	01000
						160	00000

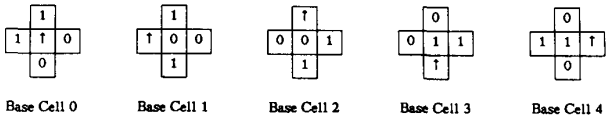

Fig. 8. Pattern written in different neighborhoods by operation 55.

transition write on p sensitizes one SPSF for the pattern in line m for all cells numbered p and it also sensitizes one DPSF for all cells numbered other than p. Thus, for example, operation 55 of Table II in which a transition write (↑) is made on cells numbered 0, the lines e–k in Algorithm 1 will sensitize all cells numbered 0 for the SPSF where a ↑ cannot be made in the presence of binary pattern 1100 in cells 4,3,2, and 1, respectively. Simultaneously, it will sensitize four DPSF's in neighborhoods in which cells 4,3,2, and 1 are the base cells. This is illustrated in Fig. 8. By executing the lines e–k 160 times for all the patterns in Table II, all the 32 SPSF's and 128 DPSF's associated with each memory cell are tested by Algorithm 1.

Algorithm 1. Parallel PSF Test in RAM:

a) Write 0 into every cell of RAM.
b) Set $m = 1$.
c) Find the cell number, $p \in \{0,1,2,3,4\}$, in operation #m from Table II on which the transition write is to be made.
d) Set $j = 0$.
e) Do in parallel the operation #m on C_{ij} for all bit lines if $(2j + i) \pmod 5 = p$.
f) Set $j = j + 1$.
g) If $j < w$, go to e); else go to h).
h) Set $j = 0$.
i) Read in parallel all cells C_{ij} and see if ERROR = 0, (if ERROR = 1, a fault is detected).
j) Set $j = j + 1$.
k) If $j < w$, go to i); else go to 1).
l) Set $m = m + 1$.
m) If $m < 161$, go to c); else exit success.

Since there are altogether 160 transition writes (plus 1 write operation is for memory initialization) and the memory is read 160 times and each operation is performed on w word lines, the overall complexity of the algorithm is $321w$. In an n-bit RAM organized into p identical square submatrices, Algorithm 1 will require $321\sqrt{n/(pe)}$ operations to test all the SPSF's and DPSF's. It may be noted that Algorithm 1 tests the pattern-sensitive faults due to transition writes only. Usually, for DRAM's using three or more transistors in each memory cell, a read operation does not result in destruction of the content of the cell. Such a read fault that depends on the pattern of stored data in the Type-1 neighborhood will be detected by the parallel comparator. Pattern-sensitive faults due to destructive-read operations in a single-cell, high-density memory cannot be tested by Algorithm 1 and will be discussed later on.

1) Computation of Effective Size of the Neighborhood: The cell number assignment allows one to describe a number of memory plane tessellations as shown in the Fig. 9. Neighborhoods in these tessellations are the corresponding tile shape. It can be seen below that by linear translations of these tile geometries, the memory plane can be covered without overlapping (i.e., tessellated). Pattern-sensitive fault procedures described in Algorithm 1 will satisfy over each of the neighborhood. The effective physical size of the neighborhood can be estimated by superimposing these tile geometries. But it is necessary to invoke a constraint that all cells numbered i ($0 \le i \le 4$) are mutually noninteractive and consistent in the sense that a transition write on a cell numbered i neither affects the content of another cell numbered i in the same neighborhood, nor does it mask the coupling effect between any other coupling cell i and a coupled cell $j \neq i$, both in the same neighborhood.

B. Algorithm for Testing Symmetric PSF in Type-2 Neighborhood

In order to detect all the static pattern-sensitive faults over the Moore's neighborhood, every cell in the memory should make both ↑ and ↓ transitions in the presence of all $2^8 = 256$ patterns in the memory. Thus, overall 512 transition writes are

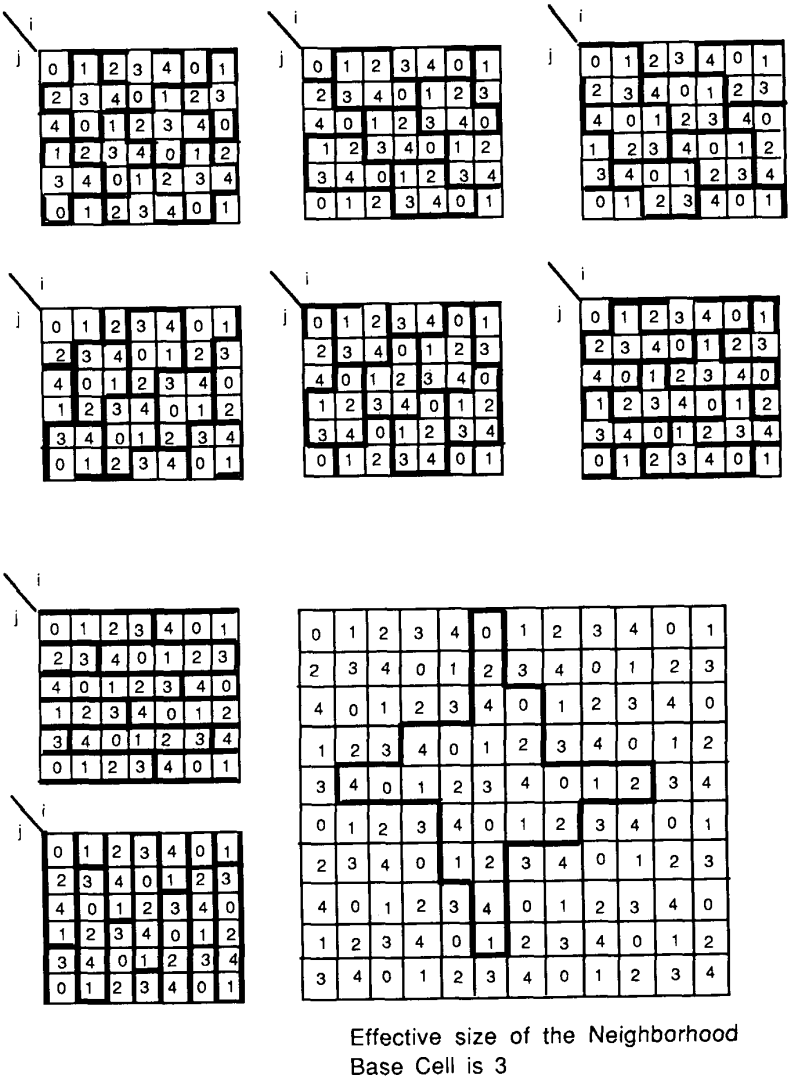

Fig. 9. Neighborhood effective size.

necessary for each cell in the memory to detect the SPSF's. In order to detect all the DPSF's, each base cell should be tested by a reading operation whenever a transition write is made over a cell in the neighborhood of a base cell. Since each cell can make two types of transition writes for all the possible binary patterns in the other eight cells and there are altogether eight neighbors of a base cell, there are altogether $2 \times 2^8 \times 8 = 4096$ transition writes in the neighborhood of a cell to test all the DPSF's. This requires a large amount of test time.

In this section, a new approach has been adopted in which cells in the 9-neighborhood are categorized into four logical groups, viz., the base cell, bit-line neighbors (N_b), word-line neighbors (N_w), and diagonal neighbors (N_2) as described in the Definition 1. It may be noted that a pattern-sensitive fault models the adjacency effect between a base cell and its physically neighboring cells. It is typically due to leakage effects between a memory cell and its adjoining cells in the presence of a particular data pattern in the neighboring cells [20]–[22]. It has been found that the leakage is maximum when the symmetrically located cells contain the same bit patterns [21]. By the above classification, many unnecessary binary combinations are avoided. For example, let there be a situation when a read operation is made to verify the transition write ↑ when its bit-line neighbors $C_{i,j+1}$ and $C_{i,j-1}$ contain 0. Clearly, at first, the bit line i will be precharged to some high potential. Now, if any of the access transistors of the bit-line neighbors is weak, then in the presence of 0's in the bit-line neighborhood, the precharge level in the bit line will be degraded, and the sense amplifier on the ith bit line will fail to detect a 1 in the base cell. Similarly, if there is a weak transistor in cell $C_{i+1,j}$, which does not allow the base cell to make a ↑ transition because $C_{i+1,j}$ is at state 0, then it is enough to test the fault when the symmetrically located cell $C_{i-1,j}$ is also at state 0, since then the leakage effect will be predominant. Thus, if a fault does not occur when both the cells $C_{i\pm1,j}$ are at state 0, then it will not occur when $C_{i+1,j}$ and $C_{i-1,j}$ have different states.

Definition 5: A *symmetric static pattern-sensitive fault*

(SSPSF) is said to occur if an operation $\psi(C_{ij})$ is faulty in the presence of a fixed symmetric pattern in $N_b \cup N_w \cup N_2$. The pattern is defined to be symmetric if all cells of $N_x \in \{N_b, N_w, N_2\}$ have the same value. This fault is denoted by $\langle \psi(C_{ij})/s(N_b)s(N_w)s(N_2)\rangle$, where $s(N_b)$, $s(N_w)$, and $s(N_2)$ represent the states of all cells in N_b, N_w, and N_2, respectively.

Definition 6: A *symmetric dynamic pattern-sensitive fault* (SDPSF) is said to occur if the state of the cell C_{ij} changes because of an operation on one or more cells in $N_b \cup N_w \cup N_2$ in the presence of a fixed symmetric pattern. An SDPSF is denoted by $s_{ij}/\psi(C_{i\pm1,j})s(N_w)s(N_2)$, $s_{ij}/s(N_b)\psi(C_{i,j\pm1})s(N_2)$, or $s_{ij}/s(N_b)s(N_w)\psi(N_2)$, depending on whether the operation made on cells in N_b, N_w, or N_2 changes s_{ij}.

Clearly, $s(N_x) = 1$ if all the cells in the neighborhood $N_x \in \{N_b, N_w, N_d\}$ are 1 and $s(N_x) = 0$ if all the cells in N_x are 0. It will be assumed that during testing of a pattern-sensitive fault, the contents of all the cells in N_x are always identical. Since all the cells in N_x may not be written in parallel, a read operation to test a fault is only made when all the cells in N_x are written identically. If all the cells in N_x make a transition write ↑, then $\psi(N_x)$ will be denoted by ↑. On the contrary, if all the transition writes in N_x are ↓, then $\psi(N_x)$ will be denoted by ↓. In order to detect the dynamic pattern-sensitive fault in N_x, it will be assumed that the fault will cause at most one transition in the base cell, whether the operations on the cells of N_x are applied one cell at a time or two cells at a time.

Definition 7: If the base cell i is in state $s_i \in \{0,1\}$ and a transition write ψ is made on one cell or simultaneously over two cells belonging to N_x, then an SDPSF is said to be consistent if the state of cell i does not change back from $\bar{s}_i$ to s_i after the successive ψ operation on the remaining cells in N_x.

Utilizing this notation, the SSPSF's and SDPSF's can be represented by Table III. The four-tuples are as described in Definitions 5 and 6. Since, there are 16 SSPSF's and 48 SDPSF's associated with each cell in the memory, it can be easily seen that at least $4 \times 2^4 = 64$ transition writes in the neighborhood will be necessary to sensitize all the SSPSF's and SDPSF's of a cell in a DRAM. A straightforward extension to an n-cell memory will require $64n$ transition writes. However, in this section, it will be shown that by cleverly combining the transition write sequence in overlapping neighborhoods, the total number of transition writes can be reduced to $16n$.

In order to accomplish the minimal transition writes per memory cell, at first, each cell C_{ij} in the memory will be assigned a positive number $k \in \{0,1,2,3\}$ such that $k = 2(i \bmod 2) + j \bmod 2$. Thus, the memory cells are divided into four types of cells 0,1,2, and 3, as shown in Fig. 10. It will be shown later that a transition write on a cell followed by a suitable sequence of read operations on the adjoining cells will simultaneously sensitize four pattern-sensitive faults and thereby the number of transition writes on each cell can be reduced by a factor of 4.

In order to obtain a test procedure which needs only 16 transition write sequences per cell, a graph theoretic approach similar to the earlier algorithm can be used. The 4-tuple,

TABLE III
ALL POSSIBLE SSPSF'S AND SDPSF'S

Fault Type	Fault Notation
SSPSF	↑/000, ↑/100, ↑/010, ↑/110, ↑/001, ↑/101, ↑/011, ↑/111. ↓/000, ↓/100, ↓/010, ↓/110, ↓/001, ↓/101, ↓/011, ↓/111.
SDPSF	0/↑00, 0/↑10, 0/↑01, 0/↑11, 1/↑00, 1/↑10, 1/↑01, 1/↑11, 0/↓00, 0/↓10, 0/↓01, 0/↓11, 1/↓00, 1/↓10, 1/↓01, 1/↓11, 0/0↑0, 0/1↑0, 0/0↑1, 0/1↑1, 1/0↑0, 1/1↑0, 1/0↑1, 1/1↑1, 0/0↓0, 0/1↓0, 0/0↓1, 0/1↓1, 1/0↓0, 1/1↓0, 1/0↓1, 1/1↓1, 0/00↑, 0/10↑, 0/01↑, 0/11↑, 1/00↑, 1/10↑, 1/01↑, 1/11↑, 0/00↓, 0/10↓, 0/01↓, 0/11↓, 1/00↓, 1/10↓, 1/01↓, 1/11↓.

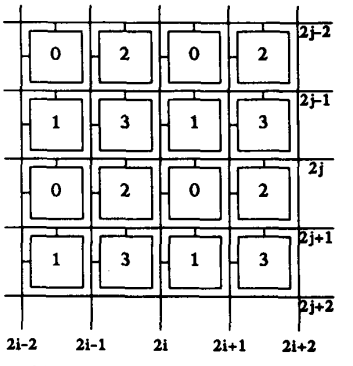

Fig. 10. Cell type assignment.

$\langle s(C_{ij})s(N_b)s(N_w)s(N_2)\rangle$ describes a state space of 16 nodes (states) where each node represents the symmetric binary pattern in the 9-neighborhood. These nodes are numbered 0–15, depending on the binary pattern in the 4-tuple. The base cell state is the MSB in the 4-tuple. By complementing the pth bit in the 4-tuple, the memory state of the neighborhood will change from k to $k - 2^p$, if the pth bit changes from 0 to 1, and from k to $k + 2^p$, if the pth bit changes from 1 to 0. All these transitions from one state to another state represent edges in the state space graph, and the resulting graph describes a symmetric four-dimensional cube as shown in Fig. 11. Clearly, node 12 in the figure represents that the base cell and its bit-line neighbors contain 1, and the word line and diagonal neighbors of the base cell contain 0. By making ↑ operation on the word-line neighbors, N_w the content of the Type-2 neighborhood, will change and it will be represented by the node 14 in Fig. 11. A directed edge from node 12 to node 14 represents this transition-write operation. In general, each directed edge corresponds to a transition write over all the cells having the same number, $k \in \{0,1,2,3\}$ in a neighborhood, and since there are always two anti-parallel directed edges, these two edges are represented as a single undirected edge in Fig. 11. The set of thick edges corresponds to changing the state of the base cell and will pertain to sensitizing the SSPSF's. Other edges pertain to sensitizing the SDPSF's. Similar to Algorithm 1, an algorithm with optimal transition writes can be obtained by deriving the write

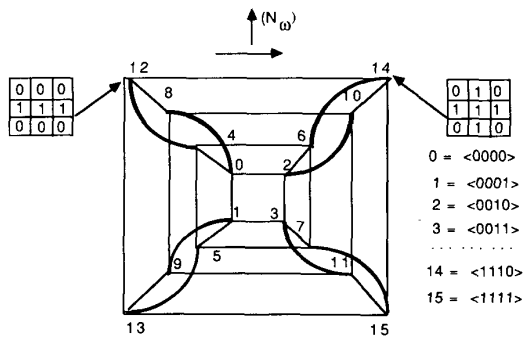

Fig. 11. State space graph for neighborhood with $k = 4$.

sequences from an Eulerian tour over the symmetric 4-cube. The resulting algorithm will have $97w$ complexity, involving $32w$ transition writes, $64w$ read, and w write operations corresponding to memory initialization. It may be noted that even though 64 transition writes over the symmetric 4-cube are needed to sensitize all the SSPSF's and SDPSF's, only $32w$ transition writes are needed over the entire memory.

Even though the test algorithm derived from an Eulerian tour has an optimal test length, for many applications like in the embedded environments, it is desirable to make a tradeoff between the test length and the built-in self-test (BIST) hardware. It is shown in [23] and [24] that the test generator circuit can be simplified considerably if the sequence of transition writes is derived by decomposing an Eulerian tour over the symmetric 4-cube into the following eight disjoint Hamiltonian cyclic tours described over a subgraph of the symmetric 4-cube:

$$H1\colon \langle 0,2,6,14,15,13,9,1,0\rangle$$

$$H2\colon \langle 0,1,9,13,15,14,6,2,0\rangle$$

$$H3\colon \langle 0,4,6,7,15,11,9,8,0\rangle$$

$$H4\colon \langle 0,8,9,11,15,7,6,4,0\rangle$$

$$H5\colon \langle 12,4,5,7,3,11,10,8,12\rangle$$

$$H6\colon \langle 12,8,10,11,3,7,5,4,12\rangle$$

$$H7\colon \langle 12,13,5,1,3,2,10,14,12\rangle,$$

and

$$H8\colon \langle 12,14,10,2,3,1,5,13,12\rangle.$$

These tours are shown in Fig. 12, where the set of dark edges constitutes two Hamiltonian cycles, one in clockwise and the other in anti-clockwise direction. Thus, Fig. 12(a) represents the Hamiltonian cycles $H1$ and $H2$, Fig. 12(b) represents the cycles $H3$ and $H4$, Fig. 12(c) represents the cycles $H5$ and $H6$, and finally, Fig. 12(d) represents the cycles $H7$ and $H8$. The length of each Hamiltonian cycle is eight, and altogether eight Hamiltonian cyclic tours are made. Thus, all the 64 edges of the symmetric 4-cube are traversed and thereby all the 16 SSPSF's and 48 SDPSF's are sensitized. Initially, the memory is initialized to zero and the first four Hamiltonian cycles are performed as indicated above. Then the memory is

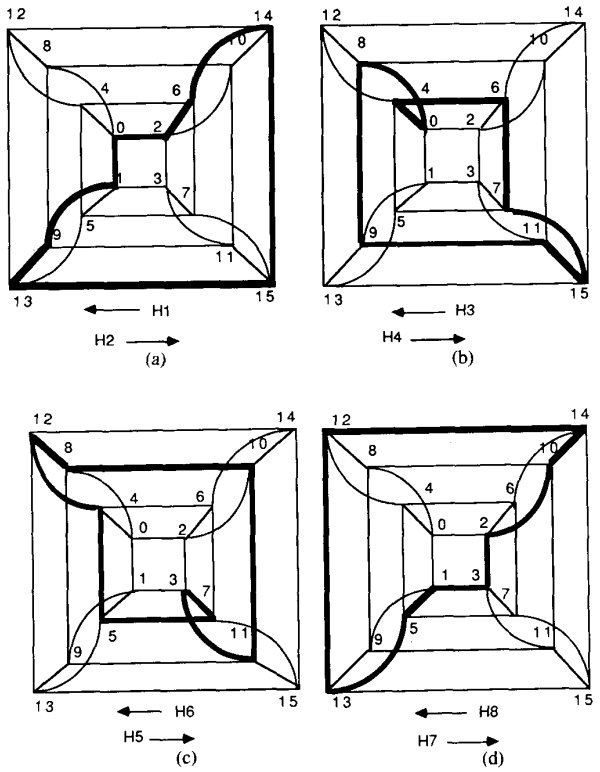

Fig. 12. Hamiltonian cycles on subcubes.

reinitialized such that it contains a column bar pattern of 0's and 1's, so that the content of the Type-2 neighborhoods in the memory is represented by node 12. This needs an additional w write operations which for a 256 Kbit square memory subarray having a memory cycle time of 50 ns will take an extra 25 μs. After memory reinitialization, the rest of the Hamiltonian tours are performed. Thus, the overall tour can be represented as $\langle H1, H2, H3, H4, \langle 0,12\rangle, H5, H6, H7, H8\rangle$, and it consists of 65 transition-write operations. Initially all the cells in the Type-2 neighborhood contain 0 and successively the transition-write operations are made to change the state of the neighborhood. Thus, after the first operation the state of the neighborhood changes to 2, and after the second operation the state of the neighborhood changes to 6, and so on. In general, an operation m will change the state S_{m-1} to S_m, where $S_m \in \{0,15\}$ denotes the state of the neighborhood after the mth operation is applied. After each operation, the whole memory is read to find out whether any SSPSF or SDPSF has occurred. The above test procedure is described in Algorithm 2 which uses these write sequences to test the SSPSF's and SDPSF's for every cell in the memory over its 9-neighborhood.

In Algorithm 2, all cells numbered $k \in \{0,1,2,3\}$ make an upward ($\uparrow$) and a downward ($\downarrow$) transition write in the presence of all binary patterns in all other cells whose numbers are not k. There are altogether eight upward transitions in cells numbered k for eight distinct binary patterns. Also, there are eight downward transitions in cells k for all eight distinct binary patterns. Hence, all 16 operations to sensitize SSPSF's are performed in Algorithm 2. Since after each transition

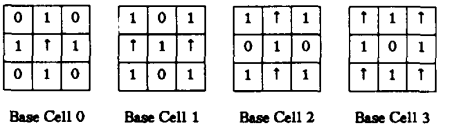

Fig. 13. Pattern written on different neighborhoods by eleventh operation.

write, all the cells are read, if any SSPSF occurs, it will be detected by the parallel comparator and error latch. Also, because of the neighborhood relationship in Fig. 10, every transition write on cells numbered k will also sensitize the SDPSF's for the other three cells for which the number is not k. For example, in the third operation in $H1$, the state of the Type-2 neighborhood changes from 6 to 14 by writing ↑ in the cell numbered 0 while the contents of the cells numbered 1,2, and 3 remain the same to 1,1, and 0, respectively. The succeeding read operation in line 4 of Algorithm 2 detects an SSPSF in all neighborhoods for which the base cell is 0, and SDPSF's in all other neighborhoods for which base cell is not 0 where the fault occurs due to transition in cell numbered 0. The effect of third operation on different neighborhoods is shown in Fig. 13. Thus, Algorithm 2 which makes all the 65 transition writes over the entire memory will sensitize both the SSPSF's and SDPSF's for every cell in the memory.

Algorithm 2. Test of Pattern-Sensitive Faults over Type-2 Neighborhood:

1) Initialize all the cells to 0.
2) Set $m = 1$.
3) If the transition write in the mth operation is on cells numbered $p \in \{0,2\}$, write on all cells having number p and are on even word lines.
4) Read the memory in parallel. Check if ERROR = 0, else error is detected.
5) If the transition write in the mth operation is on cells numbered $p \in \{1,3\}$, write on all cells having number p and are on odd word lines.
6) Read the memory in parallel. Check if ERROR = 0, else error is detected.
7) Set $m = m + 1$.
8) If $m < 66$, go to 7); else exit successfully.

V. Test Procedures for Other Faults

Nontransition Write Operations: Algorithms 1 and 2 detect the pattern-sensitive faults in the memory due to *transition* writes. Since the *nontransition* write does not change the state of a cell, it is unlikely to cause a failure. In memories where nontransition writes are also faulty, the algorithms in the preceeding section can be easily augmented by following up each transition write with a nontransition write. After each nontransition write, the entire neighborhood is read to detect the occurrence of any pattern-sensitive faults.

Destructive Read Operations: In a switched-capacitor, single-transistor DRAM which employs a *destructive*-read operation, the failure may result during the precharge, sensing, and restoration phases in a read operation. Since in the proposed parallel testing scheme, multiple cells are read and compared simultaneously, any failure in a cell that results during the precharge and sensing by the sense amplifiers will be automatically detected by the parallel comparator. A fault that occurs during the restoration phase when the original data in the cell are written back, will also be tested by Algorithms 1 and 2, provided the faulty read operation is a pattern-independent fault (PIF). It can be shown that in these algorithms every memory cell is read successively twice or more, without any intervening write operations on it. Therefore, a fault occurring in the restoration phase will be detected by Algorithms 1 and 2. But in order to test the read faults in the restoration phase which only depend on a specific stored pattern in the neighborhood, every read operation in these algorithms should be followed by an extra read operation. Read operations, which may result in static and dynamic pattern-sensitive faults similar to transition writes, can also be tested by reading all the cells in the neighborhood after each read operation on a cell is made. The complexity of these pattern-sensitive faults over the Type-1 and Type-2 neighborhoods due to the different types of memory operations are shown in Table IV. It may be emphasized that for a particular processing technology, certain operations are more likely to result in pattern-sensitive faults than others (e.g., a transition write may cause more pattern-sensitive faults than a read operation), and hence different test sizes are indicated here.

TABLE IV
TEST COMPLEXITIES OF THE DIFFERENT PSF'S

Neighborhood	Fault Coverage	Complexity
Type-1	F11 = PSF for transition writes + PIF for reads	$321\sqrt{n}$
Type-2	F21 = PSF for transition writes + PIF for reads	$98\sqrt{n}$
Type-1	F12 = F11 + PSF for nontransition writes	$641\sqrt{n}$
Type-2	F22 = F21 + PSF for nontransition writes	$194\sqrt{n}$
Type-1	F13 = F12 + SPSF for read	$801\sqrt{n}$
Type-2	F23 = F22 + SPSF for read	$258\sqrt{n}$
Type-1	F14 = F13 + DPSF for read	$1441\sqrt{n}$
Type-2	F24 = F23 + DPSF for read	$450\sqrt{n}$

$p = 1$ and $e = 1$

Decoder Faults: A fault-free decoder makes a bijective mapping of the input addresses onto the memory cells. A faulty decoder may cause three types of fault syndromes. First, an input address may not map onto any memory cell, and the resulting failure is known as a *no-access* fault. Second, an input address may select multiple cells for reading or writing, and the resulting failure is commonly known as *multiple-cells access* fault. Third, a memory cell may be accessed by multiple addresses, and the resulting failure is called *multiple-address access* fault. The different decoder mappings are shown in Fig. 14. A no-access fault usually results in a stuck-at fault and will be detected by the pattern-sensitive algorithms discussed here. It may be noted that in the event of a stuck-at fault in a cell, it manifests static pattern-sensitive faults for all the possible patterns in the neighbor-

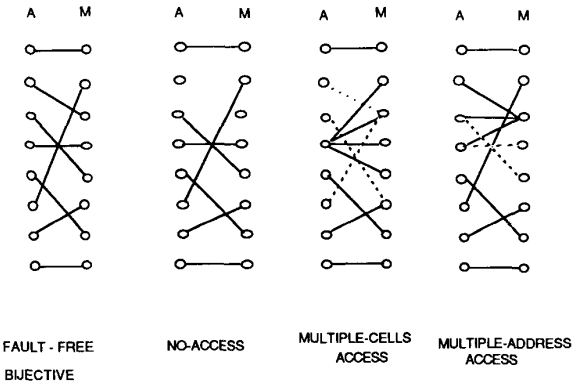

Fig. 14. Fault-free and faulty decoding.

hood. Multiple-cells access and multiple-address access faults cannot be detected by Algorithms 1 and 2. It may be noted that the bit-line decoders are grouped into g classes in the test mode, and the multiple-access faults in the bit-line decoder will be masked if the fault results in accessing bit lines within the same class. Also, it may be noted that even though the word-line decoder is not modified, these faults in the word-line decoder will not be detected by Algorithms 1 and 2. This is because in these algorithms cells having identical number are at first written in all word lines before verifying individual word-line-write operations.

In order to test these faults, a marching-type test procedure as shown in Algorithm 3 is needed. Steps 1–4 in the algorithm detect the multiple-access faults in a bit-line decoder by using $2 + 4\sqrt{n}$ operations. Steps 5–8 involve $6\sqrt{n}$ operations and are used to detect multiple-access faults in a word-line decoder. In the fault model, it is assumed that multiple access may occur asymmetrically in the sense that a line r may be accessed with a line s while the line r is selected, but the line r is not necessarily accessed along with the line s when the line s is selected.

Algorithm 3. Bit-Line and Word-Line Decoder Test:

1) Write in parallel ZERO in all cells on the arbitrarily selected word line W_j.
2) Read and compare in parallel all the cells on W_j.
   /* multiple access where decoder selects the correct bit line r along with one or more bit lines $s > r$ */
3) Starting from the cell at the crosspoint of B_0 and W_j, for each cell on W_j, at first write ONE and read the cell (one cell at a time in ascending order of the bit line).
   /* multiple access where decoder selects the correct bit line r along with one or more bit lines $s < r$ */
4) Starting from the cell at the crosspoint of $B_{\sqrt{n}-1}$ and W_j, for each cell on W_j, at first write ZERO and read the cell (one cell at a time in descending order of the bit line).
   /* Word-Line Decoder Multiple Access Test */
5) Write ZERO in all cells on the arbitrarily selected bit line B_i.
6) Read all the cells on B_i.
   /* multiple access where decoder selects the correct word line r along with one or more word lines $s > r$ */
7) Starting from the cell at the crosspoint of W_0 and B_i, for each cell on B_i, at first write ONE and read the cell (one cell at a time in ascending order of the word line).
   /* multiple access where decoder selects the correct word line r along with one or more word lines $s < r$ */
8) Starting from the cell at the crosspoint of $W_{\sqrt{n}-1}$ B_i, for each cell on B_i, at first write ZERO and read the cell (one cell at a time in descending order of the word line).

Faults in Parallel Comparator: It may be noted that the testable hardware in Fig. 4 is only tested partially by Algorithm 2. When all the even or the odd lines contain 0, the n-type transistors $P_0, \cdots, P_{m-1}$ will be tested for stuck-at 1 faults. When all the even or the odd lines contain 1, all the p-type transistors $T_0, \cdots, T_{m-1}$ will be tested for stuck-at 0 faults. The rest of the stuck-at faults in the testable hardware can be tested by Algorithm 4. Lines 1 and 2 are used to initialize the memory, and to verify this initialization, lines 3 and 4 redundantly detect the stuck-at 1 faults in n-type transistors and stuck-at 0 faults in p-type transistors. Line 5 verifies the stuck-at 0 fault in p-type transistors, and line 6 verifies the stuck-at 1 fault in n-type transistors. The overall complexity of Algorithm 4 is $4\sqrt{n} + 4$.

It may be noted that the parallel comparator does not distinguish between all the 0's and all 1's data, and the error detector may fail to indicate the occurrence of a reversal of data. Such a fault can be detected by comparing the value of data-out buffer to the expected data.

Algorithm 4. Procedure to Test the Parallel Comparator:

1) Initialize the odd cells on the first word line to 0.
2) Initialize the even cells on the first word line to 1.
3) Parallel compare the odd bit lines and check if ERROR = 0.
4) Parallel compare the even bit lines and check if ERROR = 0.
5) FOR each odd cell in the selected word line, DO
   {Write a 1, and test if ERROR = 1;
   Write a 0, and test if ERROR = 0.}
6) FOR each even cell in the selected word line, DO
   {Write a 0, and test if ERROR = 1;
   Write a 1, and test if ERROR = 0.}

VI. Conclusions

This paper has discussed an efficient technique to speed up the RAM test algorithms. Specifically, it has proposed two test algorithms to test the pattern-sensitive faults in a RAM. Algorithm 1, requiring $321\sqrt{n/(pe)}$ operations, detects all the static and dynamic pattern-sensitive faults in the memory over a localized neighborhood of five cells, called here the Type-1 neighborhood. By identifying other neighborhood geometries similar to pentomino tiles [25], [26], the algorithm has been identified to actually cover a restricted type of pattern-sensitive fault over 25 cells. The basic algorithms test the pattern-sensitive faults which occur due to transition writes. The algorithms have been extended to test the pattern-sensitive faults due to transition writes and read operations, both in a DRAM with destructive and nondestructive read. The algorithm derives its transition writes sequence from an Eulerian tour over a symmetric 5-cube, and it employs an optimal number of transition writes. As opposed to this technique, Algorithm 2, which tests symmetric pattern-sensitive faults over the Type-2 neighborhood consisting of nine cells, derives its transition writes sequence from the knowledge of eight disjoint Hamiltonian cycles over the subgraphs of the symmetric 4-cube. For an n-bit RAM organized into p submatrices, the resulting algorithm has a complexity of $98\sqrt{n/(pe)}$ operations, using an extra $\sqrt{n/(pe)}$ transition writes. For a 4 Mbit memory organized into 16 squares subarrays and with 50 ns access time, this needs an extra 25 μs compared to a similar test procedure where the transition writes are derived from an Eulerian walk as in Algorithm 1. However, this simplifies the test generator circuit considerably, and the scheme is useful for built-in self-test applications.

Even though in this paper only the testing of pattern-sensitive faults has been discussed, the proposed design-for-testability technique can be readily used to speed up the conventional algorithms to test the stuck-at and 2-coupling faults. Classical tests like the column bar and the checker board detect the stuck-at faults in memory arrays using $4n$ operations. Marching tests detect coupling faults between two arbitrary cells in the memory. Several versions of the marching tests can be found in the literature. Nair, Thatte, and Abraham [27] originally proposed a linear marching test algorithm which has complexity of $30n$ operations. Suk and Reddy [13] improved the complexity to $14n$ operations by slightly modifying the fault model, and Marinescu [28] improved the complexity to n operations for a more restricted coupling fault model. All these algorithms can be speeded up by a factor of $O(\sqrt{n/(pe)})$, but due to the parallel operations some of the coupling faults will be masked. However, the parallel algorithms for stuck-at and pattern-sensitive faults do not mask any fault.

The proposed implementation scheme for the parallel testing uses minimal extra hardware. The parallel comparator consists of $2b + 12$ transistors and the extra hardware in the modified decoder is $2 \log_2 b$ transistors. Thus, the overall extra hardware is only $2b + 2 \log_2 b + 11$ transistors, and the overall chip area expansion is only 0.4 percent for a 256 Kbit DRAM. The proposed technique needs only one transistor to fit within the pitch width and easily fits even for the vertically integrated, single-cell DRAM design with trench-type capacitor having intercell pitch width of 3λ [29].

References

[1] L. L. Lewyn and J. D. Meindl, "Physical limits of VLSI DRAM's," *IEEE J. Solid-State Circuits,* vol. SC-20, pp. 231–241, Feb. 1985.

[2] M. A. Breuer and A. D. Friedman, *Diagnosis and Reliable Design of Digital Systems,* Woodland Hills, Los Angeles, CA, 1976.

[3] J. P. Hayes, "Testing memories for single-cell pattern-sensitive faults," *IEEE Trans. Comput.,* vol. C-29, pp. 249–254, Mar. 1980.

[4] D. S. Suk and S. M. Reddy, "Test procedures for a class of pattern-sensitive faults in semiconductor random-access memories," *IEEE Trans. Comput.,* vol. C-29, pp. 419–429, June 1980.

[5] A. Fuentes *et al.*, "Random testing versus deterministic testing of RAM's," in *Proc. 16th Fault-Tolerant Comput. Symp.,* July 1986, pp. 266–271.

[6] P. Mazumder, "An efficient design of embedded memories and their testability analysis using Markov chains," in *Proc. Int. Conf. Wafer Scale Integration,* Jan. 1989, pp. 389–400.

[7] M. A. Rich and D. E. Gentry, "The economics of parallel testing," in *Proc. Int. Test Conf.,* 1983, pp. 656–661.

[8] Y. You and J. P. Hayes, "A self-testing dynamic RAM chip," *IEEE J. Solid-State Circuits,* vol. SC-20, pp. 428–435, Feb. 1985.

[9] K. Kinoshita and K. K. Saluja, "Built-in testing of memory using on-chip compact testing scheme," in *Proc. Int. Test Conf.,* 1984, pp. 271–281.

[10] K. T. Le and K. K. Saluja, "A novel approach for testing memories using built-in self testing technique," in *Proc. Int. Test Conf.,* 1986, pp. 830–838.

[11] T. Sridhar, "A new parallel test approach for large memories," in *Proc. Int. Test. Conf.,* 1985, pp. 462–470.

[12] S. H. Han and M. Malek, "Two-dimensional multiple-access testing technique for random-access memories," in *Proc. Int. Conf. Comput. Design,* Oct. 1986.

[13] D. S. Suk and S. M. Reddy, "A march test for functional faults in semiconductor random-access memories," *IEEE Trans. Comput.,* vol. C-30, pp. 982–985, Dec. 1981.

[14] J. R. Brown, "Pattern sensitivity in MOS memories," in *Dig. Symp. Testing Integrated Semiconductor Memories Comput. Mainframes,* Oct. 1972, pp. 33–46.

[15] J. P. Hayes, "Detection of pattern-sensitive faults in random-access memories," *IEEE Trans. Comput.,* vol. C-24, pp. 150–157, Feb. 1975.

[16] P. Mazumder, J. H. Patel, and W. K. Fuchs, "Design and algorithms for parallel testing of random-access and content-addressable memories," in *Proc. Design Automat. Conf.,* vol. 24, July 1987, pp. 688–694.

[17] P. Mazumder, "Planar decomposition for quadtree data structure," *Comput. Vision, Graph. Image Processing,* vol. 38, pp. 258–274, June 1987.

[18] C. L. Liu, *Introduction to Combinatorial Mathematics.* New York: McGraw-Hill, 1968.

[19] P. Mazumder, "A new strategy for octtree representation of three-dimensional objects," *Proc. Comput. Vision Pattern Recognition,* June 1988, pp. 270–275.

[20] ——, "Parallel algorithms for parametric faults testing in three-dimensional DRAM," *IEEE J. Solid-State Circuits,* vol. 23, pp. 933–941, Aug. 1988.

[21] T. C. Lo and M. R. Guidry, "An integrated test concept for switched-capacitor dynamic MOS RAM's," *IEEE J. Solid-State Circuits,* vol. SC-12, pp. 693–703, Dec. 1977.

[22] P. K. Chatterjee *et al.*, "High-density dynamic MOS memory devices," *IEEE J. Solid-State Circuits,* vol. SC-14, pp. 486–497, Apr. 1979.

[23] P. Mazumder, "Testing and fault-tolerant aspects of high-density VLSI memory," Ph.D. dissertation, Univ. of Illinois, Dept. Elec. Comput. Eng., Aug. 1987.

[24] P. Mazumder and J. H. Patel, *Design and Algorithms for Parallel Testing of High-Density VLSI Memories.* Boston, MA: Kluwer Academic.

[25] S. W. Golomb, *Polyominoes.* New York: Scribner, 1965.

[26] ——, "Tiling with polyominoes," *J. Combinatorial Theory,* vol. 2, pp. 280–296, 1966.

[27] R. Nair, S. M. Thatte, and J. Abraham, "Efficient algorithms for testing semiconductor random-access memories," *IEEE Trans. Comput.,* vol. C-27, pp. 572–576, June 1978.

[28] M. Marinescu, "Simple and efficient algorithms for functional RAM testing," in *Proc. Int. Test Conf.*, Nov. 1982, pp. 236–239.
[29] A. H. Shah *et al.*, "A 4-Mbit DRAM with trench-transistor cell," *IEEE J. Solid-State Circuits*, vol. SC-21, pp. 618–627, Oct. 1986.

Pinaki Mazumder (S'84–M'87) received the B.Sc. degree in physics from the Gauhati University, India, the B.S.E.E. degree from the Indian Institute of Science, Bangalore, the M.Sc. degree in computer science from the University of Alberta, Canada, in 1985, and the Ph.D. degree in electrical and computer engineering from the University of Illinois, Urbana-Champaign, in 1987.

He worked for over six years as a Senior Design Engineer at the Bharat Electronics Ltd. (a collaborator of RCA–GE) in its integrated circuits design-and-applications laboratory. During the summers of 1985 and 1986, he worked as a member of the Technical Staff at the AT&T Bell Laboratories in the area of hardware synthesis from system-level behavioral description. Currently, he is working as an Assistant Professor in the Department of Electrical Engineering and Computer Science of the University of Michigan, Ann Arbor. His research interests include VLSI testing, computer-aided design, parallel architecture, and image processing.

Dr. Mazumder is a member of Phi Kappa Phi.

Janak H. Patel (S'73–M'76–SM'87–F'88) was born in Bhavnagar, India. He received the B.Sc. degree in physics from Gujarat University, India, the B.Tech. degree from Indian Institute of Technology, Madras, and the M.S. and Ph.D. degrees from Stanford University, Stanford, CA, all three in electrical engineering.

From 1976 to 1979 he was on the Faculty of Electrical Engineering, Purdue University, West Lafayette, IN. Since 1980 he has been with the University of Illinois, Urbana-Champaign, where he is currently a Professor of Electrical and Computer Engineering, and Computer Science, and a Research Professor with the Coordinated Science Laboratory. He is currently engaged in research, teaching, and consulting in the areas of computer architecture, testing, and fault tolerance of VLSI systems.

Dr. Patel is a member of the Association of Computing Machinery.