Methodologies for Testing Embedded Content Addressable Memories

PINAKI MAZUMDER, JANAK H. PATEL, MEMBER, IEEE, AND W. KENT FUCHS, MEMBER, IEEE

Abstract—**This paper presents a design strategy for efficient and comprehensive testing of *embedded* dynamic content addressable memory (DCAM) where the address, data, and hit lines are not externally controllable or observable. Based on the design for testability approach, three algorithms are developed for testing common functional faults in CAM's. The first algorithm provides a new method of detecting pattern, sensitive faults over a neighborhood size of nine and thereby tests a w-word CAM in $33w + 2b + 64$ operations, where b is the number of bits in a word. The proposed algorithm is significantly more efficient than other embedded procedures for testing pattern-sensitive faults. The embedded CAM can be tested for pattern-sensitive faults by using an extra $2w + 25$ transistors without the need for a signature analyzer. Two additional simple algorithms are given in the Appendix for testing embedded CAM's for stuck-at and adjacent-cell coupling faults.**

I. Introduction

AN INCREASING number of functional devices within VLSI chips are becoming inaccessible to external testers. As a result, new testable design strategies are necessary to grapple with the testing difficulties in embedded applications. Memory devices are extensively used in VLSI applications, and testing of embedded random access memory (RAM) devices has been studied by several researchers [1]–[4]. However, as is described in this paper, these proposals for testing embedded RAM's are not appropriate for *content addressable memory* (CAM). This paper examines the problem of testing CAM's in an *embedded* environment, where the address and control lines of the CAM are not directly controllable and the data and hit lines are not externally observable.

The paper demonstrates that a CAM can be tested in a far simpler manner than a RAM. The reading operations in a CAM can be replaced by one content addressable search operation, and instead of using a signature analyzer, as is typically proposed for built-in self-test of RAM's [1], [3], simple hardware can be used to monitor the match lines in a CAM. In contrast to the signature analyzer approach, simple testable designs are proposed in this paper which require at most $2w + 25$ transistors in a w-word CAM of any arbitrary word size. The paper proposes three methodologies to test for stuck-at faults, adjacent cell coupling faults, and pattern-sensitive faults by using simple testable hardware. The contributions of the paper include: i) a simple testable design for CAM's, ii) algorithms for testing for three types of functional faults, and iii) an approach to testing CAM's utilizing a single associative search operation as opposed to the conventional w read or associative search operations [5], [6].

Section II of the paper introduces the dynamic CAM (DCAM) design illustrating its different modes of operation. This is followed by the design for testability technique which can be used to test the embedded CAM's by three simple test procedures. The new algorithm for testing embedded CAM's is presented in Section III, followed by a built-in self-test (BIST) implementation of the algorithm in Section IV.

Manuscript received June 1, 1987; revised August 27, 1987. This work was supported in part by the Semiconductor Research Corporation under Contract 86-12-109. An earlier version of this paper was presented at the 17th International Fault-Tolerant Computing Symposium, 1987. The review of this paper was arranged by Associate Editors, E. J. McCluskey and S. Mourad.

The authors are with the Computer Systems Group, Coordinated Science Laboratory, University of Illinois, Urbana, IL 61801.

IEEE Log Number 8717437.

II. Testable CAM Design

A. CAM Overview

A typical organization for content addressable memory is shown in Fig. 1. A b-bit, w-word CAM is organized as an $n = b \times w$ two-dimensional array where all word lines (rows) and bit lines (columns) can be accessed randomly and in parallel. Each memory cell is connected to one pair of horizontal lines (comprising a row) and one pair of vertical lines (comprising a column). A memory cell is denoted by C_{ij} if it is located at the cross point of an ith bit line and a jth word line and is connected to the horizontal lines W_j, V_j, and to the vertical lines D_i, $\overline{D}_i$. Lines D_i and $\overline{D}_i$ are the ith BIT and $\overline{\text{BIT}}$ lines driven by the sense amplifiers. Line W_j is the jth word line and is selected by the word line decoder, which decodes the content of the $\log_2 w$-bit memory address register (MAR). Line V_j is derived by the control cell (CC_j) and its value depends on the logical value of W_j and the $R/\overline{W}$, called the read/write line.

The basic memory cell, shown in Fig. 2, consists of four transistors [7]. Transistors Q_1 and Q_2 form the left half-cell, which is somewhat similar to a switched capacitor dynamic RAM cell. As in DRAM, the logic state is stored by the parasitic capacitance at the gate of Q_1. The right half-cell consists of transistors Q_3 and Q_4 and is identical to the left half-cell. In contrast to DRAM, both halves are necessary to constitute a single CAM cell. Otherwise, as will be evident subsequently, the search operation cannot be done with a half-cell. The left half-cell

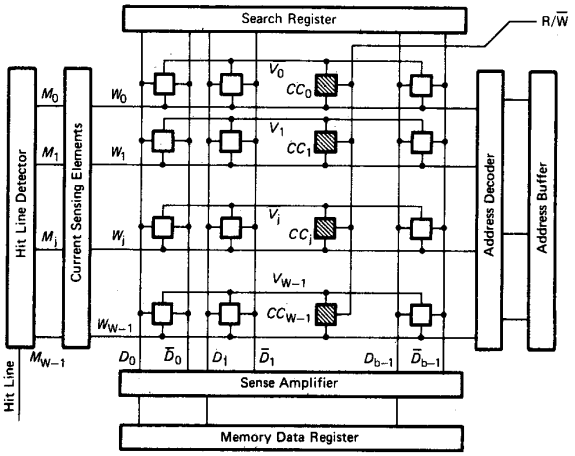


Fig. 1. Basic organization of a dynamic CAM.

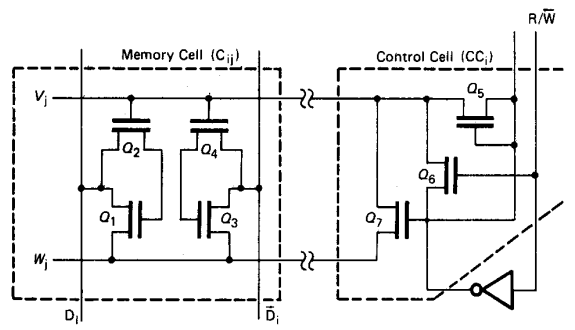


Fig. 2. Dynamic CAM cell.

and the right half-cell contain the complementary data. The cell C_{ij} is said to store a logical 0 if the parasitic capacitance of Q_1 is discharged and the parasitic capacitance of Q_3 is charged. A logical 1 is stored if the parasitic capacitance of Q_1 is charged and that of Q_3 is discharged. In order to replenish the charges due to leakage, periodic refreshing is necessary; hence the memory is called dynamic content addressable memory (DCAM). Line V_j in C_{ij} is supplied by the control cell, CC_j, which consists of three transistors, Q_5, Q_6, and Q_7. These three transistors realize the Boolean function $V_j = \overline{R}.(W_j + R)$, so that transistors Q_2 and Q_4 can be turned on only in write mode when $R = 0$. Depending on the values of R, D_i, $\overline{D}_i$, and W_j, the memory cell operates in the modes shown in Table I. In the following, the five different modes of DCAM operation are explained.

1) *Write Mode:* In the write mode, $W_j = 1$ and $R = 0$. Thus $V_j = 1$ and transistors Q_2 and Q_4 are turned on so the values of D_i and $\overline{D}_i$ will be stored in the parasitic gate capacitances.
2) *Masked Write Modes:* If $W_j = 0$ with $R = 0$, then $V_j = 0$ and transistors Q_2 and Q_4 do not turn on and no write operation can be made on the cell. Also if $W_j = 1$, $R = 0$, and both D_i and $\overline{D}_i$ are at high impedance, the content of memory will remain unchanged.

TABLE I
TRUTH TABLE FOR CAM

Function	Input				Output
	D_i	$\overline{D}_i$	W_j	R	
Write 0	0	1	1	0	
Write 1	1	0	1	0	
Masked Write	x	x	0	0	
Masked Write	z	z	1	0	
Read 0	1	1	0	1	Current in $\overline{D}_i$
Read 1	1	1	0	1	Current in D_i
Masked Read	1	1	1	1	
Match 0	0	1	1	1	Current in W_j
Match 1	1	0	1	1	if Mismatch occurs
Masked Match	1	1	1	1	

x: either 0 or 1; z: high impedance state.

3) *Read Mode:* In the read mode, $W_j = 0$ and $R = 1$. The bit lines D_i and $\overline{D}_i$ are precharged to 1. Depending on the content of the cell, either Q_1 or Q_3 (but not both) will turn on and either D_i or $\overline{D}_i$ will be discharged by the word line. Thus the sense amplifier can read a 0 or a 1.
4) *Content-Addressable Search:* In this mode, $W = 1$ and $R = 1$. The data in the search register drive the D_j and $\overline{D}_j$ lines, and these values are compared with the stored content of the cell. If the left half-cell is in state 0(1) and right half-cell is in state 1(0), then the memory cell C_{ij} will be said to be in state 1(0). If the values of the $D_i = 1(0)$, $\overline{D}_i = 0(1)$ and the memory cell is in state 0(1), then $Q_3(Q_1)$ will remain turned off and $Q_1(Q_3)$ will be turned on but no current will flow through the word line W_j since both W_j and $D_i(\overline{D}_i)$ are at value 1. This will detect a match. In case of a mismatch, current flows from the W_j through Q_1 if $D_i = 0$ or through Q_3 if $\overline{D}_i = 0$, respectively. This current is detected by the current sensing elements, which pull up the corresponding MATCH line M_j to high value. The HIT LINE detector in Fig. 1 is a logic NAND of all MATCH lines and if any mismatch occurs HIT LINE goes high.
5) *Masked Search:* A memory cell can be masked in search mode if both the D_i and $\overline{D}_i$ are set to 1. In this case, even if Q_1 or Q_3 turns on due to a mismatch between the state of the cell and D_i or $\overline{D}_i$, respectively, no current will flow through the word line W_j since both the bit lines and word line are at value 1.

In addition to the above modes, an *internal masking* of cells occurs if both the cells contain a 0 during a read operation [7]. Here, no current will flow through D_i and $\overline{D}_i$, and the reading will be masked.

The operation of writing a 0 or a 1 on a memory cell will be denoted by, respectively, W_0 and W_1. The reading operation will be denoted by R. If the state of the memory cell does not change, then the write operation is called nontransition write. On the contrary, if the state of the

cell changes due to writing 1(0), it is called transition write 1(0) and is denoted by ↑(↓). In the search operation, all the cells in the selected word lines can be compared with a bit pattern in the search register. The content of the search register is denoted by a regular expression, e.g., if the content of the search register is ⟨010101 · · ·⟩, it is denoted by SR = (01)*.

B. Design for Testability

The testable design proposed here exploits the fact that test procedures often generate a test pattern which repeats after each p-word line by doing mutual comparison of every p word lines. Thus, the proposed technique replaces the w read/associative search operations by a single associative search. Giles and Hunter [5] have demonstrated how to modify the HIT LINE detector to augment the testability of stuck-at faults in a CAM. In this paper a similar approach has been adopted to test a large class of functional faults, and the odd word lines and the even word lines have been grouped into two different classes (i.e., $p = 2$). The HIT LINE detector in Fig. 1 has been modified and the testable hardware in Fig. 3 has been designed to detect whether all the lines in a class (i.e., all even lines and all odd lines) either hit or miss simultaneously. Transistors $Q_0, \cdots, Q_{m-1}$, $(m = b/2)$, detect the simultaneous mismatch in all even word lines, while transistors $Q_m, \cdots, Q_{2m-1}$ detect the simultaneous mismatch of all odd lines. Transistors $P_0, \cdots, P_{m-1}$ detect the simultaneous match in all even word lines, while transistors $P_m, \cdots, P_{2m-1}$ detect the simultaneous match of all odd lines. Transistors $C_0, \cdots, C_3$ are comparator transistors and $A = 0$ if all the even lines either match or mismatch simultaneously, while $D = 0$ if all the odd lines either match or mismatch simultaneously. $A = 1$ if some of the even lines match and some mismatch. Similarly $D = 1$ if some of the odd lines match and some mismatch. If either $A = 1$ or $D = 1$, error latch triggers and ERROR = 1. If $A = 0$ and $D = 0$, then ERROR = 0. HIT LINE indicates whether any word line matches and is a Boolean NAND of B and C. Clearly $B = 0$ if at least one even line matches and $C = 0$ if at least one odd line matches. Transistors $T_0, \cdots, T_7$ are used for precharging by precharge clock ϕ_P.

The remainder of the paper demonstrates how, by using the testable hardware, pattern-sensitive faults in a CAM can be detected. Test procedures for simple *stuck-at* and *bridging* faults are given in the Appendix. The procedures in the Appendix are modifications of those proposed by Jain and Stroud [1] for testing embedded RAM's. In the following section a new algorithm is proposed for built-in self-test of the pattern-sensitive faults in DCAM's.

III. Pattern-Sensitive Faults in Embedded Dynamic CAM

A pattern-sensitive fault models the adjacency effect between a memory cell, called the 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 [8], [9]. If a write operation is faulty due to the presence of a particular bit pattern in the neighboring cells, then the fault is called a static pattern-sensitive fault (SPSF). On the other hand, if the content of the base cell changes due to a WRITE operation in one of the neighboring cells and other neighboring cells hold a fixed pattern of 0's and 1's, then the fault is called a dynamic pattern-sensitive fault (DPSF). Since any memory cell can be arbitrarily masked in a CAM, both SPSF's and DPSF's may occur. The problem of detecting SPSF's and DPSF's over a neighborhood of nine cells is addressed in this paper. Such a neighborhood is shown in Fig. 4 and is commonly called a 9-neighborhood. C_{ij} is the base cell. Cells $C_{i\pm1,j}$ share the same word line (line j in Fig. 1) with the base cell and are called its word line neighbors (N_w). Cells $C_{i,j\pm1}$ share the same bit line (line i in Fig. 1) with the base cell and are called the bit line neighbors (N_b) of the base cell. The four corner cells in Fig. 3 are diagonally adjacent to the base cell and are called its diagonal neighbors (N_d).

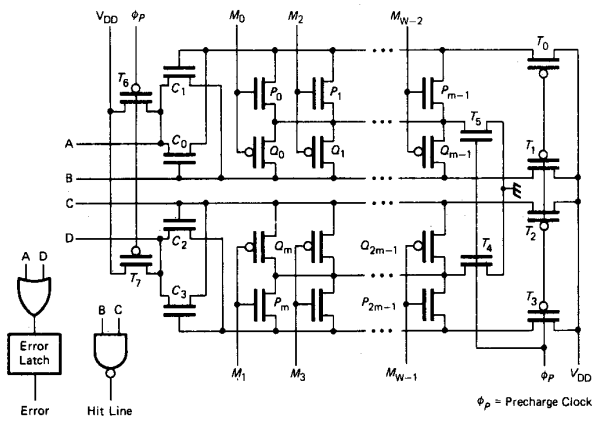


Fig. 3. Testable CAM circuit.

In order to detect all the static pattern-sensitive faults in the memory, 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 necessary for each cell in the memory to detect the SPSF. In order to detect all the dynamic pattern-sensitive faults, each base cell should be tested by an ordinary read or associative read 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 paper, an approach has been adopted in which cells in 9-neighborhood are categorized into four logical groups, *viz.*, the base cell, bit line neighbors, word line neighbors, and diagonal neighbors. It may be noted that in practice the faults occur due to leakages between the

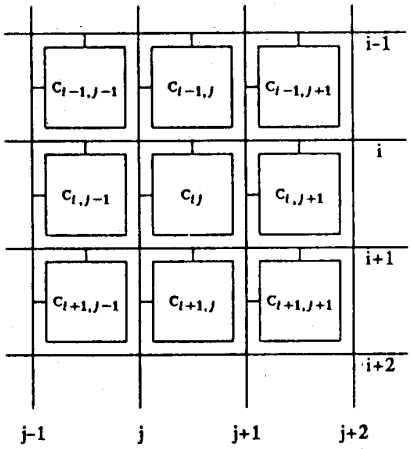


Fig. 4. Base cell C_{ij} and its 9-neighborhood.

TABLE II
ALL POSSIBLE SPSF'S AND DPSF'S

fault type	fault notation
SPSF	↑/000, ↑/100, ↑/010, ↑/110, ↑/001, ↑/101, ↑/011, ↑/111, ↓/000, ↓/100, ↓/010, ↓/110, ↓/001, ↓/101, ↓/011, ↓/111.
DPSF	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↓.

cells within the neighborhood. It has been found that the leakage is maximum when the symmetrically located cells contain the same bit patterns [8]. 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 even when $C_{i+1,j}$ is at 0(1) and $C_{i-1,j}$ is at 1(0).

Definition 1: A pattern-sensitive fault is called static (SPSF) if a transition write $W_\tau(C_{ij})$ on the base cell in N, where W_τ is either ↑ or ↓, is faulty in the presence of a fixed pattern in $N_b \cup N_w \cup N_d$. This fault is denoted by $W_\tau(C_{ij})/s(N_b)\ s(N_w)\ s(N_d)$, where $s(N_b)$, $s(N_w)$, and $s(N_d)$ represent the states of all cells in N_b, N_w, and N_d, respectively.

Definition 2: A pattern-sensitive fault is called dynamic (DPSF) if the state of base cell $s(C_{ij})$ changes because of a transition write W_τ on one or more cells in $N_b \cup N_w \cup N_d$. A DPSF is denoted by $s(C_{ij})/W_\tau(N_b)\ s(N_w)\ s(N_d)$, $s(C_{ij})/s(N_b)\ W_\tau(N_w)\ s(N_d)$, or $s(C_{ij})/s(N_b)\ s(N_w)\ W_\tau(N_d)$, depending on whether the transition write made on the cells of N_b, N_w, or N_d changes $s(C_{ij})$.

Clearly, $s(N') = 1$ if all the cells in the neighborhood $N' \in \{N_b, N_w, N_d\}$ are 1 and $s(N') = 0$ if all the cells in N' are 0. It will be assumed that during testing of a pattern-sensitive fault, the contents of all the cells in N' are always identical. Since all the cells in N' may not be written in parallel, a read operation to test a fault is only made when all the cells in N' are written identically. If all the cells in N' make a transition write ↑, then $W_\tau(N')$ will be denoted by ↑. On the contrary, if all the transition writes in N' are ↓, then $W_\tau(N')$ will be denoted by ↓. In order to detect the dynamic pattern-sensitive fault in N, it will be assumed that the cells in N' are mutually consistent in the sense that if an operation W_τ is applied to one cell or simultaneously to more than one cell belonging to N', then the next operation in the test will be to apply W_τ to the remaining cells in N' and the state transition in base cell occurs only once.

Definition 3: If the base cell i is in state $s_i \in \{0, 1\}$ and a transition write W_τ is made over one or two cells simultaneously belonging to N', then a DPSF is said to be consistent if the state of cell i does not change back from $\overline{s_i}$ to s_i after the successive W_τ operation on the remaining cells in N'.

Utilizing this notation, the SPSF's and DPSF's can be represented by Table II. The four tuples are as described in Definitions 1 and 2. Since, there are 16 SPSF's and 48 DPSF's associated with each cell in the memory, it can be easily seen that at most $4 \times 2^4 = 64$ transition writes on each cell will be necessary to sensitize all the SPSF's and DPSF's in a CAM. In this section, it will be shown that by cleverly combining the transition write sequence over the neighborhood, the total number of transition writes on each cell can be reduced to 16.

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. 5. 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 sensitize four pattern-sensitive faults; 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

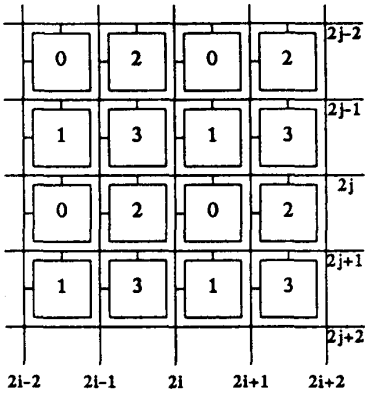


Fig. 5. Cell type assignment.

TABLE III
CONTENT OF MEMORY CELLS

Op #	$s_3s_1s_2s_0$	Op #	$s_3s_1s_2s_0$	Op #	$s_3s_1s_2s_0$	Op #	$s_3s_1s_2s_0$
1	0001	17	0001	33	0001	49	0001
2	0011	18	0000	34	1001	50	0101
3	0010	19	1000	35	1101	51	0111
4	0110	20	1010	36	1100	52	1111
5	0111	21	0010	37	1000	53	1101
6	0101	22	0011	38	0000	54	1001
7	0100	23	1011	39	0100	55	1011
8	1100	24	1111	40	0110	56	1010
9	1101	25	0111	41	0010	57	1000
10	1111	26	0110	42	1010	58	1100
11	1110	27	1110	43	1110	59	1110
12	1010	28	1100	44	1111	60	0110
13	1011	29	0100	45	1011	61	0100
14	1001	30	0101	46	0011	62	0000
15	1000	31	1101	47	0111	63	0010
16	1001	32	0101	48	0011	64	0000

transition write sequences per cell, a graph theoretic approach will be used. The 4-tuple in Table III describes a state space of 16 nodes (states) where each node represents the binary pattern in the 9-neighborhood. These nodes are numbered 0 to 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 graph describes a four-dimensional cube as shown in Fig. 6. Each undirected edge in Fig. 6 represents two antiparallel directed edges, and 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. The set of thick edges corresponds to changing the state of the base cell and will pertain to sensitizing the SPSF's. Other edges pertain to sensitizing the DPSF's. A minimal number of write sequences which will traverse all the directed edges in the 4-cube will constitute an Eulerian tour. The problem of deriving an Eulerian tour over an n-cube can be performed in several ways, such as the recursive technique [10], [11], and from

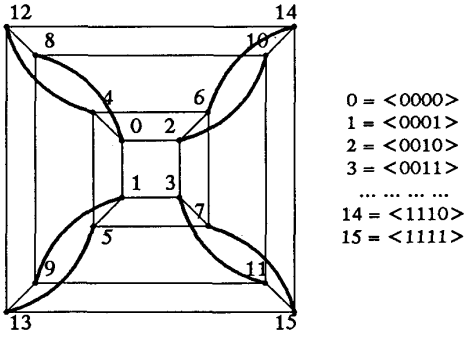


Fig. 6. State space graph for neighborhood with $k = 4$.

reflected gray code [12]. Actually, the technique of obtaining an Eulerian tour from the reflected gray code can be extended to any arbitrary gray code to generate any arbitrary Eulerian path. It can be easily seen that the sequence of operations shown in Table III represents an Eulerian tour over the 4-cube. Algorithm 1 uses these write sequences to test the SPSF's and DPSF's for every cell in the memory over its 9-neighborhood.

Theorem 1: Algorithm 1 detects all modeled static pattern-sensitive faults (SPSF) and dynamic pattern-sensitive faults (DPSF) in a memory array. It also detects all stuck-at faults in the comparator and error detector logic.

Proof: From the cell assignments, the neighborhood relationships between cells having different numbers are shown in Table IV. In Algorithm 1, 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. In Table III 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 SPSF's are performed in Algorithm 1. Since after each transition write, an associative search is done, if any SPSF occurs, current will flow through the WORD (MATCH) line, which in turn will be detected by the parallel comparator and error latch. Also, because of the neighborhood relationship in Table IV, every transition write on cells numbered k will also sensitize the DPSFs for the other three cells for which the number is not k. For example, in operation #5 of Table III, the state of the cells numbered 0 changed from 0 to 1 while the contents of the cells numbered 1, 2, and 3 remain the same to 1, 1, and 0, respectively. The succeeding associative search operation in line (8) of Algorithm 1 detects an SPSF in all neighborhoods for which the base cell is 0 and DPSF's in all other neighborhoods for which base cell is not 0 and the fault occurs due to transition in cell numbered 0. The effect of operation #5 on different neighborhoods is shown in Fig. 7. Thus Algorithm 1, which writes all the patterns in Table III over the entire memory, will sensitize both the SPSF's and DPSF's for every cell in the memory.

Since at lines (8) and (10) of Algorithm 1, the contents

Algorithm 1: Test of Pattern Sensitive Faults in Embedded CAM

(1) Initialize all the cells to 0.

(2) Set $k = 0$.

(3) Set the k-th bit in SR to 1 and rest of the bits to 0. Associatively read the memory. Check if ERROR=0, else fault is detected. Complement the k-th bit in SR.

(4) Set $k = k+1$.

(5) If $k < b$, go to (3); else go to (6).

(6) Set $i = 0$.

(7) If the transition write in i-th operation is on cells numbered $p \in \{0,2\}$, write on all cells having number p and are on even word lines.

(8) Set SR=$(s_0 s_2)^*$, where s_0 and s_2 are the contents of cells numbered 0 and 2, respectively. Associatively read the memory. Check if ERROR=0, else error is detected.

(9) If the transition write in i-th operation is on cells numbered $p \in \{1,3\}$, write on all cells having number p and are on odd word lines.

(10) Set SR=$(s_1 s_3)^*$, where s_1 and s_3 are the contents of cells numbered 1 and 3, respectively. Associatively read the memory. Check if ERROR=0, else error is detected.

(11) Set $i = i+1$.

(12) If $i = 10$, then go to (13); else go to (17).

(13) Set $k = 0$.

(14) Set the k-th bit in SR to 0 and rest of the bits to 0. Associative read the memory. Check if ERROR=0, else fault is detected. Complement the k-th bit in SR.

(15) Set $k = k+1$.

(16) If $k < b$, go to (14); else go to (17).

(17) If $i < 65$, go to (7); else exit successfully.

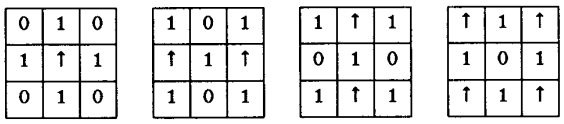


Fig. 7. Pattern written on different neighborhoods by operation 5.

TABLE IV
CELL TYPES AND THEIR NEIGHBORS

Base Cell	Bit Line Neighbors	Word Line Neighbors	Diagonal Neighbors
0	1	2	3
1	0	3	2
2	3	0	1
3	2	1	0

of the memory cells are tested by observing and comparing the MATCH lines of different memory cells, comparator logic of each memory cell will be tested simultaneously. Since Algorithm 1 tests all the patterns of Table III, the comparator logic will be tested correctly for matching operations if there is no SPSF and DPSF in any memory cell. But the mismatch in associative search operations cannot be detected by lines (8) and (10). Since the MATCH line is a wired OR of different memory cells on a single word line, each memory cell in a word line will be tested independently. In lines (2) through (5), the memory is tested with a marching pattern of 1's when all the cells of the memory are initialized to 0. Since the pattern is different in only one bit position for every word in the memory, it will sensitize faults in the comparator logic for mismatch operations, and in every word line there will be a mismatch. Similarly, in lines (13) through (16), the memory is tested by a marching pattern of 0's in the background of 1's. □

Theorem 2: Algorithm 1 tests a w word CAM in $33w + 2b + 64$ operations, where b is the number of bits in each word.

Proof: w operations are needed to initialize the memory in line (1). Lines (2) through (5) and (13) through (16) test in $2b$ operations the mismatch in an associative search for each memory cell. There are altogether 64 patterns in Table III and in each pattern there is only one transition write on a cell. In lines (7) and (9), $w/2$ word lines are written with each pattern in Table III. In lines (8) and (10), a single associative search is made to test the contents of the CAM after each pattern in Table III is written on the relevant word lines. Thus altogether $32w + 64$ operations are made to write all the 64 patterns in Table III. Hence, the overall complexity of Algorithm 1 is $33w + 2b + 64$. □

By employing the technique of simultaneous access of even and odd word lines in addition to the bit lines [5], the test speed can be further reduced to $97 + 2b$ for any arbitrary size CAM. Generally, for CAM's of moderately small size, this will not deteriorate the write cycle time.

But for relatively large size CAM's, this will require an increased write cycle time and will increase power consumption.

It may be noted that the testable hardware in Fig. 3 is only tested partially by Algorithm 1. When all the even and odd lines match, the n-type transistors $Q_0, \cdots, Q_{2m}$ will be tested for stuck-at 1 faults. When all the even and odd lines mismatch, all the p-type transistors $P_0, \cdots, P_{2m}$ will be tested for stuck-at 0 faults. The rest of the stuck-at faults in the testable hardware can be tested by the following procedure.

Test Procedure For Additional Hardware

(1) Initialize the memory with all 0's.
(2) Set (SR) = (00)* and check if ERROR = 0.
(3) FOR each word line, DO
{ Write 1's in all its bit cells.
Set (SR) = (00)* and check if ERROR = 1.
Write 0's in all its bit cells.
Set (SR) = (00)* and check if ERROR = 0.
(4) Initialize the memory with all 1's.
(5) FOR each word line, DO
{ Write 0's in all its bit cells.
Set (SR) = (11)* and check if ERROR = 1.
Write 1's in all its bit cells.
Set (SR) = (11)* and check if ERROR = 0. }

In this section only one algorithm is presented to detect the symmetric pattern-sensitive faults. The rationale for symmetric pattern-sensitive faults is to utilize the knowledge of the leakage currents in the CAM cell and to transform the neighborhood consisting of nine physically adjacent cells into a logical neighborhood of size 4. However, these leakage currents are unlikely to occur in static CAM design and, therefore, a slightly different algorithm can be used to test the pattern-sensitive faults in a static CAM. In [13] an alternative testing algorithm has been designed to test the PSF's in static CAM. The PSF's in [13] are different from those discussed in this paper in the sense that instead of testing the transition writes in the presence of all possible patterns in the neighborhood, they test whether a 0 and a 1 can be stored and held in the base cell in the presence of all valid patterns in the neighborhood. The algorithm tests a w-word CAM in $33w + 2b + 160$ steps and detects PSF's over a physical neighborhood size of 25 cells [14], [15]. This algorithm can be readily used to test the DCAM discussed in this paper. By modifying the address decoder such that all word lines having j (mod 5) same (where j is the address of a word line) are written in parallel, the complexity can be improved to $325 + 2b$ operations, where a constant 325 operations are needed to test the pattern-sensitive faults and $2b$ operations are used to test the comparator logic in the b-bit-wide CAM design.

In addition, two conventional commercial test procedures for reduced fault models can be adapted (see the Appendix) to the testable CAM without the need for signature analyzers which are commonly used for embedded applications. The overhead (i.e., the additional test logic with the associated routing) involved in the conventional techniques utilizing a signature analyzer in a dynamic RAM has been found to be between 3 percent and 5 percent of the total chip area [1]. The proposed technique needs only $2w + 25$ extra transistors in a w-bit CAM; thereby the overhead is typically less than 1 percent for a 32-bit wide w-bit CAM. It may be noted that the presence of testable hardware obviates the need for a separate HIT LINE detector and thereby the overhead is considerably reduced.

IV. BIST Implementation of Algorithm 1

The operations in Table III can be generated within the chip by using the circuit shown in Fig. 8. The reflected gray code is generated by the JK flip-flops with suitable feed back. The flip-flop equations are

$$J_A = BC\overline{D} + \overline{B}\,\overline{C}\,\overline{D} + \overline{B}CD + B\overline{C}D \quad \text{and} \quad K_A = \overline{J_A} \tag{1}$$

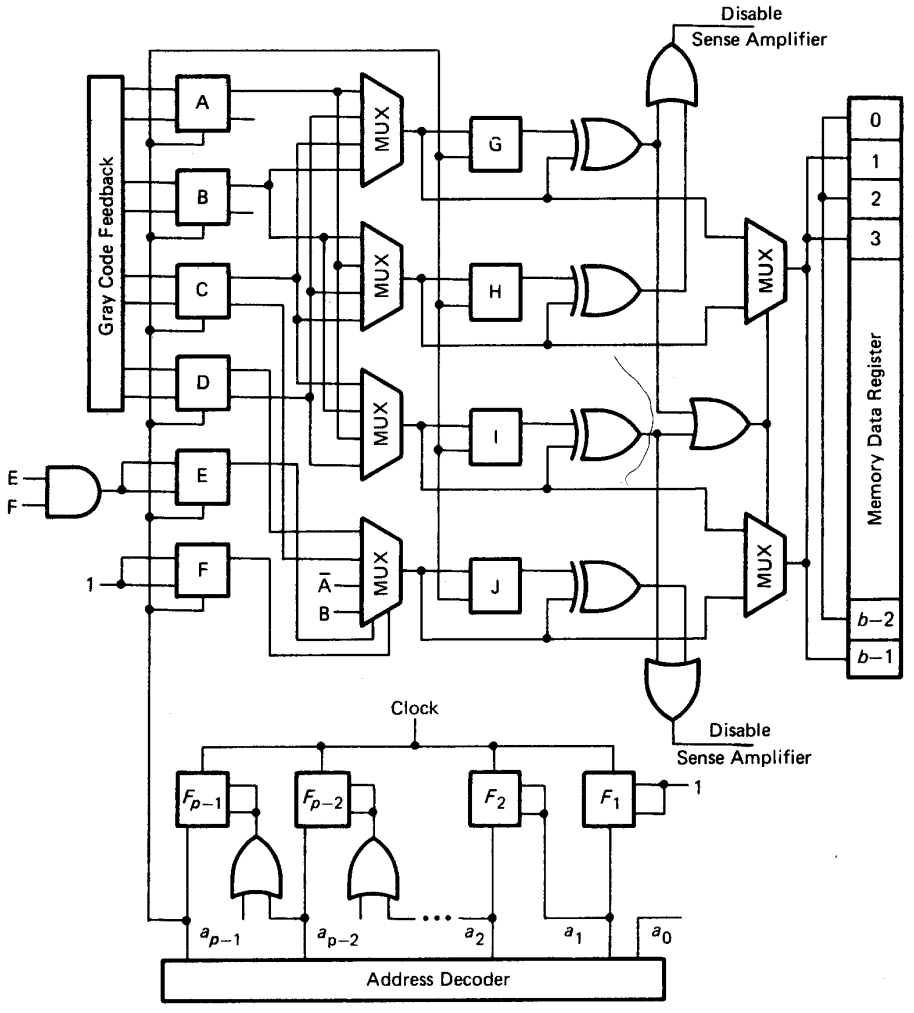
$$J_B = A\overline{C}\,\overline{D} + ACD \quad \text{and} \quad K_B = AC\overline{D} + A\overline{C}D \tag{2}$$

$$J_C = \overline{A}B\overline{D} \quad \text{and} \quad K_C = \overline{A}BC \tag{3}$$

$$J_D = \overline{A}\,\overline{B}C \quad \text{and} \quad K_D = \overline{A}\,\overline{B}\,\overline{C}. \tag{4}$$

The flip-flops with the feedback connections generate the reflected gray code corresponding to the 16 operations (#1–#15, #64) in Table III. The remainder of the operations in the Table III are obtained from these four flip-flops by selecting the suitable lines (complemented or noncomplemented output of flip-flops) and by reordering them. This is achieved by the set of multiplexers shown in Fig. 8. Nontransition writes are disallowed by disabling the corresponding sense amplifiers, and the associated bit lines are set to high impedance. In order to write on the even or odd word lines a $(\log_2 w - 1)$-bit synchronous counter has been used and it is connected to the address lines of the word line decoder. The LSB of the address line is connected to 0 if the even word lines are to be accessed, or to 1 if the odd word lines are to be selected.

The BIST generator circuit can be further simplified by modifying Algorithm 1 slightly as described here. In Algorithm 1, the sequence of transition writes was generated by describing an Eulerian tour over the symmetric 4-cube. The hardware overhead in Fig. 8 can be reduced over a factor of 2 by decomposing the Eulerian tour into eight disjoint Hamiltonian cyclic tours: ⟨0, 1, 9, 13, 15, 14, 6, 2, 0⟩, ⟨0, 2, 6, 14, 15, 13, 9, 1, 0⟩, ⟨0, 8, 9, 11, 15, 7, 6, 4, 0⟩, ⟨0, 4, 6, 7, 15, 11, 9, 8, 0⟩, ⟨12, 4, 5, 7, 3, 11, 10, 8, 12⟩, ⟨12, 8, 10, 11, 3, 7, 5, 4, 12⟩, ⟨12, 13, 5, 1, 3, 2, 10, 14, 12⟩, and ⟨12, 14, 10, 2, 3, 1, 5, 13, 12⟩. The Hamiltonian cycles were described over the subgraph of the symmetric 4-cube. The length of each Hamilton cycle is 8. Altogether 8 disjoint Hamiltonian tours are made. Thus all the 64 edges of the symmetric



A–F, $F_1 \cdots F_{p-1}$ are J–K Flip Flops

G–J are D Flip Flops

Fig. 8. BIST implementation of the circuit.

4-cube are traversed and in this way all the 16 SPSF's and 48 DPSF'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 reinitialized such that it contains a column bar pattern of 0's and 1's. This needs an additional w write operations, which for a 1K byte memory having a memory cycle time of 50 ns will take less than an extra 50 μs. But the economy in hardware is considerable [14] and is shown in Table V.

TABLE V
TEST SIZE OPTIMALITY VERSUS BIST HARDWARE

Component	Hamiltonian Tour	Eulerian Tour	Extra Component
	$34w+2b+64$	$33w+2b+64$	
Flipflop	$7+p$	$10+p$	3
MUX (4 to 1)	0	4	4
MUX (2 to 1)	1	2	1
ExOr Gate	3	4	1
Or Gate	1	8	7
And Gate	3	15	12
Inverter	1	1	0

V. FINAL REMARKS

This paper proposes a testing strategy for *embedded* DCAM's. The strategy uses minimal extra hardware and can fit within the interceller pitch width of a DCAM. Two conventional commercial test procedures are easily adapted (Appendix) to the testable CAM without the need for the signature analyzers which are commonly used for embedded applications. Another advantage of the proposed strategy is that all the memory cells are associatively compared and tested by a single operation. This is in contrast to the w read operations typically required to test the content of a w-word CAM. Algorithm 1 is a test

procedure which detects SPSF's and DPSF's over a 9-neighborhood which has been grouped into four classes. On the basis of physical reasoning and practical observations, nine cells in a neighborhood have been grouped into four classes, and the test Algorithm 1 tests the SPSF's and DPSF's in only $33w + 2b + 64$ operations and in $97 + 2b$ operations if even and odd word lines are simultaneously accessed.

APPENDIX

Algorithm 2: Stuck-at Faults Test in Embedded CAM

FOR $i = 0$ to $w-1$ DO
{ Write on word i with the binary value of i.}

(1) FOR $i = 0$ to $w-1$ DO
{ Search the memory with the binary value of i and check if ERROR=1.}

(2) FOR $i = 0$ to $w-1$ DO
{ Write on word i with the complement of binary value of i.}

(3) FOR $i = 0$ to $w-1$ DO
{ Search the memory with the complement of binary value of i and check if ERROR=1.}

(4) FOR $i = w-1$ down to 0 DO
{ Search the memory with the complement of binary value of i and check if ERROR=1.}

(5) FOR $i = w-1$ down to 0 DO
{ Write on word i with the complement of binary value of i.}

(6) FOR $i = 0$ to $w-1$ DO
{ Search the memory with the complement of binary value of i and check if ERROR=1.}

(7) FOR $i = w-1$ down to 0 DO
{ Search the memory with the complement of binary value of i and check if ERROR=1.}

(8) Initialize all the cells to 0.

(9) Set $k = 0$.

(10) Set the k-th bit in SR to 1 and rest of the bits to 0. Associative read the memory. Check if ERROR=0, else fault is detected. Complement the k-th bit in SR.

(11) Set $k = k+1$.

(12) If $k < b$, go to (11); else go to (14).

(13) Initialize all the cells to 1.

(14) Set $k = 0$.

(15) Set the k-th bit in SR to 0 and rest of the bits to 1. Associative read the memory. Check if ERROR=0, else fault is detected. Complement the k-th bit in SR.

(16) Set $k = k+1$.

(17) If $k < b$, go to (14); else exit successfully.

Algorithm 2 takes $10w+2b$ steps to test the stuck-at faults in the CAM cells and decoder circuitry.

Algorithm 3: Checker Board Test in Embedded CAM

(1) Write a pattern of $(01)^*$ on all even word lines.

(2) Write a pattern of $(10)^*$ on all odd word lines.

(3) Set SR=$(10)^*$ and do an associative search. Check if ERROR=0.

(4) Set SR=$(01)^*$ and do an associative search. Check if ERROR=0.

(5) Write a pattern of $(10)^*$ on all even word lines.

(6) Write a pattern of $(01)^*$ on all odd word lines.

(7) Set SR=$(10)^*$ and do an associative search. Check if ERROR=0.

(8) Set SR=$(01)^*$ and do an associative search. Check if ERROR=0.

(9) Initialize all the cells to 0.

(10) Set $k = 0$.

(11) Set the k-th bit in SR to 1 and the remaining bits to 0. Associatively read the memory. Check if ERROR=0, else fault is detected. Complement the k-th bit in SR.

(12) Set $k = k+1$.

(13) If $k < b$, go to (11); else go to (14).

(14) Initialize all the cells to 1.

(15) Set $k = 0$.

(16) Set the k-th bit in SR to 0 and the remaining bits to 1. Associatively read the memory. Check if ERROR=0, else fault is detected. Complement the k-th bit in SR.

(17) Set $k = k+1$.

(18) If $k < b$, go to (14); else exit successfully.

Algorithm 3 takes $4w+2b+4$ operations to test the stuck-at faults and the adjacent cells bridging faults.

REFERENCES

[1] S. Jain and C. Stroud, "Built-in self testing of embedded memories," *IEEE Design and Test of Computers*, pp. 27-37, Oct. 1986.

[2] Z. Sun and L. T. Wang, "Self-testing of embedded RAMs," in *Proc. Int. Test Conf.*, 1984, pp. 148-156.

[3] M. Nicolaidis, "An efficient built-in self test scheme for functional test of embedded RAMs," in *Proc. Int. Test Conf.*, 1985, pp. 118-123.

[4] W. H. McAnney, P. H. Bardell, and V. P. Gupta, "Random testing for stuck-at storage cells in an embedded memory," in *Proc. Int. Test Conf.*, 1984, pp. 157-166.

[5] G. Giles and C. Hunter, "A methodology for testing content addressable memories," in *Proc. Int. Test Conf.*, 1985, pp. 471-474.

[6] K. E. Grosspietsch, H. Huber, and A. Mueller, "The concept of a fault-tolerant and easily-testable associative memory," in *Proc. 16th Fault-Tolerant Computing Symp.*, July 1986, pp. 34-39.

[7] J. L. Mundy, J. F. Burgess, R. E. Joynson, and C. Neugebauer, "Low-cost associative memory," *IEEE J. Solid-State Circuits*, vol. SC-7, pp. 364-369, 1972.

[8] 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.

[9] P. K. Chatterjee, G. W. Tayor, A. F. Tasch, and H. S. Fu, "High-density dynamic MOS memory devices," *IEEE J. Solid-State Circuits*, vol. SC-14, pp. 486-497, Apr. 1979.

[10] 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.

[11] C. L. Liu, *Introduction to Combinatorial Mathematics*. New York: Mc-Graw Hill, 1968.

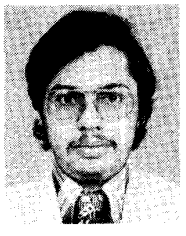
[12] J. P. Hayes, "Testing memories for single-cell pattern-sensitive faults," *IEEE Trans. Comput.*, vol. C-29, pp. 249-254, Mar. 1980.

[13] 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.

[14] P. Mazumder, "Testing and fault-tolerant aspects of high-density VLSI memory," Ph.D. thesis, Department of Electrical and Computer Engineering, University of Illinois, Aug. 1987.

[15] P. Mazumdar, "Planar decomposition for quadtree data structure," *Computer Vision, Graphics, and Image Processing*, vol. 38, pp. 258-274, June 1987.

*



Pinaki Mazumder 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, and the Ph.D. degree in electrical and computer engineering from the University of Illinois.

Presently he is working as an Assistant Professor in the Department of Electrical Engineering and Computer Science of the University of Michigan, Ann Arbor. Prior to this he worked two years as a research assistant at the Coordinated Science Laboratory, University of Illinois, and more than six years at Bharat Electronics Ltd. (a collaborator of RCA-GE) in the area of integrated circuit design and applications. During the summers of 1985 and 1986, he worked as a Member of Technical Staff at the AT&T Bell Laboratories, Indian Hill, Naperville, in the area of hardware synthesis from system-level behavioral descriptions. His research interests include VLSI testing, computer aided design, parallel architecture, and image processing.

*

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

From 1976 to 1979 he was on the faculty of Electrical Engineering at Purdue University, West Lafayette, IN. Since 1980 he has been with the University of Illinois at 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.

*

W. Kent Fuchs (S'83-M'85) received the B.S.E. degree in electrical engineering from Duke University in 1977 and the M.S. degree in electrical engineering from the University of Illinois, Urbana, in 1982. In 1984 he received the M.Div. degree from Trinity Evangelical Divinity School in Deerfield, IL, and in 1985 he received the Ph.D. degree in electrical engineering from the University of Illinois.

He is currently an Assistant Professor in the Departments of Electrical and Computer Engineering, Computer Science, and the Coordinated Science Laboratory, University of Illinois. His research interests include all aspects of VLSI system design, with emphasis on reliable computing.

Recent awards received by Dr. Fuchs include the 1987 Xerox Faculty Research Award, College of Engineering, University of Illinois; the 1986 Digital Equipment Corporation Incentives for Excellence Faculty Award; the Best Paper Award, IEEE/ACM Design Automation Conference (DAC) 1986, simulation and test category; and nomination for the best paper award, DAC 1987, simulation and test category. Current funding agencies for his research include the Semiconductor Research Corporation, the Office of Naval Research, Digital Equipment Corporation, the National Aeronautics and Space Administration, the Joint Services Electronics Program, the Microelectronics and Computer Technology Corporation, Texas Instruments, and the National Science Foundation.