Testing for Input Stuck-at Faults in Content-Addressable Memories*

Piotr R. Sidorowicz   *Janusz A. Brzozowski*

Department of Computer Science
University of Waterloo
Waterloo, Ontario, N2L 3G1
Canada

Abstract

Results pertaining to testing for input stuck-at faults in a n-word by l-bit static CMOS content-addressable memory (CAM) array are presented. First, a formal approach to modeling CAM cells is developed. The basis of this approach is the mathematical framework proposed by Brzozowski and Jürgensen for testing and diagnosis of sequential machines. Next, an input stuck-at fault model for a CAM is defined and a test of length $7n + 2l + 5$ with 100% fault coverage with respect to this fault model is constructed. This test also detects all the usual cell stuck-at and transition faults. Some design-for-testability (DFT) modifications facilitating a further reduction of this test's length are also proposed. Finally, two other CAM tests are verified with respect to our fault model.

Keywords. CMOS, content-addressable, design for testability, fault modeling, stuck-at faults, testing.

1 Introduction

Content-addressable memories are word-oriented storage devices that allow instantaneous searches of the memory content for a given key word. This type of memory plays a fundamental role in caches, table-lookaside buffers,

*This research was supported by the Natural Sciences and Engineering Research Council of Canada under grant No. OGP0000871. (Report issued April 17th, 1998)

ATM switches, and other ASICs where searching is a critical operation. In this report we investigate the testability properties of the CAM based on the cell shown in Figure 1(a). This CAM is commonly used in the telecommunications industry [1, 8, 12].

Testing is intrinsic to VLSI circuit production. The great complexity of modern ICs requires formal mathematical tools to design useful tests. Existing CAM testing schemes lack a common formalism for qualitative comparisons. A formal approach to modeling CAMs is presented in this report; the basis of this approach is the mathematical framework of Brzozowski and Jürgensen [2]. This framework has been successfully utilized for modeling faults in RAMs (See [3, 6, 7], for examples).

The focus of this report is the testing of CAM cells for faults in the *input stuck-at fault model.* This fault model consists of any stuck-at fault which affects input lines of the CAM cell.

The remainder of this report is organized as follows: A detailed model of the CAM cell's behavior is presented in Section 2. Section 3 describes the derivation of a more abstract formal model, suitable for high-level analysis. The fault model is presented in Section 4, where a test for a single cell CAM is given. In Section 5, 6 and 7 the test derived for a single cell is extended to n-word by 1-bit, 1-word by l-bit and n-word by l-bit CAMs, respectively. Design-for-testability modifications facilitating further reductions of the test length without affecting fault coverage are proposed in Section 8. Verification of other CAM tests is presented in Sections 9. Section 10 contains some concluding remarks.

2 Low-Level CAM Model

The circuit of the cell shown in Figure 1(a) can be divided on the basis of its functionality into a *storage section* and a *comparison section.* The storage section consists of two cross-coupled CMOS inverters, differential bit lines ($bit/\overline{bit}$) used for reading and writing data into a column of cells, and a word select line (WL) that enables these operations in a row of cells. In a quiescent state WL is driven low, the $bit/\overline{bit}$ lines are driven high and the cell stores either 0 or 1. During a 'write 1' or a 'write 0' operation, the $bit/\overline{bit}$ lines are driven to a true/complementary representation of the desired bit value. Raising and then lowering WL stores the bit in the cell. Changes of the cell's state, when writing a 1 to a cell containing a 0, for example, are a result of the dominant influence of stronger pull-down transistors. (Weaker pull-up transistors maintain quiescent states of the cell.) The 'write $\times$' operation

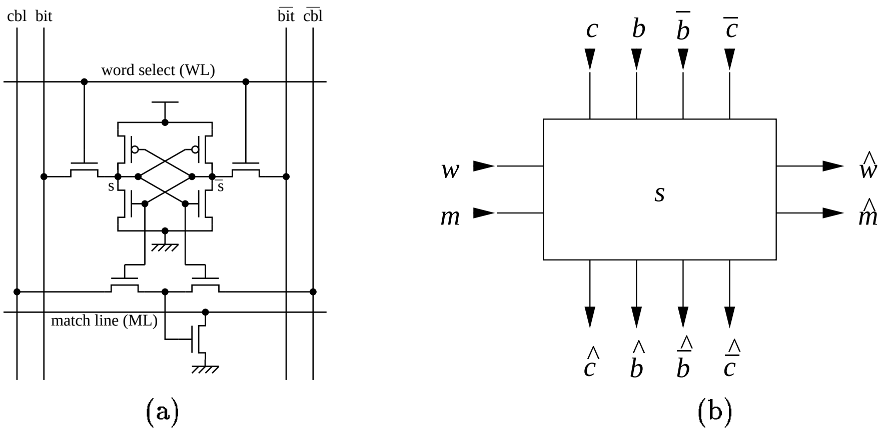
cbl bit
$\overline{\text{bit}}$ $\overline{\text{cbl}}$
word select (WL)
s
$\overline{\text{s}}$
match line (ML)
(a)
c b $\overline{b}$ $\overline{c}$
w
m
s
$\hat{w}$
$\hat{m}$
$\hat{c}$ $\hat{b}$ $\hat{\overline{b}}$ $\hat{\overline{c}}$
(b)

Figure 1: (a) Static CAM cell with dedicated compare lines, (b) its model.

can be implemented in two ways. One implementation requires both of the $bit/\overline{bit}$ lines to be driven high; proper margining of the transistors assures the preservation of the cell's state. The other implementation is exactly like a 'read' operation described below, except that any output is disregarded. We have chosen to analyze the former implementation, and draw conclusions on the latter by studying the 'read' operation itself.

A 'read' operation is performed by isolating both $bit/\overline{bit}$ lines, and then raising WL, thus causing one of the $bit/\overline{bit}$ lines to discharge. The resultant voltage differential between the $bit/\overline{bit}$ lines is detected by sense amplifiers that re-create the content of the accessed word.

The comparison section consists of the bottom three transistors of the cell's circuit, the differential compare bit lines ($cbl/\overline{cbl}$) for performing matching operations, and the match line (ML). In a quiescent state $cbl/\overline{cbl}$ are driven low and ML is driven high. The 'compare' operation is similar to the 'read' operation. First, ML is isolated. Next, $cbl/\overline{cbl}$ are driven according to the desired search key. If a mismatch occurs, the bottom-most transistor will be forced to conduct, thus discharging ML. In the case of a 'compare ×' operation, both of $cbl/\overline{cbl}$ remain low, resulting in an unconditional match. Altogether, seven operations can be performed on a CAM cell: within the storage section - 'write 0', 'write 1', 'write ×' and 'read', and within the comparison section - 'compare 0', 'compare 1' and 'compare ×'.

In order to model input stuck-at faults, we have to understand how they affect the internal operation of the cell. Since no clock signal is supplied to individual cells, a single CAM cell can be viewed as an asynchronous sequential circuit, whose behavior can be modeled by the finite state machine of Figure 1(b). Since $bit/\overline{bit}$ lines and ML are used for both input and output in the circuit, they are represented by separate variables in the model. For brevity we use b, $\bar{b}$, w, etc. for $bit/\overline{bit}$, WL, etc. Now, b, $\bar{b}$ and m are inputs and $\widehat{b}$, $\widehat{\bar{b}}$, and $\widehat{m}$ are outputs. Although WL and $cbl/\overline{cbl}$ are not usually meant to provide any output, monitoring the state of these lines, if possible, might improve the CAM's testability. For this reason, we generalize our model to include these lines; w, c and $\bar{c}$ for input, and $\widehat{w}$, $\widehat{c}$ and $\widehat{\bar{c}}$ for output.

The total state of the cell is defined by the values present on the input and output lines of the cell, and by its internal state s. It is represented by the 13-tuple:

$$C_T = (w, b, \bar{b}, c, \bar{c}, m, s, \widehat{w}, \widehat{b}, \widehat{\bar{b}}, \widehat{c}, \widehat{\bar{c}}, \widehat{m}).$$

However, it turns out that in the correct cell, as well as in the presence of input stuck-at faults, the output values are identical to those of the inputs; hence, we omit them for simplicity. The "reduced" total state is symbolically represented by

$$C = w\ b\bar{b}\ c\bar{c}\ m \cdot s,$$

where the input variables have been separated by spaces to amplify their functional separation and the symbol $\cdot$ has been inserted to separate the input variables from the internal state.

The domain of each variable in state C is the set $Y = \{0, 1, \tilde{0}, \tilde{1}, I\}$. The values 0 and 1 represent lines driven to the logic values 0 and 1 respectively, while $\tilde{0}$ and $\tilde{1}$ denote lines that were first discharged and then isolated (*floated low*) or precharged and then isolated (*floated high*). Symbol I stands for an intermediate logic value, which is caused in a CMOS circuit when both pull-up and pull-down transistors simultaneously conduct. The CAM cell has two possible initial states: $C = 0\ 11\ 00\ 1 \cdot 0$ and $C = 0\ 11\ 00\ 1 \cdot 1$.

The behavior of a cell is represented by sequences of events that take place during the seven operations. Table 1 lists events that occur during these operations. By events we mean changes in the value of the total state C. The occurrences of these events are ordered from top to bottom. Note that, for every operation, the top and bottom entries in each column are initial states.

Example: 'Write 0' operation.

> Initial state of the cell: $0\ 11\ 00\ 1 \cdot 1$. First, b is lowered: $0\ 01\ 00\ 1 \cdot 1$. Then w is raised: $1\ 01\ 00\ 1 \cdot 1$. As a result, the state s changes: $1\ 01\ 00\ 0 \cdot 0$. Next, w is lowered: $0\ 01\ 00\ 0 \cdot 0$. Finally, both b and $\bar{b}$ are raised: $0\ 11\ 00\ 0 \cdot 0$.

Table 1: Read, write and compare operations in a fault-free CAM cell.

Read operations

	($s = 0$)	($s = 1$)	
Description	$w\ b\bar{b}\ c\bar{c}\ m{\cdot}s$	$w\ b\bar{b}\ c\bar{c}\ m{\cdot}s$	
initial state	0 11 00 1·0	0 11 00 1·1	
float $bit/\overline{bit}$	0 $\tilde{1}\tilde{1}$ 00 1·0	0 $\tilde{1}\tilde{1}$ 00 1·1	
raise WL	1 $\tilde{1}\tilde{1}$ 00 1·0	1 $\tilde{1}\tilde{1}$ 00 1·1	
bit or $\overline{bit}$ discharges†	1 $0\tilde{1}$ 00 1·0	1 $\tilde{1}0$ 00 1·1	
read $bit/\overline{bit}$ & lower WL	0 $\tilde{0}\tilde{1}$ 00 1·0	0 $\tilde{1}\tilde{0}$ 00 1·1	
raise $bit/\overline{bit}$	0 11 00 1·0	0 11 00 1·1	

Write operations ($s = 0$)

	W-0	W-1	W-×
Description	$w\ b\bar{b}\ c\bar{c}\ m{\cdot}s$	$w\ b\bar{b}\ c\bar{c}\ m{\cdot}s$	$w\ b\bar{b}\ c\bar{c}\ m{\cdot}s$
initial state	0 11 00 1·0	0 11 00 1·0	0 11 00 1·0
set $bit/\overline{bit}$	0 01 00 1·0	0 10 00 1·0	0 11 00 1·0
raise WL	1 01 00 1·0	1 10 00 1·0	1 11 00 1·0
new state	1 01 00 1·0	1 10 00 1·1	1 11 00 1·0
lower WL	0 01 00 1·0	0 10 00 1·1	0 11 00 1·0
raise $bit/\overline{bit}$	0 11 00 1·0	0 11 00 1·1	0 11 00 1·0

Write operations ($s = 1$)

	W-0	W-1	W-×
Description	$w\ b\bar{b}\ c\bar{c}\ m{\cdot}s$	$w\ b\bar{b}\ c\bar{c}\ m{\cdot}s$	$w\ b\bar{b}\ c\bar{c}\ m{\cdot}s$
initial state	0 11 00 1·1	0 11 00 1·1	0 11 00 1·1
set $bit/\overline{bit}$	0 01 00 1·1	0 10 00 1·1	0 11 00 1·1
raise WL	1 01 00 1·1	1 10 00 1·1	1 11 00 1·1
new state	1 01 00 1·0	1 10 00 1·1	1 11 00 1·1
lower WL	0 01 00 1·0	0 10 00 1·1	0 11 00 1·1
raise $bit/\overline{bit}$	0 11 00 1·0	0 11 00 1·1	0 11 00 1·1

Compare operations ($s = 0$)

	C-0	C-1	C-×
Description	$w\ b\bar{b}\ c\bar{c}\ m{\cdot}s$	$w\ b\bar{b}\ c\bar{c}\ m{\cdot}s$	$w\ b\bar{b}\ c\bar{c}\ m{\cdot}s$
initial state	0 11 00 1·0	0 11 00 1·0	0 11 00 1·0
float ML	0 11 00 $\tilde{1}$·0	0 11 00 $\tilde{1}$·0	0 11 00 $\tilde{1}$·0
set $cbl/\overline{cbl}$	0 11 01 $\tilde{1}$·0	0 11 10 $\tilde{1}$·0	0 11 00 $\tilde{1}$·0
ML discharges	0 11 01 $\tilde{1}$·0	0 11 10 0·0	0 11 00 $\tilde{1}$·0
read ML & ground $cbl/\overline{cbl}$	0 11 00 $\tilde{1}$·0	0 11 00 $\tilde{0}$·0	0 11 00 $\tilde{1}$·0
raise ML	0 11 00 1·0	0 11 00 1·0	0 11 00 1·0

Compare operations ($s = 1$)

	C-0	C-1	C-×
Description	$w\ b\bar{b}\ c\bar{c}\ m{\cdot}s$	$w\ b\bar{b}\ c\bar{c}\ m{\cdot}s$	$w\ b\bar{b}\ c\bar{c}\ m{\cdot}s$
initial state	0 11 00 1·1	0 11 00 1·1	0 11 00 1·1
float ML	0 11 00 $\tilde{1}$·1	0 11 00 $\tilde{1}$·1	0 11 00 $\tilde{1}$·1
set $cbl/\overline{cbl}$	0 11 01 $\tilde{1}$·1	0 11 10 $\tilde{1}$·1	0 11 00 $\tilde{1}$·1
ML discharges	0 11 01 0·1	0 11 10 $\tilde{1}$·1	0 11 00 $\tilde{1}$·1
read ML & ground $cbl/\overline{cbl}$	0 11 00 $\tilde{0}$·1	0 11 00 $\tilde{1}$·1	0 11 00 $\tilde{1}$·1
raise ML	0 11 00 1·1	0 11 00 1·1	0 11 00 1·1

†There is a connection from bit to V_{dd} when $s = 0$ and from $\overline{bit}$ to V_{dd} when $s = 1$. But this connection is through a weak p-transistor and an n-transistor, and drivers for the $bit/\overline{bit}$ are not used. Hence, we still represent these cases as floating values.

It should be noted that the analyses presented in Table 1, are done under the assumption that operations occur one at a time. However, 'compare' operations utilize a separate set of differential lines and thus some concurrent executions are feasible. For example, a 'compare' operation can be performed in parallel with a 'read' operation, but it cannot be performed until a 'write' operation is completed. Modeling concurrent operations is an open research topic.

We recognize that at this level of detail, where any input can be controlled and any output observed, testing is a trivial process. It would suffice to compare the state of each line with its expected value; any disagreement would signify a presence of a fault. Unfortunately, this detail of monitoring is not feasible in any real circuit and, thus, a more abstract CAM cell model is necessary.

3 High-Level CAM Model

Most testing algorithms utilize sequences of 'read', 'write' and 'compare' operations as input, and observe the resulting output. Accordingly, we introduce a high-level model of a CAM cell. This model is derived from the low-level model of Section 2.

We represent the behavior of a fault-free CAM cell as a finite-state machine (FSM), which is shown in Figure 2(a). The input x of this FSM comprises all seven operations of a CAM cell, and the output y comprises the responses to these operations without regard to the output's origin, i.e., the $bit/\overline{bit}$ lines or *ML*. State s represents the value stored by the cell. Formally, a *fault-free CAM cell* is a Mealy automaton

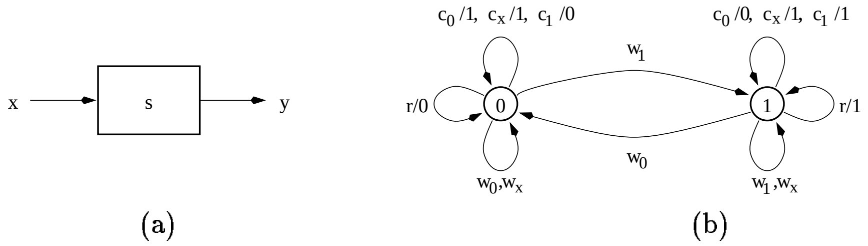

Figure 2: (a) Simplified behavioral model (b) Behavior of a fault-free cell.

$$M = (Q, X, Y, \delta, \lambda),$$

where $Q = \{0,1\}$ is the set of states, $X = \{r, w_0, w_1, w_\times, c_0, c_1, c_\times\}$ is the set of input symbols, $Y = \{0, 1, I, \$\}$ is the set of output symbols, where \$ is a formal symbol denoting lack of output during 'write' operations, and the transition function δ and the output function λ are defined by

$$\delta(q,x) = \begin{cases} 0, & \text{if } x = w_0, \\ 1, & \text{if } x = w_1, \\ q, & \text{otherwise} \end{cases}$$

and

$$\lambda(q,x) = \begin{cases} q, & \text{if } x = r, \\ 1, & \text{if } x = c_p \text{ and } q = p, \\ 0, & \text{if } x = c_p \text{ and } q \neq p, \\ 1, & \text{if } x = c_\times, \\ \$, & \text{otherwise.} \end{cases}$$

This automaton is depicted in Figure 2(b), where the symbol \$ has been omitted for clarity.

Example: For 'write 1' in state 0, $\delta(0, w_1) = 1$ and $\lambda(0, w_1) = \$$. For 'compare 1' in state 0, $\delta(0, c_1) = 0$ and $\lambda(0, c_1) = 0$.

4 Fault Model

A low-level behavioral analysis has been performed for every potential input stuck-at fault under a single-fault assumption; these analyses are presented in Appendix A. Subsequently, high-level models for each of these faults have been developed. The resulting *faulty CAM cells* are presented in Figures 3 and 4, where incorrect operations and outputs are in boldface.

Example: 'Write 0' operation in the presence of the $\bar{b}$*-sa-0* fault. Initial state: 0 10 00 1 $\cdot$ 1. First, b is lowered: 0 00 00 1 $\cdot$ 1. Then w is raised: 1 00 00 1 $\cdot$ 1. Since both b and $\bar{b}$ are low, s becomes indeterminate: 1 00 00 1 $\cdot$ I. Next, w is lowered: 0 00 00 1 $\cdot$ I. Finally, both b and $\bar{b}$ are raised: 0 10 00 1 $\cdot$ 0$|$1, and eventually s becomes either 0 or 1.

From the low-level model we construct an automaton. In this example, only operations w_0, $w_\times$ and r are incorrect, all others are correct; hence we get Figure 3(c).

As the example indicates, the b*-sa-0* and $\bar{b}$*-sa-0* faults increase the state set Q by the "indeterminate" state I. State I can be interpreted in two

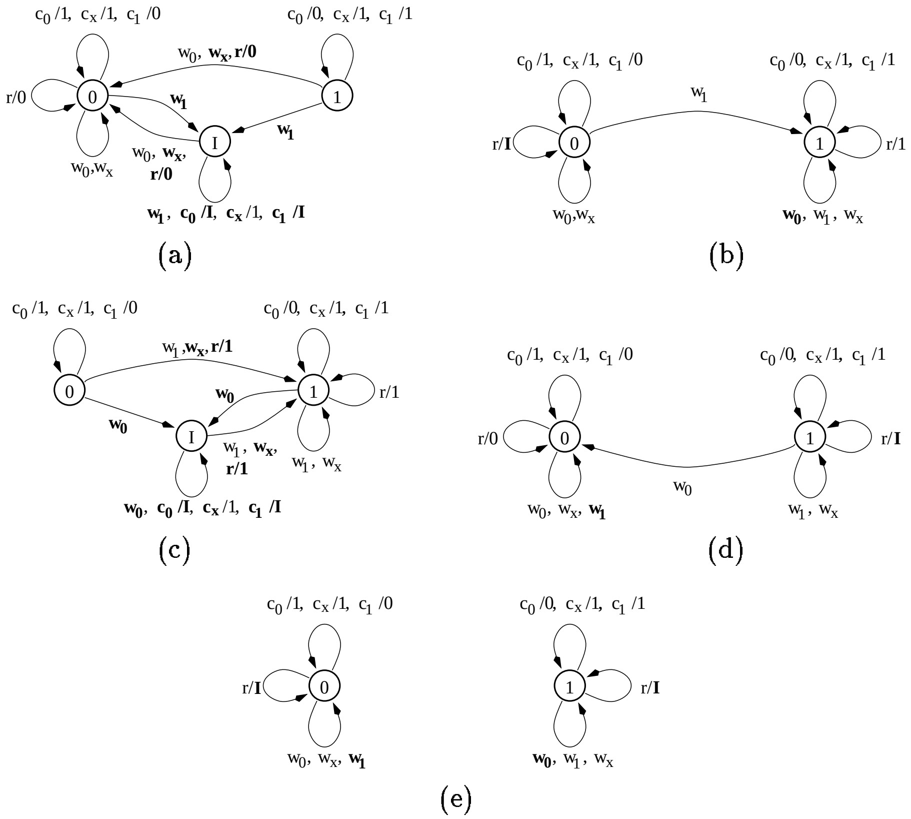

Figure 3: Faulty behaviors of a CAM cell: (a) *b-sa-0*, (b) *b-sa-1*, (c) *$\bar{b}$-sa-0*, (d) *$\bar{b}$-sa-1*, (e) *w-sa-0*.

ways. From an "asynchronous" point of view it represents a temporary, metastable state, where the cell holds some indeterminate logic value. This interpretation is important due to the possibility of simultaneous operations in this type of CAM. From the "synchronous" perspective it represents the loss of information regarding the current state of the cell due to the non-deterministic resolution of a metastable state. In either case, state *I* is not considered as an initial state.

For the two implementations of the 'write ×' operation, described in Section 2, the reader may verify that in the presence of faults both 'read' and 'write ×' operations affect the cell in a similar manner (when the output is ignored). The high-level model presented here is, therefore, appropriate regardless of how the 'write ×' operation has been implemented.

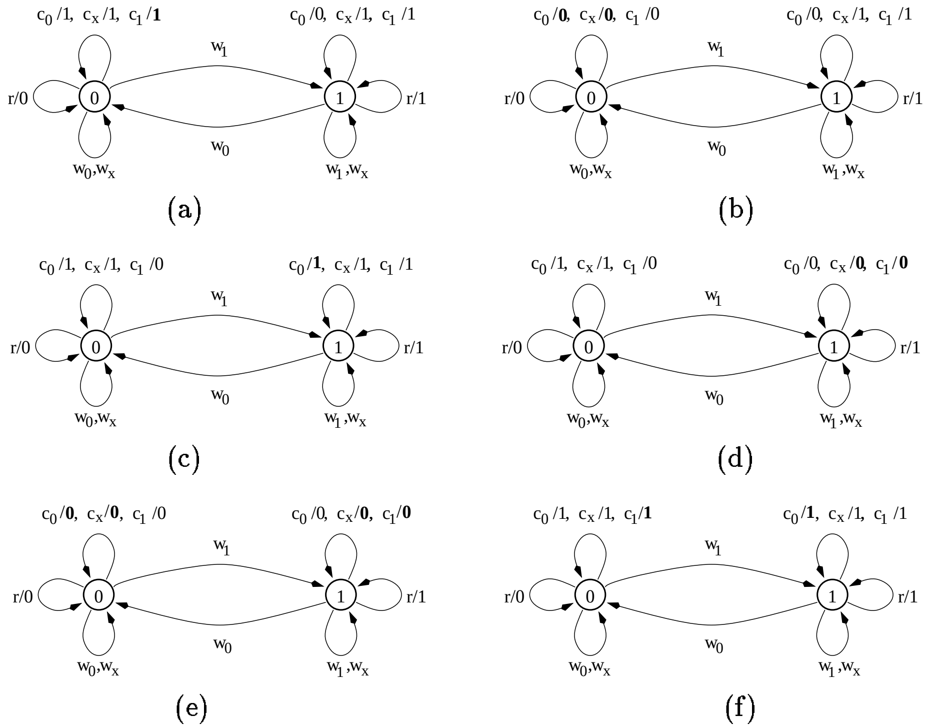

Figure 4: Faulty behaviors of a CAM cell: (a) *c-sa-0*, (b) *c-sa-1*, (c) *$\bar{c}$-sa-0*, (d) *$\bar{c}$-sa-1*, (e) *m-sa-0*, (f) *m-sa-1*.

The comparison of each of the faulty machines to the fault-free CAM cell has led to the derivation of simple tests for each fault. Table 2 lists all the shortest tests that end in either a 'compare' or a 'read' operation. These tests have been generated by the OBSERVER[1] program for all input stuck-at faults except the *w-sa-1* fault. We refer to the faults that are detectable in a single cell as *independently testable*. The use of $w_\times$ and 'compare' operations rather than 'read' operations for the purpose of propagating faulty responses is dictated by the unreliable output of the 'read' operation for four faults related to the storage section of the cell: *b-sa-1*, *$\bar{b}$-sa-1*, *w-sa-0* and *w-sa-1*. In the last example a $w_0 w_\times$ forces the faulty cell to state 1, whereas a good cell would be in state 0. A w_0 alone is not enough to force the faulty cell to the erroneous state.

[1] OBSERVER is a program for diagnosing and testing sequential machines. It is based on the theory developed in [2] and was written at the University of Waterloo by C.-J. Shi; additional features were later added by P. Kwiatkowski and P. R. Sidorowicz.

The *w-sa-1* fault automaton, is identical to that of the fault-free CAM cell; therefore, no test exists for a single cell. Consequently, this is the only input stuck-at fault that is not independently testable. However, this fault is detectable in conjunction with other words, and will be considered later.

Partly with the aid of the Observer program, we have found tests

$$T_{isaf} = w_0 w_1 w_\times c_1 c_0 w_0 w_\times c_0 c_1,$$

and

$$T'_{isaf} = w_1 w_0 w_\times c_0 c_1 w_1 w_\times c_1 c_0$$

of length 9, that detect all independently testable input stuck-at faults. It has been shown that, for every independently testable fault, there exists a test that is included in T_{isaf} (T'_{isaf}). The details are presented in Appendix B. Moreover, these two tests are *irredundant*, since the removal of any of the input symbols in the test will result in some fault becoming undetectable. Tests for individual faults that are included in T_{isaf} are indicated in Table 2 in boldface. With the exception of tests for *c-sa-0* and $\bar{c}$*-sa-0* faults, these tests have been chosen because their response in a fault-free cell is a match, and in a faulty cell, a mismatch. Thus, the CAM's property, that a match line discharges in the presence of even a single bit mismatch, can be efficiently utilized.

A cell stuck-at-0 (stuck-at-1) fault is similar to a cell with *w-sa-0*, when that cell is started in state 0 (state 1). This fault is detected by $w_1 c_1$ ($w_0 c_0$). Also, a cell unable to undergo the $1 \rightarrow 0$ ($0 \rightarrow 1$) transition is similar to *b-sa-1* ($\bar{b}$*-sa-1*). This fault is detected by $w_1 w_0 c_0$ ($w_0 w_1 c_1$).

5 Extension to n-word by 1-bit CAM

We now consider an n-bit CAM where each word consists of a single cell. At this point, some assumptions must be made about the functionality of the peripheral circuitry in a fault-free CAM.

1. The address decoder can raise at most a single write line. Consequently, only one word can be written to or read from at any given time.

2. Match detection is done through a *hit* line (high when at least one match is detected), a *multi-hit* line (high when multiple matches are detected), and an encoder that returns the address k of the highest priority match line[2].

[2] By convention, address 1 has the highest, and address n the lowest priority.

Table 2: Summary of input stuck-at faults.

Fault	Test Sequence	Faulty Response	Fault-Free Response
b-sa-0	$\mathbf{w_1 w_\times c_1}$	0	1
	$w_1 w_\times c_0$	1	0
	$w_1 r$	0	1
b-sa-1	$\mathbf{w_1 w_0 c_0}$	0	1
	$w_1 w_0 c_1$	1	0
	$w_1 w_0 r$	1	0
$\bar{b}$*-sa-0*	$\mathbf{w_0 w_\times c_0}$	0	1
	$w_0 w_\times c_1$	1	0
	$w_0 r$	1	0
$\bar{b}$*-sa-1*	$\mathbf{w_0 w_1 c_1}$	0	1
	$w_0 w_1 c_0$	1	0
	$w_0 w_1 r$	0	1
w-sa-0	$\mathbf{w_1 c_1 w_0 c_0}$	1 0 or 0 1	1 1
	$w_0 c_0 w_1 c_1$	″	″
	$w_1 c_0 w_0 c_1$	″	0 0
	$w_0 c_1 w_1 c_0$	″	″
w-sa-1	see Section 5.1		
c-sa-0	$\mathbf{w_0 c_1}$	1	0
c-sa-1	$\mathbf{w_0 c_0}$	0	1
	$w_0 c_\times$	0	1
$\bar{c}$*-sa-0*	$\mathbf{w_1 c_0}$	1	0
$\bar{c}$*-sa-1*	$\mathbf{w_1 c_1}$	0	1
	$w_1 c_\times$	0	1
m-sa-0	$\mathbf{c_1 c_0}$	0 0	1 0 or 0 1
	$c_0 c_1$	″	″
	$c_\times$	0	1
m-sa-1	$\mathbf{c_1 c_0}$	1 1	1 0 or 0 1
	$c_0 c_1$	″	″

We model the correct behavior of an n-word by 1-bit word CAM as a finite state automaton

$$M = (Q, X, Y, \delta, \lambda),$$

where $Q = \{0,1\}^n$, $X = (\bigcup_{i=1}^{n} A_i) \cup C$ with $A_i = \{r^i, w_0^i, w_1^i, w_\times^i\}$, $C = \{c_0, c_1, c_\times\}$, $Y = \{0, 1, \$, k\}$, where $0 \leq k \leq n$, and the transition functionδ and the output function λ are defined by

$$\delta((q^1, \ldots, q^i, \ldots, q^n), x) = \begin{cases} (q^1, \ldots, 0, \ldots, q^n), & \text{if } x = w_0^i, \\ (q^1, \ldots, 1, \ldots, q^n), & \text{if } x = w_1^i, \\ (q^1, \ldots, q^i, \ldots, q^n), & \text{otherwise,} \end{cases}$$

and

$$\lambda((q^1, \ldots, q^i, \ldots, q^n), x) = \begin{cases} q^i, & \text{if } x = r^i, \\ min(k), & \text{if } x = c_p \text{ and } q^k = p, \\ 0, & \text{if } x = c_p \text{ and } \neg\exists k : q^k = p, \\ 1, & \text{if } x = c_\times, \\ \$, & \text{otherwise.} \end{cases}$$

The inputs $r^i, w_0^i, w_1^i, w_\times^i$ denote the 'read', 'write 0', 'write 1' and 'write ×' operations on word i, respectively. The inputs $c_0, c_1, c_\times$ denote compare operations with the contents of every cell in the CAM. The output \$ is merely a formal symbol denoting lack of output during 'write' operations.

5.1 Testing the *w-sa-1* fault

As indicated earlier, *w-sa-1* is not independently testable and affects all the cells along the faulty *WL*. However, this fault is detectable in the following manner:

Assume *w^i-sa-1*. Since word i is always accessed, a 'read' or 'write' operation may be applied to two words simultaneously. Note that if the two accessed words contain opposing values, 'read' operations will yield unreliable results. A possible test is:

$$T_{w\text{-}sa\text{-}1} = (w_0^i)^{\updownarrow} w_1^n c_1 w_0^n w_1^u c_1,$$

where $1 \leq i \leq n$ and $1 \leq u < n$. The symbol $(\ )^{\updownarrow}$ denotes n operations. These can be done in order either from n to 1 or from 1 to n; hence the $\updownarrow$. Initially, 0s are written into every word. If word u is the faulty word, $u \neq n$, then w_1^n will write 1s to both u and n. The first c_1 will generate a multiple hit and the returned value of k will indicate the address of the faulty word, as the faulty word has a higher priority than word n. If word n is the faulty word, then word u is not, so w_0^n restores the initial state of the CAM, w_1^u will write 1 to words u and n, and the second c_1 will generate a multiple hit. Here the value of k is ignored, as the location of the faulty word is known to be n. Thus, the occurrence of a multiple hit indicates the *w-sa-1* fault.

The reader may note that the *w-sa-1* fault is similar to a coupling fault.

5.2 T_{isaf} for n-word by 1-bit CAM

The test T_{isaf} is modified into a march test

$$T_{n\text{-}isaf} = (w_0^i)^{\updownarrow}(w_1^i w_\times^i c_1)^{n\downarrow 1} c_0 (w_0^i w_\times^i c_0)^{n\downarrow 1} c_1$$

for the n-word by 1-bit CAM. The symbol $(\ )^{n\downarrow 1}$ denotes the direction of the march elements: from n to 1. This direction is dictated by the priority scheme used in the match line encoder and always follows from the lowest to the highest priority. First, all the words are initialized to 0. Then, in the fault-free CAM, for each $w_1^i w_\times^i$ in a march element, the subsequent c_1 input should produce a value of k: $k = i$. Multiple hits during this test are expected; thus the status of the *multi-hit* line must be ignored. The mismatching c_0 is performed once per march element, on a CAM uniformly filled with 1s, and should produce a value $k = 0$, indicating a global mismatch. Any hit, i.e., any $k > 0$, indicates a fault. The rest of the test sequence is similar, with 0 and 1 interchanged.

5.3 Proposed test for n-word by 1-bit word CAM

To achieve 100% fault coverage under the input stuck-at fault model, we combine $T_{n\text{-}isaf}$ with $T_{w\text{-}sa\text{-}1}$:

$$T_{n\text{-}saf} = (w_0^i)^{\updownarrow}(w_1^i w_\times^i c_1)^{n\downarrow 1} c_0 (w_0^i w_\times^i c_0)^{n\downarrow 1} c_1 (w_1^n c_1 w_0^n w_1^u c_1).$$

We remind the reader that we are using the single-fault assumption. Note that the initial $(w_0^i)^{\updownarrow}$ of the $T_{w\text{-}sa\text{-}1}$ test has been dropped, as the CAM is expected to hold only 0s at the end of $T_{n\text{-}isaf}$. Note also that $w_\times$ is not needed in the last part of the test, because the faults *b-sa-0* and $\bar{b}$*-sa-0* which require $w_\times$ would have been detected by the first part of the test.

The $T_{n\text{-}saf}$ test has of $5n + 3$ 'write' operations and $2n + 4$ 'compare' operations and has an overall length of $7n + 7$.

6 Extension to a 1-word by l-bit CAM

We now consider a single row of cells comprising a CAM word. Recall, that a match line is a 'wired AND' of match responses of all cells in a word.

Let $\mathcal{B} = \{0,1\}$ and $\mathcal{T} = \{0,1,\times\}$. To handle masking of selected bits we define a function $\diamond : \ \mathcal{T} \times \mathcal{B} \to \mathcal{B}$, where

$$\begin{array}{ll} 0 \diamond 0 = 0, & 0 \diamond 1 = 0, \\ 1 \diamond 0 = 1, & 1 \diamond 1 = 1, \\ \times \diamond 0 = 0, & \times \diamond 1 = 1. \end{array}$$

In this manner, for $x \in \mathcal{T}$ and $y \in \mathcal{B}$, $x \diamond y$ returns the value of y if $x = \times$, or the value of x otherwise.

We extend this function to l-bit words as a bit-by-bit operation. Let $\mathcal{V} = \mathcal{B}^l$ and $\mathcal{P} = \mathcal{T}^l$ and let $t \in \mathcal{P}$ and $t' \in \mathcal{V}$. Then $t = (t_1, \ldots, t_l)$ and $t' = (t'_1, \ldots, t'_l)$, where $t_i \in \mathcal{T}$, $t'_i \in \mathcal{B}$ for $1 \leq i \leq l$, and

$$t \diamond t' = t_1 \diamond t'_1, \ldots, t_l \diamond t'_l.$$

The behavior of a 1-word by l-bit CAM can be specified by the automaton

$$M = (Q, X, Y, \delta, \lambda)$$

where $Q = \mathcal{V}$, $X = \{r\} \cup (\bigcup_{p \in \mathcal{P}} w_p) \cup (\bigcup_{p \in \mathcal{P}} c_p)$, and $Y = \mathcal{V} \cup \{0, 1, \$\}$. For $q \in \mathcal{V}$, $x \in X$ the transition function δ and the output function λ are defined by

$$\delta(q,x) = \begin{cases} q, & \text{if } x = r \text{ or } x = c_p,\ p \in \mathcal{P}, \\ p \diamond q, & \text{if } x = w_p,\ p \in \mathcal{P}, \end{cases}$$

and

$$\lambda(q,x) = \begin{cases} q, & \text{if } x = r, \\ 1, & \text{if } x = c_p \text{ for some } p \in \mathcal{P} \text{ and } p \diamond q = q, \\ 0, & \text{if } x = c_p \text{ for some } p \in \mathcal{P} \text{ and } p \diamond q \neq q, \\ \$, & \text{otherwise.} \end{cases}$$

Faults affecting the *bit*/$\overline{bit}$ lines of each of the l cells are detectable by performing the respective tests on all cells in parallel since these faults manifest themselves with a mismatch. Detection of the *w-sa-0*, *m-sa-0* and the *m-sa-1* faults is also done in parallel, as these faults affect the entire word the same manner. Detection of each of the c^j*-sa-0* and $\bar{c}^j$*-sa-0* faults, where $1 \leq j \leq l$, is more complex. Their respective tests have a mismatch as a fault-free response and a match as a faulty one; thus, they cannot be applied to all the cells in parallel, as no faulty response would ever be propagated along the *ML*. To propagate a faulty response along the *ML*, a mismatch must be attempted on each cell j individually, while simultaneously applying a $c_\times$ to the remaining $l-1$ cells. Let $\times^{j-1}0\times^{l-j}$ be the l-bit word with 0 in position j and $\times$ in every other position. The word $\times^{j-1}1\times^{l-j}$ is similarly defined. The symbol $[c_{\times^{j-1}0\times^{l-j}}]^{\leftrightarrow}$ represents a sequence of l 'compare' operations to words of the form $\times^{j-1}0\times^{l-j}$, where j varies from 1 to l or from l to 1. The extension of the T_{isaf} test to the 1-word by l-bit CAM takes the form

$$\begin{aligned} T_{l\text{-}isaf} \;=\;& w_{0\ldots0}w_{1\ldots1}w_{\times\ldots\times}c_{1\ldots1}[c_{\times^{j-1}0\times^{l-j}}]^{\leftrightarrow} \\ & w_{0\ldots0}w_{\times\ldots\times}c_{0\ldots0}[c_{\times^{j-1}1\times^{l-j}}]^{\leftrightarrow}, \end{aligned}$$

where $w_{0\ldots0}$ denotes the writing of an all-0 word, etc.

The $T_{l\text{-}isaf}$ test consists of 5 'write' operations and $2l+2$ 'compare' operations and has an overall length of $2l+7$.

7 Extension to n-word by l-bit CAM

The combination of both extensions described above yields the specification for the behavior of a n-word by l-bit CAM.

Let $\mathcal{V}$, $\mathcal{P}$ and $\diamond$ be defined as before. The correct behavior of a n-word by l-bit CAM is denoted by the automaton

$$M = (Q, X, Y, \delta, \lambda),$$

where $Q = \mathcal{V}^n$, $X = (\bigcup_{i=1}^{n} A_i) \cup C$ with $A_i = \{r^i\} \cup (\bigcup_{p\in\mathcal{P}} w_p^i)$, $C = \bigcup_{p\in\mathcal{P}} c_p$, $Y = \mathcal{V} \cup \{0, \ldots, n, \$\}$. For $1 \le i \le n$ and for $q^i \in \mathcal{V}$, $x \in X$, $p \in \mathcal{P}$, the transition function δ and the output function λ are defined by

$$\delta((q^1,\ldots,q^i,\ldots,q^n),x) = \begin{cases} (q^1,\ldots,q^i,\ldots,q^n), & \text{if } x = r^i \text{ or } x = c_p, \\ (q^1,\ldots,p \diamond q^i,\ldots,q^n), & \text{if } x = w_p^i, \end{cases}$$

and

$$\lambda((q^1,\ldots,q^i,\ldots,q^n),x) = \begin{cases} q^i, & \text{if } x = r^i,\ 1 \le i \le n \\ min(k), & \text{if } x = c_p \text{ and } p \diamond q^k = q^k, \\ 0, & \text{if } x = c_p \text{ and } \neg\exists k : p \diamond q^k = q^k, \\ \$, & \text{otherwise.} \end{cases}$$

The following test provides 100% fault coverage under the input stuck-at fault model:

$$T_{saf} \;=\; (w^i_{0\ldots0})^{\updownarrow}(w^i_{1\ldots1}w^i_{\times\ldots\times}c_{1\ldots1})^{n\downarrow1}[c_{\times^{j-1}0\times^{l-j}}]^{\leftrightarrow}$$

$$(w^i_{0\ldots0} w^i_{\times\ldots\times} c_{0\ldots0})^{n\downarrow1} [c_{\times^{j-1}1\times^{l-j}}]^{\leftrightarrow}$$
$$(w^n_{1\ldots1} c_{1\ldots1} w^n_{0\ldots0} w^u_{1\ldots1} c_{1\ldots1}).$$

This test consists of two parts, analogous to those of the $T_{n\text{-}saf}$ test. The first part consists of an initialization which sets all words to $0\ldots0$, followed by two march elements. For a fault-free CAM, each of the march elements should result in n hits with $k = i$. Each march element should be followed by l mismatches at $k = 0$. Any other response indicates a fault. In the second part (last line), multiple hits are monitored, since they indicate faults.

The T_{saf} test consists of $5n+3$ 'write' operations and $2n+2l+2$ 'compare' operations. Therefore, the length of our test for all input stuck-at faults in an n-word by l-bit CAM is $7n + 2l + 5$.

8 DFT Suggestions

The length of our test for input stuck-at faults is linear in the number of bits in the CAM. This test length can be substantially shortened by certain DFT hardware enhancements.

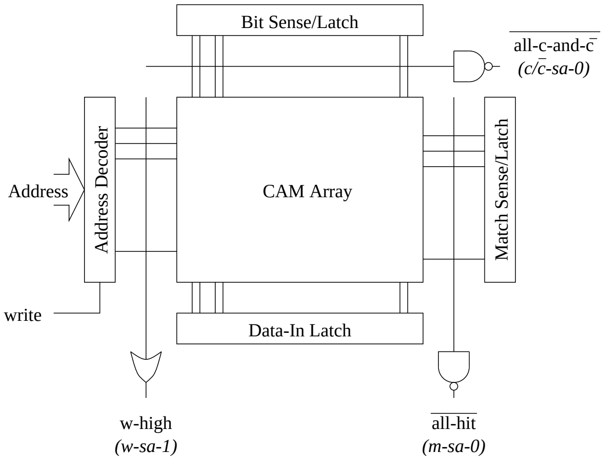

Figure 5: DFT suggestions.

The enhancements depicted as logic gates in Figure 5 target faults that contribute most to the length of the test; these enhancements are described below.

- An $\overline{\textit{all-hit}}$ output, indicating that a match has not been detected on every word in the CAM simultaneously. Note that this output directly detects the *m-sa-0* fault. Given this additional output, test T_{saf} can be modified to

$$\begin{aligned} T'_{saf} &= (w^i_{0\ldots0})^{\updownarrow}(w^i_{1\ldots1}w^i_{\times\ldots\times})^{n\downarrow1}c_{1\ldots1}[c_{\times^{j-1}0\times^{l-j}}]^{\leftrightarrow} \\ &\quad (w^i_{0\ldots0}w^i_{\times\ldots\times})^{n\downarrow1}c_{0\ldots0}[c_{\times^{j-1}1\times^{l-j}}]^{\leftrightarrow} \\ &\quad (w^n_{1\ldots1}c_{1\ldots1}w^n_{0\ldots0}w^u_{1\ldots1}c_{1\ldots1}), \end{aligned}$$

the length of which is reduced to $5n + 2l + 7$.

- An $\overline{\textit{all-c-and-}\bar{c}}$ output, indicating that all c and all $\bar{c}$ inputs are high. This output would eliminate elaborate testing for the *c-sa-0* and $\bar{c}$*-sa-0* faults. As a result, T'_{saf} can be further reduced to

$$\begin{aligned} T''_{saf} &= (w^i_{0\ldots0})^{\updownarrow}(w^i_{1\ldots1}w^i_{\times\ldots\times})^{n\downarrow1}c_{1\ldots1}(w^i_{0\ldots0}w^i_{\times\ldots\times})^{n\downarrow1}c_{0\ldots0}c_{1\ldots1} \\ &\quad (w^n_{1\ldots1}c_{1\ldots1}w^n_{0\ldots0}w^u_{1\ldots1}c_{1\ldots1}), \end{aligned}$$

  which has length $5n + 8$.

- A *w-high* output, indicating that at least one write line is active. This output can directly detect any *w-sa-1* fault, as well as *w-sa-0* faults during 'write' operations; hence, the suffix of the test T''_{saf},

$$(w^n_{1\ldots1}c_{1\ldots1}w^n_{0\ldots0}w^u_{1\ldots1}c_{1\ldots1}),$$

  can be omitted. If, however, the *w-high* output is unavailable, the *w-sa-1* fault can still be detected by reducing the above-mentioned suffix to $([\bar{b}^l = 0\ldots0]c_{1\ldots1})$, where $[\bar{b}^l = 0\ldots0]$ represents setting all the $\overline{\textit{bit}}$ lines to ground — like the first step in the 'write 0' operation — without actually raising any of the *WL*s. This will result in the word affected by the *w-sa-1* fault having $1\ldots1$ written into it. A match resulting from a subsequent $c_{1\ldots1}$ operation indicates a presence of this fault.

- The ability to perform a *bulk-write* operation to all words in the memory simultaneously, in conjunction with the $\overline{\textit{all-hit}}$ output, will result in a test length that is independent of the number words in the CAM. For example, the length of T'_{saf} would be reduced to $2l + 12$, and the length of T''_{saf} to 13.

Although these enhancements are meant to operate in *test mode*, it is possible that some of them may adversely affect the *mission mode* operation of the CAM; therefore, a judicious compromise between testability and performance may be necessary.

9 Verification of Other CAM Tests

Several algorithms for testing CAMs have been reported [4, 8, 9]. These algorithms are applicable to CAM designs which incorporate an explicit word addressing scheme of the type found in conventional RAMs. They are not applicable to designs with implicit entry addressing, like that of McAuley and Cotton [10].

9.1 Giles and Hunter (1985)

One of the first tests for CAMs, of length $8n+2l$, was presented by Giles and Hunter [4]; we refer to this test as $T_{G\&H}$. The following eight steps of this test are quoted from [4]; we have re-labeled the array dimension variables to be consistent with our notation:

1. Proceeding from top to bottom of the array, write the entry's position number into each entry.
2. Then from the bottom up, compare for each number and observe each entry hit. At this point, the CAM is in a known initial state.
3. Repeat steps 1 and 2 but with the entry numbers complemented.
4. From bottom up, write the entry number into each entry.
5. From top down, compare for each entry number and observe each entry hit.
6. Fill the entire CAM array with 0's.
7. Do l compares walking a 1 through a field of 0's. Each of these l compares should miss.
8. Repeat steps 6 and 7 except fill with 1's, and compare walking a 0 through a field of 1's.

The authors claim that this test detects all single stuck-at-faults in the CAM array found in the Motorola MC68851 Paged Memory Management Unit, but give no proof that this indeed is the case. Since the transistor circuit for this CAM cell was not provided, we were unable to verify this claim. We have, however, applied this test to the CAM circuit of Figure 1(a) to determine this test's fault coverage with respect to our input stuck-at fault model, under the single-fault assumption.

We represent this test using our notation. Since we index the words in our CAM from 1 to n, (the former being the top, and the latter the bottom of the CAM array) the value of k written into each word i is equal to $i-1$; $\bar{k}$ denotes the 1's complement of k.

$$
\begin{aligned}
T_{G\&H} \; = \; & (w^i_k)^{1\downarrow n}(c_k)^{n\uparrow 1} \\
& (w^i_{\bar{k}})^{1\downarrow n}(c_{\bar{k}})^{n\uparrow 1} \\
& (w^i_k)^{n\uparrow 1}(c_k)^{1\downarrow n} \\
& (w^i_{0\ldots 0})^{\updownarrow}[c_{0^{j-1}10^{l-j}}]^{\leftrightarrow} \\
& (w^i_{1\ldots 1})^{\updownarrow}[c_{1^{j-1}01^{l-j}}]^{\leftrightarrow}.
\end{aligned}
$$

The $T_{G\&H}$ test is limited to CAM arrays where $n \leq 2^l$ in order to ensure that a distinct bit pattern is written into every word of the array. In arrays where $n > 2^l$, duplicate bit patterns would be unavoidable. These duplicates would cause multiple hits, thus precluding the verification of individual match lines, and also precluding the detection of the *w-sa-1* faults.

The analysis of the $T_{G\&H}$ test in our automaton model of input stuck-at faults shows that, in the cell of Figure 1(a), $T_{G\&H}$ does not reliably detect the *b-sa-0* and $\bar{b}$*-sa-0* faults. This is due to a metastable state caused by 'write 1' and 'write 0' operations in the presence of these two faults respectively; thus any subsequent output is not reliable. The details are presented in Appendix C.

There are $2l$ *b-sa-0* and $\bar{b}$*-sa-0* faults in an n-word by l-bit CAM. Since there are $4n+8l$ possible single faults in our fault model, $T_{G\&H}$ detects only $(1 - \frac{l}{2n+4l}) \cdot 100\%$ of faults in the input stuck-at fault model. For example [8], for $n = 32$, $l = 29$ only 83.89% of faults are detected.

We next modified the $T_{G\&H}$ test to remedy the problem of the *b-sa-0* and $\bar{b}$*-sa-0* faults. We have shown that, in the presence of these two faults, a 'write ×' operation forces the faulty cell to a determinate but erroneous state that is easily detectable. Therefore, by judiciously inserting 'write ×' operations into $T_{G\&H}$, we can achieve 100% fault coverage with respect to our fault model under the assumption that $n \leq 2^l$. This augmented test, presented below, has length $11n + 2l$.

$$
\begin{aligned}
T'_{G\&H} \; = \; & (w^i_k w^i_\times)^{1\downarrow n}(c_k)^{n\uparrow 1} \\
& (w^i_{\bar{k}} w^i_\times)^{1\downarrow n}(c_{\bar{k}})^{n\uparrow 1} \\
& (w^i_k w^i_\times)^{1\uparrow n}(c_k)^{n\downarrow 1} \\
& (w^i_{0\ldots 0})^{\updownarrow}[c_{0^{j-1}10^{l-j}}]^{\leftrightarrow} \\
& (w^i_{1\ldots 1})^{\updownarrow}[c_{1^{j-1}01^{l-j}}]^{\leftrightarrow}
\end{aligned}
$$

9.2 Mazumder et al. (1987)

Mazumder et al. [9] propose a test that can detect pattern-sensitive faults as well as stuck-at faults. In general, faults are considered to be 'pattern-sensitive' if a faulty behavior of a cell is manifested only when the remaining cells hold specific values. In their analysis, the authors make assumptions about the effect of certain operations on the state of the accessed memory cells and define a restricted pattern-sensitive fault model (PSF):

- Write operations which cause a cell to change state are called 'transition write' operations.
- Only write operations may be pattern-sensitive.
- 'Read' operations do not alter a cell's state.
- A base cell together with four additional cells immediately to the 'north', 'south', 'east' and 'west' of it constitute a 'von Neumann' neighborhood.
- A PSF occurs when a transition write operation to the base cell fails due to a fixed pattern in the neighborhood.

This test requires the incorporation of special circuitry in the match sensing periphery that allows simultaneous comparisons of every fifth match line; an enhancement that is not available in most CAMs. We have also demonstrated that 'read' operations alter a cell's state in the presence of some faults, thus violating the initial assumptions for this test. Thus, this test is not universally applicable to static CAMs and will not be investigated further.

9.3 Kornachuk et al. (1994)

The test presented by Kornachuk et al. [8] is a built-in self test (BIST) for embedded CAMs that use the cell of Figure 1(a). We refer to this test as $T_{\text{C-SM}}$. It is an adaptation of the SMARCH test, of length $24nl$, originally developed for testing embedded SRAMs [11]. SMARCH has been shown to detect stuck-at and coupling faults as well as stuck-open faults in the address decoder circuitry. In fact, SMARCH is a serialized version of the 'March C-' test [5]. The serialization was necessary in order to utilize built-in scan-path facilities of the CAM circuit.

The SMARCH test consists of the following six steps (march elements) as presented in [11]. We have re-labeled the array dimension variables to be consistent with our notation and use our symbols. The reader should be familiar with [8, 11].

```
1. FOR address $i = 1$ to $n$
   $(r^i_- w^i_0)^{(j-1)\to 0}$ $(r^i_0 w^i_0)^{(j-1)\to 0}$
   {this step initializes the CAM with 0's }
2. FOR address $i = 1$ to $n$
   $(r^i_0 w^i_1)^{(j-1)\to 0}$ $(r^i_1 w^i_1)^{(j-1)\to 0}$
   { read 0's and replace with 1's}
3. FOR address $i = 1$ to $n$
   $(r^i_1 w^i_0)^{(j-1)\to 0}$ $(r^i_0 w^i_0)^{(j-1)\to 0}$
   { read 1's and replace with 0's}
4. FOR address $i = n$ to 1
   $(r^i_0 w^i_1)^{(j-1)\to 0}$ $(r^i_1 w^i_1)^{(j-1)\to 0}$
   { read 0's and replace with 1's}
5. FOR address $i = n$ to 1
   $(r^i_1 w^i_0)^{(j-1)\to 0}$ $(r^i_0 w^i_0)^{(j-1)\to 0}$
   { read 1's and replace with 0's}
6. FOR address $i = n$ to 1
   $(r^i_0 w^i_0)^{(j-1)\to 0}$ $(r^i_0 w^i_0)^{(j-1)\to 0}$
   {only the first read is important. Final state is selectable.}
```

The subscript '$-$' indicates an arbitrary binary value used during the initialization step of the test. (Note that these 'read' and 'write' operations are applied to individual cells.)

The adaptation of the SMARCH test to the CAM circuit was possible due to the dual-port nature of this particular CAM design, where one set of differential lines ($bit/\overline{bit}$) is used only for the 'read' and 'write' operations, and a second set of differential lines ($cbl/\overline{cbl}$) is dedicated solely to the 'compare' operations. During $T_{C\text{-}SM}$, 'compare' operations occur in the same clock cycle as the 'read' and 'write' operations and thus are perceived to be executed in parallel. To be precise, 'compare' operations are indeed performed concurrently with the 'read' operations, but they occur immediately after the completion of the 'write' operations; thus the newly stored value is the subject of the comparison. Since these operations are performed in parallel, the length of $T_{C\text{-}SM}$ remains $24nl$, although twice as many operations are actually executed.

$T_{C\text{-}SM}$ presented in [8] can be represented using our notation. In this case, 'read' and 'write' and 'compare' operations are performed on entire words. The subscripts of 'write' and 'compare' operations represent the decimal equivalent of an l-bit binary number. The subscripts of 'read' operations represent the expected output of the operation. In this case, the subscript '$-$' indicates an arbitrary decimal value used during the initialization step of the test. The concurrent 'compare' operations are listed directly below the 'read' and 'write' operations.

$$
\begin{aligned}
T_{C\text{-}SM} \;=\; & \left(\begin{bmatrix} r^i_- w^i_0 \\ c_- c_- \end{bmatrix}^{0}_{j=l-1} \begin{bmatrix} r^i_0 w^i_0 \\ c_0 c_0 \end{bmatrix} \right)^{1\Downarrow n} \\
& \left(\begin{bmatrix} r^i_{(2^l-2^{(j+1)})} \; w^i_{(2^l-2^j)} \\ c_{(2^l-2^j)} \;\; c_{(2^l-2^j)} \end{bmatrix}^{0}_{j=l-1} \begin{bmatrix} r^i_{(2^l-1)} w^i_{(2^l-1)} \\ c_{(2^l-1)} c_{(2^l-1)} \end{bmatrix} \right)^{1\Downarrow n} \\
& \left(\begin{bmatrix} r^i_{(2^{(j+1)}-1)} \; w^i_{(2^j-1)} \\ c_{(2^j-1)} \;\; c_{(2^j-1)} \end{bmatrix}^{0}_{j=l-1} \begin{bmatrix} r^i_0 w^i_0 \\ c_0 c_0 \end{bmatrix} \right)^{1\Downarrow n} \\
& \left(\begin{bmatrix} r^i_{(2^l-2^{(j+1)})} \; w^i_{(2^l-2^j)} \\ c_{(2^l-2^j)} \;\; c_{(2^l-2^j)} \end{bmatrix}^{0}_{j=l-1} \begin{bmatrix} r^i_{(2^l-1)} w^i_{(2^l-1)} \\ c_{(2^l-1)} \; c_{(2^l-1)} \end{bmatrix} \right)^{n\Uparrow 1} \\
& \left(\begin{bmatrix} r^i_{(2^{(j+1)}-1)} \; w^i_{(2^j-1)} \\ c_{(2^j-1)} \;\; c_{(2^j-1)} \end{bmatrix}^{0}_{j=l-1} \begin{bmatrix} r^i_0 w^i_0 \\ c_0 c_0 \end{bmatrix} \right)^{n\Uparrow 1} \\
& \left(\begin{bmatrix} r^i_0 w^i_0 \\ c_0 c_0 \end{bmatrix}^{0}_{j=l-1} \begin{bmatrix} r^i_0 w^i_0 \\ c_0 c_0 \end{bmatrix} \right)^{n\Uparrow 1}
\end{aligned}
$$

Since multiple matches can be expected, a priority encoder is used to produce the highest address where a match was obtained. Table 3 illustrates the execution of the second march element for the CAM BIST for a 3×3 sample CAM; after the first march element, all cells are 0.

The analysis of the $T_{\text{C-SM}}$ test in our automaton model of input stuck-at faults shows that $T_{\text{C-SM}}$ detects all the faults in the model. The details are presented in Appendix D.

10 Concluding Remarks

Using a static CMOS CAM as an example, we have developed a fault modeling and test derivation methodology. The modeling steps involved include: circuit analysis, asynchronous behavior analysis, and a finite automaton

Table 3: CAM BIST March Element [8].

SMARCH Operation	CAM Inputs D0 D1 D2	Core Contents			Serial Data Out	Compare Results D0D1D2 to Core
		Word 0	Word 1	Word 2		
Word 0 Addressed						
Read	100	000	000	000	0	Miss
Write	100	100	000	000	0	Hit Word 0
Read	110	100	000	000	0	Miss
Write	110	110	000	000	0	Hit Word 0
Read	111	110	000	000	0	Miss
Write	111	111	000	000	0	Hit Word 0
Read	111	111	000	000	1	Hit Word 0
Write	111	111	000	000	1	Hit Word 0
Word 1 Addressed						
Read	100	111	000	000	0	Miss
Write	100	111	100	000	0	Hit Word 1
Read	110	111	100	000	0	Miss
Write	110	111	110	000	0	Hit Word 1
Read	111	111	110	000	0	Hit Word 0
Write	111	111	111	000	0	Hit Word 0,1
Read	111	111	111	000	1	Hit Word 0,1
Write	111	111	111	000	1	Hit Word 0,1
Word 2 Addressed						
Read	100	111	111	000	0	Miss
Write	100	111	111	100	0	Hit Word 2
Read	110	111	111	100	0	Miss
Write	110	111	111	110	0	Hit Word 2
Read	111	111	111	110	0	Hit Word 0,1
Write	111	111	111	111	0	Hit Word 0,1,2
Read	111	111	111	111	1	Hit Word 0,1,2
Write	111	111	111	111	1	Hit Word 0,1,2

representation of the fault-free and faulty circuits. An input stuck-at fault model has been defined for an n-word by l-bit static CMOS CAM. A test with 100% fault coverage with respect to this fault model has been constructed. This test also detects all the usual cell stuck-at and transition faults. It has also been demonstrated that 'read' operations are inappropriate sources of output for testing static CMOS CAMs, and that the 'compare' is a more reliable choice. In addition, some DFT enhancements that reduce the test length have been suggested.

The effect of other types of faults on the operation of this CAM cell is the topic of continuing research.

Acknowledgments

We thank Ken Schultz of Nortel Corporation's Memory Development Team, for providing extensive information regarding various defects and their effects on the behavior of a CAM. We also express our gratitude to Robert Berks, Tracey Bogue, Edward Dengler, Radu Negulescu and all members of the Maveric Group for their insightful comments and suggestions.

References

[1] H. Bergh, J. Eneland, and L-E. Lundström. A fault-tolerant associative memory with high-speed operation. *IEEE Journal of Solid-State Circuits*, 25(4):912–919, August 1990.

[2] J. A. Brzozowski and H. Jürgensen. A model for sequential machine testing and diagnosis. *Journal of Electronic Testing: Theory and Applications*, 3:219–234, 1992.

[3] B. F. Cockburn and J. A. Brzozowski. Near-optimal tests for classes of write-triggered coupling faults in RAMs. *Journal of Electronic Testing: Theory and Applications*, 3:251–264, 1992.

[4] G. Giles and C. Hunter. A methodology for testing content-addressable memories. In *Proc. International Test Conference*, pages 471–474. IEEE Computer Society Press, 1985.

[5] A. J. van de Goor. *Testing Semiconductor Memories.* John Wiley & Sons, 1991.

[6] A. J. van de Goor and B. Smit. The automatic generation of march tests. In *Records of the IEEE International Workshop on Memory Technology, Design and Testing*, pages 86–91, San Jose, CA, August 1994. IEEE Computer Society Press.

[7] A. J. van de Goor and B. Smit. Automating the verification of memory tests. In *IEEE VLSI Test Symposium*, pages 312–318, Cherry Hill, NJ, April 1994. IEEE Computer Society Press.

[8] S. Kornachuk, L. McNaughton, R. Gibbins, and B. Nadeau-Dostie. A high speed embedded cache design with non-intrusive BIST. In *Records of the IEEE International Workshop on Memory Technology, Design and Testing*, pages 40–45. IEEE, August 1994.

[9] P. Mazumder, J. H. Patel, and W. K. Fucks. Design and algorithms for parallel testing of random access and content addressable memories. In *Proc. ACM/IEEE Design Automation Conference*, pages 688–694. IEEE Computer Society Press, 1987.

[10] A. J. McAuley and C. J. Cotton. A self-testing reconfigurable CAM. *IEEE Journal of Solid-State Circuits*, 26(3):257–261, March 1991.

[11] B. Nadeau-Dostie, A. Silburt, and V.K. Agarval. Serial interfacing for embedded-memory testing. *IEEE Design & Test of Computers*, 7(2):56–64, April 1990.

[12] K. J. Schultz. *CAM-Based Circuits for ATM Switching Networks.* PhD thesis, University of Toronto, 1996.

A CAM Fault Analysis

In this section operations observably affected by input stuck-at faults will be discussed. The behaviors of operations not listed here may differ slightly from their fault-free counterparts in the low-level model; however in the high-level model these differences cannot be observed. Abbreviated fault names appear in parentheses.

***bit-sa-0* (*b-sa-0*) fault.** This fault does not affect operations where the bit lines are not used, or those where the bit line is normally driven to 0. This includes all 'compare' operations, the 'write 0' operation and the 'read' operation in state 0. Table 4 describes the faulty behaviors in the presence of this fault. Deviations from the fault-free behavior are indicated by bold type.

Table 4: CAM cell operation with *bit-sa-0* fault.

Read operations		
	$(s = 1)$	
Description	$w\ b\bar{b}\ c\bar{c}\ m{\cdot}s$	
initial state	0 **0**1 00 1·1	
float $bit/\overline{bit}$	0 **0**$\tilde{1}$ 00 1·1	
raise *WL*	1 **0**$\tilde{1}$ 00 1·1	
bit or $\overline{bit}$ discharges	1 **0**$\tilde{1}$ 00 1·**0**	
read $bit/\overline{bit}$ & lower *WL*	0 **0**$\tilde{1}$ 00 1·**0**	
raise $bit/\overline{bit}$	0 **0**1 00 1·**0**	

Write operations $(s = 0)$		
	W-1	
Description	$w\ b\bar{b}\ c\bar{c}\ m{\cdot}s$	
initial state	0 **0**1 00 1·0	
set $bit/\overline{bit}$	0 **0**0 00 1·0	
raise *WL*	1 **0**0 00 1·0	
new state	1 **0**0 00 1·**I**	
lower *WL*	0 **0**0 00 1·**I**	
raise $bit/\overline{bit}$	0 **0**1 00 1·**0**|**1**	

Write operations $(s = 1)$		
	W-1	**W-×**
Description	$w\ b\bar{b}\ c\bar{c}\ m{\cdot}s$	$w\ b\bar{b}\ c\bar{c}\ m{\cdot}s$
initial state	0 **0**1 00 1·1	0 **0**1 00 1·1
set $bit/\overline{bit}$	0 **0**0 00 1·1	0 **0**1 00 1·1
raise *WL*	1 **0**0 00 1·1	1 **0**1 00 1·1
new state	1 **0**0 00 1·**I**	1 **0**1 00 1·**0**
lower *WL*	0 **0**0 00 1·**I**	0 **0**1 00 1·**0**
raise $bit/\overline{bit}$	0 **0**1 00 1·**0**|**1**	0 **0**1 00 1·**0**

During the 'read' operation in state 1, the grounded bit line forces the *s* node to ground, causing the cell's state to change before the $\overline{bit}$ line has a chance to discharge. As a result, a 0 is always detected by the sense amplifiers. During the 'write 1' operation, when *WL* is asserted, both $bit/\overline{bit}$ lines are 0 causing both pull-up transistors to conduct which, in turn, results in a metastable state. Once *WL* is de-asserted one of the pull-down transistors dominates over the other and the cell eventually reverts back to one of the two quiescent states. Since the state of the cell cannot be predicted after a 'write 1', this operation must be followed immediately by a 'write ×' or a 'read' operation which will force the cell into a determinate but faulty state. The 'write ×' operation changes the cell's state to 0, thus resembling the behavior of a 'write 0' operation.

Analogous behavior is exhibited in the presence of the $\overline{bit}$*-sa-0* fault.

***bit-sa-1* (*b-sa-1*) fault.** The reader can verify that every operation except for 'read' and 'write 0' is not affected by this fault. This is because either *bit* is always 1 by definition, as in the case of 'write 1', 'write ×', and all 'compare' operations, or as in the case of 'write 0' in state 0, when the fact that *bit* is stuck-at 1 does not alter the cell's state. Table 5 describes the faulty behaviors of a CAM cell affected by the *bit-sa-1* fault.

Table 5: CAM cell operation with *bit-sa-1* fault.

Read operation	
	$(s = 0)$
Description	$w\ b\bar{b}\ c\bar{c}\ m{\cdot}s$
initial state	0 11 00 1·0
float $bit/\overline{bit}$	0 $\mathbf{1}\tilde{1}$ 00 1·0
raise *WL*	1 $\mathbf{1}\tilde{1}$ 00 1·0
bit or $\overline{bit}$ discharges	1 $\mathbf{1}\tilde{1}$ 00 1·0
read $bit/\overline{bit}$ & lower *WL*	0 $\mathbf{1}\tilde{1}$ 00 1·0
raise $bit/\overline{bit}$	0 11 00 1·0

Write 0 operation	
	$(s = 1)$
Description	$w\ b\bar{b}\ c\bar{c}\ m{\cdot}s$
initial state	0 11 00 1·1
set $bit/\overline{bit}$	0 **11** 00 1·1
raise *WL*	1 **11** 00 1·1
new state	1 **11** 00 1·**1**
lower *WL*	0 **11** 00 1·**1**
raise $bit/\overline{bit}$	0 11 00 1·**1**

A ‘read’ operation in state 0 will be misinterpreted by the sense circuitry because both bit lines remain high; therefore, a ‘read’ is not a reliable source of output. (Since the ‘compare’ operations are not affected by this fault, they are more suitable for fault detection.) When the cell is in state 1, a ‘write 0’ operation fails, because the state transition does not occur.

Analogous behavior is exhibited in the presence of the $\overline{bit}$*-sa-1* fault.

***WL-sa-0* (*w-sa-0*) fault.** Since this fault affects *WL*, it alters the behavior of operations for which this line is used, i.e. ‘read’ and ‘write’ operations. ‘Compare’ operations are not affected by this fault, and function correctly. Table 6 describes the faulty behaviors of a CAM cell in the presence of a *WL-sa-0* fault. From this table, we see that a ‘read’ operation will fail, since neither bit line is allowed to discharge.

Table 6: CAM cell operation with *WL-sa-0* fault.

Read operations		
	$(s = 0)$	$(s = 1)$
Description	$w\ b\bar{b}\ c\bar{c}\ m{\cdot}s$	$w\ b\bar{b}\ c\bar{c}\ m{\cdot}s$
initial state	$0\ 11\ 00\ 1{\cdot}0$	$0\ 11\ 00\ 1{\cdot}1$
float $bit/\overline{bit}$	$0\ \tilde{1}\tilde{1}\ 00\ 1{\cdot}0$	$0\ \tilde{1}\tilde{1}\ 00\ 1{\cdot}1$
raise *WL*	$\mathbf{0}\ \tilde{1}\tilde{1}\ 00\ 1{\cdot}0$	$\mathbf{0}\ \tilde{1}\tilde{1}\ 00\ 1{\cdot}1$
bit or $\overline{bit}$ discharges	$\mathbf{0}\ \tilde{\mathbf{1}}\tilde{1}\ 00\ 1{\cdot}0$	$\mathbf{0}\ \tilde{1}\tilde{\mathbf{1}}\ 00\ 1{\cdot}1$
read $bit/\overline{bit}$ & lower *WL*	$0\ \tilde{\mathbf{1}}\tilde{1}\ 00\ 1{\cdot}0$	$0\ \tilde{1}\tilde{\mathbf{1}}\ 00\ 1{\cdot}1$
raise $bit/\overline{bit}$	$0\ 11\ 00\ 1{\cdot}0$	$0\ 11\ 00\ 1{\cdot}1$

Write operations $(s = 0)$		
	W-1	
Description	$w\ b\bar{b}\ c\bar{c}\ m{\cdot}s$	
initial state	$0\ 11\ 00\ 1{\cdot}0$	
set $bit/\overline{bit}$	$0\ 10\ 00\ 1{\cdot}0$	
raise *WL*	$\mathbf{0}\ 10\ 00\ 1{\cdot}0$	
new state	$\mathbf{0}\ 10\ 00\ 1{\cdot}\mathbf{0}$	
lower *WL*	$0\ 10\ 00\ 1{\cdot}\mathbf{0}$	
raise $bit/\overline{bit}$	$0\ 11\ 00\ 1{\cdot}\mathbf{0}$	

Write operations $(s = 1)$		
	W-0	
Description	$w\ b\bar{b}\ c\bar{c}\ m{\cdot}s$	
initial state	$0\ 11\ 00\ 1{\cdot}1$	
set $bit/\overline{bit}$	$0\ 01\ 00\ 1{\cdot}1$	
raise *WL*	$\mathbf{0}\ 01\ 00\ 1{\cdot}1$	
new state	$\mathbf{0}\ 01\ 00\ 1{\cdot}\mathbf{1}$	
lower *WL*	$0\ 01\ 00\ 1{\cdot}\mathbf{1}$	
raise $bit/\overline{bit}$	$0\ 11\ 00\ 1{\cdot}\mathbf{1}$	

Lack of differential voltages on the bit lines yields an unpredictable response of the sense circuitry. Therefore, a 'read' operation is not a reliable source of output. Since *WL* is never asserted, 'write 1' from state 0 and 'write 0' from state 1 are also affected, as they do not result in a transition to the desired state. 'Write 0' from state 0, 'write 1' from state 1 and 'write ×' operations, are not affected, as they effectively "do nothing".

***WL-sa-1* (*w-sa-1*) fault.** As the reader can verify using Table 1, in the presence of this fault, although there are minor differences in the low-level model, all operations are correct in the high-level model of a single cell.

***cbl-sa-0* (*c-sa-0*) fault.** Again, this fault has no effect on 'read' and 'write' operations. As indicated in Table 7, this fault affects only the 'compare 1' operation when the cell is in state 0. The fault prevents *ML* from discharging resulting in an erroneous match.

Table 7: CAM cell operation with *cbl-sa-0* fault.

Compare operations ($s = 0$)	
	C-1
Description	$w\ b\bar{b}\ c\bar{c}\ m\cdot s$
initial state	$0\ 11\ 00\ 1\cdot 0$
float *ML*	$0\ 11\ 00\ \tilde{1}\cdot 0$
set $cbl/\overline{cbl}$	$0\ 11\ \mathbf{0}0\ \tilde{1}\cdot 0$
ML discharges	$0\ 11\ \mathbf{0}0\ \tilde{\mathbf{1}}\cdot 0$
read *ML* & ground $cbl/\overline{cbl}$	$0\ 11\ 00\ \tilde{\mathbf{1}}\cdot 0$
raise *ML*	$0\ 11\ 00\ 1\cdot 0$

Analogous behavior is exhibited in the presence of the $\overline{cbl}$*-sa-0* fault.

***cbl-sa-1* (*c-sa-1*) fault.** This fault has no effect on 'read' and 'write' operations. As indicated in Table 8, this fault only affects 'compare 0 ' and 'compare ×' operations when the cell is in state 0. In this state the initial value of the *ML* is *I*, which is the result of a conducting match line discharge transistor when *ML* is driven high by the peripheral circuitry. In effect, this fault may also manifest itself by an increased quiescent power supply current. Once *ML* is floated, it is immediately discharged by the faulty line.

Analogous behavior is exhibited in the presence of the $\overline{cbl}$*-sa-1* fault.

Table 8: CAM cell operation with *cbl-sa-1* fault.

Compare operations ($s = 0$)		
	C-0	C-×
Description	$w\ b\bar{b}\ c\bar{c}\ m{\cdot}s$	$w\ b\bar{b}\ c\bar{c}\ m{\cdot}s$
initial state	0 11 **10** **I**·0	0 11 **10** **I**·0
float *ML*	0 11 **10** **0**·0	0 11 **10** **0**·0
set $cbl/\overline{cbl}$	0 11 **11** **0**·0	0 11 **10** **0**·0
ML discharges	0 11 **11** **0**·0	0 11 **10** **0**·0
read *ML* & ground $cbl/\overline{cbl}$	0 11 **10** **0**·0	0 11 **10** **0**·0
raise *ML*	0 11 **10** **I**·0	0 11 **10** **I**·0

***ML-sa-0* (*m-sa-0*) fault.** Table 9 describes the faulty behaviors of a CAM cell in the presence of a *ML-sa-0* fault. This fault does not affect the behavior of the cell during 'read' and 'write' operations. It does, however, affect 'compare' operations by returning an unconditional mismatch.

Table 9: CAM cell operation with *ML-sa-0* fault.

Compare operations ($s = 0$)		
	C-0	C-×
Description	$w\ b\bar{b}\ c\bar{c}\ m{\cdot}s$	$w\ b\bar{b}\ c\bar{c}\ m{\cdot}s$
initial state	0 11 00 **0**·0	0 11 00 **0**·0
float *ML*	0 11 00 **0**·0	0 11 00 **0**·0
set $cbl/\overline{cbl}$	0 11 01 **0**·0	0 11 00 **0**·0
ML discharges	0 11 01 **0**·0	0 11 00 **0**·0
read *ML* & ground $cbl/\overline{cbl}$	0 11 00 **0**·0	0 11 00 **0**·0
raise *ML*	0 11 00 **0**·0	0 11 00 **0**·0

Compare operations ($s = 1$)		
	C-1	C-×
Description	$w\ b\bar{b}\ c\bar{c}\ m{\cdot}s$	$w\ b\bar{b}\ c\bar{c}\ m{\cdot}s$
initial state	0 11 00 **0**·1	0 11 00 **0**·1
float *ML*	0 11 00 **0**·1	0 11 00 **0**·1
set $cbl/\overline{cbl}$	0 11 10 **0**·1	0 11 00 **0**·1
ML discharges	0 11 10 **0**·1	0 11 00 **0**·1
read *ML* & ground $cbl/\overline{cbl}$	0 11 00 **0**·1	0 11 00 **0**·1
raise *ML*	0 11 00 **0**·1	0 11 00 **0**·1

***ML-sa-1* (*m-sa-1*) fault.** Table 10 describes the faulty behaviors of a CAM cell in the presence of a *ML-sa-1* fault. As expected, this fault has no bearing on the 'read' and 'write' operations, but it does affect 'compare' operations by returning an unconditional match.

Table 10: CAM cell operation with *ML-sa-1* fault.

Compare operations ($s = 0$)	
	C-1
Description	$w\ b\bar{b}\ c\bar{c}\ m{\cdot}s$
initial state	0 11 00 1·0
float *ML*	0 11 00 **1**·0
set $cbl/\overline{cbl}$	0 11 10 **1**·0
ML discharges	0 11 10 **1**·0
read *ML* & ground $cbl/\overline{cbl}$	0 11 00 **1**·0
raise *ML*	0 11 00 1·0

Compare operations ($s = 1$)	
	C-0
Description	$w\ b\bar{b}\ c\bar{c}\ m{\cdot}s$
initial state	0 11 00 1·1
float *ML*	0 11 00 **1**·1
set $cbl/\overline{cbl}$	0 11 01 **1**·1
ML discharges	0 11 01 **1**·1
read *ML* & ground $cbl/\overline{cbl}$	0 11 00 **1**·1
raise *ML*	0 11 00 1·1

B Verification of the T_{isaf} Test

In this section the T_{isaf} test for a single cell is verified. In Table 11 the fault-free response is provided for comparison with the faulty responses of all independently testable faults. The deviations from the fault-free response are indicated in boldface. Tests for individual faults that are included in T_{isaf} are also given.

In all cases except *w-sa-0* the initial state of the cell is not important because, after the execution of the first three write operations $w_0w_1w_\times$, the initial state is forgotten.

Tests for some of the faults (ex. *b-sa-1*: $w_1w_0c_0$) appear separated by other operations. These interleaved operations are combinations of $w_\times$, c_0 and c_1 which do not change the intended state of the cell and, therefore, do not invalidate these tests.

The T_{isaf} test is *irredundant*, which means that the removal of any one of the operations that make up T_{isaf} will prevent some fault from being detected.

Example: If the first w_0 is dropped, then *$\bar{b}$-sa-1* fault is no longer detectable.

Table 11: Verification of the T_{isaf} test.

Fault	T_{isaf}	Test Sequence for Faults within T_{isaf}
	$w_0w_1w_\times c_1c_0w_0w_\times c_0c_1$	
fault-free cell	$- \; - \; - \; 10 \; - \; - \; 10$	
b-sa-0	$- \; - \; - \; \mathbf{01} \; - \; - \; 10$	$w_0\mathbf{w_1}\mathbf{w}_\times\mathbf{c_1}c_0w_0w_\times c_0c_1$
b-sa-1	$- \; - \; - \; 10 \; - \; - \; \mathbf{01}$	$w_0\mathbf{w_1}w_\times c_1c_0\mathbf{w_0}w_\times\mathbf{c_0}c_1$
$\bar{b}$-sa-0	$- \; - \; - \; 10 \; - \; - \; \mathbf{01}$	$w_0w_1w_\times c_1c_0\mathbf{w_0}\mathbf{w}_\times\mathbf{c_0}c_1$
$\bar{b}$-sa-1	$- \; - \; - \; \mathbf{01} \; - \; - \; 10$	$\mathbf{w_0}\mathbf{w_1}w_\times\mathbf{c_1}c_0w_0w_\times c_0c_1$
w-sa-0 (s=0)	$- \; - \; - \; \mathbf{01} \; - \; - \; 10$	$w_0\mathbf{w_1}w_\times\mathbf{c_1}c_0\mathbf{w_0}w_\times\mathbf{c_0}c_1$
w-sa-0 (s=1)	$- \; - \; - \; 10 \; - \; - \; \mathbf{01}$	″
c-sa-0	$- \; - \; - \; 10 \; - \; - \; 11$	$w_0w_1w_\times c_1c_0\mathbf{w_0}w_\times c_0\mathbf{c_1}$
c-sa-1	$- \; - \; - \; 10 \; - \; - \; 00$	$w_0w_1w_\times c_1c_0\mathbf{w_0}w_\times\mathbf{c_0}c_1$
$\bar{c}$-sa-0	$- \; - \; - \; 11 \; - \; - \; 10$	$w_0\mathbf{w_1}w_\times c_1\mathbf{c_0}w_0w_\times c_0c_1$
$\bar{c}$-sa-1	$- \; - \; - \; \mathbf{00} \; - \; - \; 10$	$w_0\mathbf{w_1}w_\times\mathbf{c_1}c_0w_0w_\times c_0c_1$
m-sa-0	$- \; - \; - \; \mathbf{00} \; - \; - \; \mathbf{00}$	$w_0w_1w_\times\mathbf{c_1}\mathbf{c_0}w_0w_\times c_0c_1$
m-sa-1	$- \; - \; - \; 11 \; - \; - \; 11$	$w_0w_1w_\times\mathbf{c_1}\mathbf{c_0}w_0w_\times c_0c_1$

C Analysis of $T_{G\&H}$

For the sake of simplicity we assume an n-word by l-bit CAM where $n = 2^l$. In this manner, the address of word (row index) is used as the unique bit-pattern. Whenever necessary, we comment on CAMs where $n < 2^l$, as the unique bit-pattern constraint is also satisfied in this case.

We can represent $T_{G\&H}$ from the perspective of an arbitrary cell located at coordinates (i, j) in the array where $1 \leq i \leq n$ and $0 \leq j \leq l-1$. Variable p is a row index (and, coincidentally, the decimal representation of the word content).

$$
\begin{aligned}
T^{i,j}_{G\&H} \quad = \quad & w^{i,j}_{((i-1)\,\mathrm{div}\,2^j)\bmod 2}\;[c^j_{(p\,\mathrm{div}\,2^j)\bmod 2}]^0_{p=n-1} && (1)\\
& w^{i,j}_{1-(((i-1)\,\mathrm{div}\,2^j)\bmod 2)}\;[c^j_{1-((p\,\mathrm{div}\,2^j)\bmod 2)}]^0_{p=n-1} && (2)\\
& w^{i,j}_{((i-1)\,\mathrm{div}\,2^j)\bmod 2}\;[c^j_{(p\,\mathrm{div}\,2^j)\bmod 2}]^{n-1}_{p=0} && (3)\\
& w^{i,j}_0\;[c^j_{I[p,j]}]^{l-1}_{p=0} && (4)\\
& w^{i,j}_1\;[c^j_{1-I[p,j]}]^{l-1}_{p=0} && (5)
\end{aligned}
$$

where $I[\,]$ is a $n \times l$ identity matrix.

We have shown in Section 4 that 'compare' operations, even when faulty, do not affect the state of the cell, so interleaving 'write' operations with arbitrary number of 'compare' operations will not influence any state transitions resulting from 'write' operations. Each cell (i, j), therefore, is subject to one of the following two sequences of write operations:

$$w_0 w_1 w_0 w_0 w_1 \quad \text{or} \quad w_1 w_0 w_1 w_0 w_1.$$

In the case of $n = 2^l$, each of the initial three 'write' operations are followed by $n/2$ 'compare 0' operations and $n/2$ 'compare 1' operations. The order of 'compare' operations depends on the column j in which the given cell is located, but can be treated as arbitrary. If $n < 2^l$, in the worst case, each of the initial three 'write' operations will only be followed by a sequence of "matching" 'compare' operations, i.e., a $w_0^{i,j}$ would precede a sequence of c_0^j's or a $w_1^{i,j}$ would precede a sequence of c_1^j's. The last two 'write' operations are always followed by $l-1$ "mismatching" 'compare' operations, where a w_0 precedes a c_1 and vice-versa, and a single "matching" 'compare' operation, for all possible values of n and l. The order of occurrence of the single "matching" 'compare' operation depends on the column j in which the given cell is located.

C.1 Proofs of fault detection

The proofs presented below indicate that $T_{G\&H}^{i,j}$, when applied to an arbitrary single CAM cell, will not detect any of the b^j-*sa-0* and $\bar{b}^j$-*sa-0* faults, where $0 \leq j \leq l-1$. We, therefore, conclude that these faults will not be detected by $T_{G\&H}$ when it is applied to an n-word by l-bit CAM. All other input stuck-at faults are detected.

b-sa-0 According to Table 2, a test that detects this fault is $w_1 w_\times c_1$. The sequence $w_1^{i,j} w_\times^{i,j}$ does not occur in $T_{G\&H}^{i,j}$. Since after the initial $w_1^{i,j}$ the cell finds itself in an indeterminate state, none of the subsequent 'compare' operations can result in a dependable output.

The reader can verify that a similar reasoning holds for the symmetric fault $\bar{b}$-*sa-0*.

b-sa-1 According to Table 2, a test that detects this fault is $w_1 w_0 c_0$. The sequence $w_1^{i,j} w_0^{i,j}$ occurs during the execution of (1) and (2), or (2) and (3) in $T_{G\&H}^{i,j}$. Each $w_0^{i,j}$ is guaranteed to be followed by at least one c_0^j.

A similar reasoning holds for the symmetric fault $\bar{b}$-*sa-1*.

w-sa-0 According to Table 2, a test that detects this fault is $w_1c_1w_0c_0$. The sequence $w_1^{i,j}w_0^{i,j}$ occurs during the execution of (2) and (3), or (3) and (4) in $T^{i,j}_{G\&H}$. Each $w_1^{i,j}$ is guaranteed to be followed by at least one c_1^j; each $w_0^{i,j}$ is guaranteed to be followed by at least one c_0^j.

w-sa-1 Assume that there are no duplicate words in the array. Then there exists a column j where two distinct cells (i', j) and (i'', j) will have complementary values written to them. Suppose that cell (i', j) is written to first. If row i' is faulty, then during the 'write' operation to cell (i'', j), the cell (i', j) will be overwritten to a complementary and erroneous value. The erroneous value will be detected by subsequent 'compare' operations. This reasoning is symmetric, as 'write' operations in $T_{G\&H}$ are performed in both descending and ascending order.

c-sa-0 According to Table 2, a test that detects this fault is w_0c_1. This test occurs in (4) of $T^{i,j}_{G\&H}$, as the $w_0^{i,j}$ is guaranteed to be followed by a c_1^j.

A similar reasoning holds for the symmetric fault $\bar{c}$*-sa-0*.

c-sa-1 According to Table 2, a test that detects this fault is w_0c_0. A $w_0^{i,j}$ occurs in either (1) or (2) of $T^{i,j}_{G\&H}$. Each $w_0^{i,j}$ is guaranteed to be followed by at least one c_0^j.

A similar reasoning holds for the symmetric fault $\bar{c}$*-sa-1*.

m-sa-0* and *m-sa-1 Assume that there are no duplicate words in the array. According to Table 2, tests that detect these faults are c_1c_0 or c_0c_1. Either $c_1^jc_0^j$ or $c_0^jc_1^j$ occurs in (4) or (5) of $T^{i,j}_{G\&H}$ after $w_0^{i,j}$ or $w_1^{i,j}$, respectively. In the presence of a duplicate word, however, any *m-sa-0* fault would be masked by a match on the fault-free duplicate.

D Analysis of $T_{C\text{-}SM}$

In the following analysis we assume an n-word by l-bit CAM. We can represent $T_{C\text{-}SM}$ from the perspective of an arbitrary cell located at coordinates (i, j) in the array where $1 \leq i \leq n$ and $0 \leq j \leq l - 1$. The subscripts of 'write' and 'compare' operations represent single bit values, where '$-$' stands for an arbitrary value used during initialization. The subscripts of 'read' operations denote the expected output of the operation.

$$T^{i,j}_{C\text{-}SM} = \begin{bmatrix} r^{i,j}_{-} w^{i,j}_{0} \\ c^{j}_{-}\ c^{j}_{-} \end{bmatrix}^{((l-1)-j)} \begin{bmatrix} r^{i,j}_{-} w^{i,j}_{0} \\ c^{j}_{-}\ c^{j}_{-} \end{bmatrix} \begin{bmatrix} r^{i,j}_{0} w^{i,j}_{0} \\ c^{j}_{0}\ c^{j}_{0} \end{bmatrix}^{(j+1)} \tag{6}$$

$$\begin{bmatrix} r^{i,j}_{0} w^{i,j}_{0} \\ c^{j}_{0}\ c^{j}_{0} \end{bmatrix}^{((l-1)-j)} \begin{bmatrix} r^{i,j}_{0} w^{i,j}_{1} \\ c^{j}_{1}\ c^{j}_{1} \end{bmatrix} \begin{bmatrix} r^{i,j}_{1} w^{i,j}_{1} \\ c^{j}_{1}\ c^{j}_{1} \end{bmatrix}^{(j+1)} \tag{7}$$

$$\begin{bmatrix} r^{i,j}_{1} w^{i,j}_{1} \\ c^{j}_{1}\ c^{j}_{1} \end{bmatrix}^{((l-1)-j)} \begin{bmatrix} r^{i,j}_{1} w^{i,j}_{0} \\ c^{j}_{0}\ c^{j}_{0} \end{bmatrix} \begin{bmatrix} r^{i,j}_{0} w^{i,j}_{0} \\ c^{j}_{0}\ c^{j}_{0} \end{bmatrix}^{(j+1)} \tag{8}$$

$$\begin{bmatrix} r^{i,j}_{0} w^{i,j}_{0} \\ c^{j}_{0}\ c^{j}_{0} \end{bmatrix}^{((l-1)-j)} \begin{bmatrix} r^{i,j}_{0} w^{i,j}_{1} \\ c^{j}_{1}\ c^{j}_{1} \end{bmatrix} \begin{bmatrix} r^{i,j}_{1} w^{i,j}_{1} \\ c^{j}_{1}\ c^{j}_{1} \end{bmatrix}^{(j+1)} \tag{9}$$

$$\begin{bmatrix} r^{i,j}_{1} w^{i,j}_{1} \\ c^{j}_{1}\ c^{j}_{1} \end{bmatrix}^{((l-1)-j)} \begin{bmatrix} r^{i,j}_{1} w^{i,j}_{0} \\ c^{j}_{0}\ c^{j}_{0} \end{bmatrix} \begin{bmatrix} r^{i,j}_{0} w^{i,j}_{0} \\ c^{j}_{0}\ c^{j}_{0} \end{bmatrix}^{(j+1)} \tag{10}$$

$$\begin{bmatrix} r^{i,j}_{0} w^{i,j}_{0} \\ c^{j}_{0}\ c^{j}_{0} \end{bmatrix}^{((l-1)-j)} \begin{bmatrix} r^{i,j}_{0} w^{i,j}_{0} \\ c^{j}_{0}\ c^{j}_{0} \end{bmatrix} \begin{bmatrix} r^{i,j}_{0} w^{i,j}_{0} \\ c^{j}_{0}\ c^{j}_{0} \end{bmatrix}^{(j+1)} \tag{11}$$

D.1 Proofs of fault detection

The proofs presented below indicate that $T^{i,j}_{C\text{-}SM}$ when applied to an arbitrary single CAM cell will detect all faults in the input stuck-at fault model. We, therefore, conclude that these faults will also be detected by $T_{C\text{-}SM}$ when it is applied to a n-word by l-bit CAM.

b-sa-0 According to Table 2, a test that detects this fault is $w_1 r$. The $w^{i,j}_1 r^{i,j}_1$ sequence occurs in (7) of $T^{i,j}_{C\text{-}SM}$. Although after the initial $w^{i,j}_1$ the cell's state is indeterminate, the subsequent $r^{i,j}_1$ forces the cell to a determinate, erroneous state 0 and returns this erroneous value.

The reader can verify that a similar reasoning holds for the symmetric fault *$\bar{b}$-sa-0*.

b-sa-1 According to Table 2, a test that detects this fault is $w_1 w_0 r$. In the presence of this fault, 'read' operations do not affect the cell's state. The sequence $w^{i,j}_1 w^{i,j}_0$ occurs during the execution of (7) and (8) in $T^{i,j}_{C\text{-}SM}$. Each $w^{i,j}_0$ is guaranteed to be followed by at least one $r^{i,j}_0$.

A similar reasoning holds for the symmetric fault *$\bar{b}$-sa-1*.

w-sa-0 In the presence of this fault 'read' operations do not provide a reliable output; they do not, however, affect the cell's state. According to Table 2, a test that detects this fault is $w_1c_1w_0c_0$. The sequence $w_1^{i,j}w_0^{i,j}$ occurs during the execution of (7) and (8) in $T^{i,j}_{C\text{-}SM}$. Although $c_1^jc_0^j$ is performed "in parallel", these 'compare' operations actually occur after the 'write' is completed, and thus they are applied to the newly written value; hence the required test occurs in $T^{i,j}_{C\text{-}SM}$.

w-sa-1 Since the array is "filled" serially, there exists a column j where two distinct cells (i', j) and (i'', j) will hold complementary values at some point during each march element. Suppose that $i' < i''$ and a march element is being executed in the ascending order. The cell (i', j) is written to first. If row i'' is faulty, then during the 'write' operation to cell (i', j), the cell (i'', j) will be overwritten prematurely to a complementary and erroneous value. Since all march elements begin with a 'read' operation, the value obtained from cell (i'', j) during that initial 'read' will differ from the expected value. This reasoning is symmetric, as march elements in $T_{C\text{-}SM}$ are performed in both descending and ascending order.

c-sa-0 According to Table 2, a test that detects this fault is w_0c_1. This test occurs during the execution of (6) and (7) in $T^{i,j}_{C\text{-}SM}$, before the first $w_1^{i,j}$. During (6) the cell is subject to a $w_0^{i,j}$. During (7), a c_1^j occurs concurrently with the $r_0^{i,j}$, that precedes the first $w_1^{i,j}$. Since in the presence of this fault 'read' operations do not affect the state of the cell, the required test occurs in $T^{i,j}_{C\text{-}SM}$.

A similar reasoning holds for the symmetric fault $\bar{c}$*-sa-0*.

c-sa-1 According to Table 2, a test that detects this fault is w_0c_0. A $w_0^{i,j}$ is certain to occur in (6), (8), (10) and (11) of $T^{i,j}_{C\text{-}SM}$. Each $w_0^{i,j}$ is guaranteed to be followed by one c_0^j during the same clock cycle.

A similar reasoning holds for the symmetric fault $\bar{c}$*-sa-1*.

m-sa-0* and *m-sa-1 Since the array is "filled" serially, each of the rows in the array holds a unique value some point during the execution of a march element. A comparison with an identical key-word that results a single match is performed (See Table 3). According to Table 2, tests that detect these faults are c_1c_0 or c_0c_1 while the state of the cell remains unchanged. The sequence $c_0^jc_1^j$ occurs during the execution of (6) and (7), and during the execution of (8) and (9) in $T^{i,j}_{C\text{-}SM}$ concurrently with $w_0^{i,j}r_0^{i,j}$ that immediately precedes a $w_1^{i,j}$. Symmetrically, the sequence $c_1^j c_0^j$ occurs during the execution of (7) and (8) and during the execution of (9) and (10) in $T_{C\text{-}SM}$ concurrently with $w_1^{i,j} r_1^{i,j}$ that immediately precedes a $w_0^{i,j}$.