Verification of CAM Tests for Input Stuck-at Faults*

Piotr R. Sidorowicz Janusz A. Brzozowski
Department of Computer Science
University of Waterloo
Waterloo, ON, Canada, N2L 3G1

Abstract

A comparison of three tests applied to an n-word by l-bit static CMOS content-addressable memory (CAM) array is performed, with respect to the cell input stuck-at fault model. We briefly review the methodology facilitating this comparison, and the proof of correctness of the Sidorowicz & Brzozowski test, of length $7n + 2l + 5$, which was derived using this methodology. Then, we examine the Giles & Hunter test and demonstrate that it fails to detect bit-sa-0 and $\overline{bit}$-sa-0 faults. We also show how this test can be modified to a test, of length $11n + 2l$, that detects these faults. Next, we verify that the Kornachuk et al. test, which is of length $24nl$ and is used in BIST, detects all the faults in the fault model.

1. Introduction

Although the concept of a content-addressable memory (CAM) has been known for almost thirty years, only in the last decade have CAMs begun appearing in mainstream integrated circuit designs. As their utilization grows, so does the issue of generating reliable and efficient tests for these memories.

Several algorithms for testing CAMs have been reported [4, 8, 9, 11]. The tests in [4, 8, 11] are applicable to CAM designs which incorporate an explicit word addressing scheme of the type found in conventional RAMs. The test of [9] imposes extra requirements on the addressing scheme; thus it is not universally applicable, and will not be considered further.

A systematic approach to modeling and testing memories has been presented in [11]. It includes circuit analysis, asynchronous behavior analysis, and a finite automaton representation of the fault-free and faulty circuits. The basis for this approach to modeling and testing memories is the framework of [2]. This framework has been successfully utilized for modeling common faults in RAMs (See [3, 6, 7], for examples).

The remainder of this report is organized as follows: The CAM cell and its model are presented in Section 2. Section 3 describes the event-sequence model of a cell's behavior. A higher level, FSM model of a cell's behavior is introduced in Section 4. Tests by Sidorowicz & Brzozowski, Giles & Hunter, and Kornachuk et al. are described in Sections 5, 6, and 7 respectively. Section 8 concludes the paper. Proofs of fault detection for the tests by Giles & Hunter and Kornachuk et al. are presented in Appendices A and B respectively.

2. CAM Circuit and Model

Figure 1 shows a cell of a common static CAM circuit [1, 8]. It consists of a *storage section* and a *comparison*

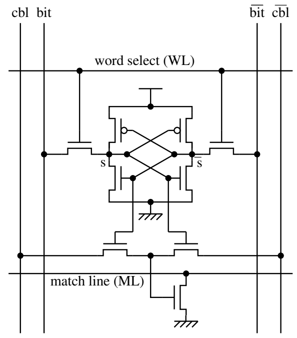

Figure 1. Static CAM cell with dedicated compare lines.

section. The storage section has 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

*This research was supported by the Natural Sciences and Engineering Research Council of Canada under grant No. OGP0000871.

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. The 'write $\times$' operation preserves the state of the cell. Its 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. A 'read' operation is done by isolating both *bit*/$\overline{bit}$ lines, and then raising *WL*, thus causing one of the *bit*/$\overline{bit}$ lines to discharge. The voltage differential between the *bit*/$\overline{bit}$ lines is detected giving the cell's content.

The comparison section consists of the bottom three transistors, the differential compare bit lines (*cbl*/$\overline{cbl}$) for 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 $\times$' operation, both of *cbl*/$\overline{cbl}$ remain low, resulting in an unconditional match.

Altogether seven operations can be performed on this cell: 'read', 'write 0', 'write 1', 'write $\times$', 'compare 0', 'compare 1' and 'compare $\times$'. The *input stuck-at fault model* [11] consists of stuck-at-0 and stuck-at-1 faults on the six lines: *bit*, $\overline{bit}$ (bit lines), *cbl*, $\overline{cbl}$ (compare-bit lines), *WL* (word select line) and *ML* (match line), under the single fault assumption.

3. The Event-Sequence Model

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 Fig. 2. 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, $\bar{b}$ w, c, $\bar{c}$ and m are inputs, and $\widehat{b}$, $\widehat{\bar{b}}$, and $\widehat{m}$ are outputs. The total state of the cell is defined by the values on the input and output lines of the cell, and by its internal state s. 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. Thus the total state of the cell is formally represented by the 7-tuple: $C = (w, b, \bar{b}, c, \bar{c}, m, s)$, but for notational simplicity, we represent it as $C = w\ b\bar{b}\ c\bar{c}\ m \cdot s$, where the input variables have been separated by spaces for readability, and the $\cdot$ has been inserted to separate the input variables from the internal state.

The domain of each variable in 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

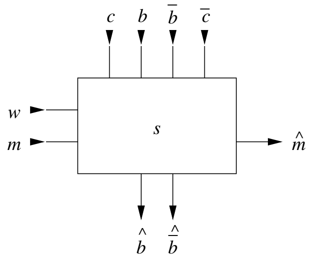

Figure 2. Static CAM cell model.

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 conduct.

A correct CAM cell has two possible initial states: $0\ 11\ 00\ 1 \cdot 0$ and $0\ 11\ 00\ 1 \cdot 1$.

Table 1. A 'write 0' in a correct and faulty CAM cell.

Write **0** ($s = 1$)		
	Correct	$\bar{b}$-*sa-0*
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 1·1	0 1**0** 00 1·1
set *bit*/$\overline{bit}$	0 01 00 1·1	0 0**0** 00 1·1
raise *WL*	1 01 00 1·1	1 0**0** 00 1·1
new state	1 01 00 1·0	1 0**0** 00 1·**I**
lower *WL*	0 01 00 1·0	0 0**0** 00 1·**I**
raise *bit*/$\overline{bit}$	0 11 00 1·0	0 1**0** 00 1·0|**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 a 'write 0' operation in a correct and a faulty cell whose state s is 1. 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 the top and bottom entries in each column are initial states. Faulty entries are shown in boldface type.

Example: 'Write 0' operation in a correct cell and in the presence of the $\bar{b}$-*sa-0* fault. Initial state of the cell: $0\ 11\ 00\ 1 \cdot 1$ (correct), $0\ 1\mathbf{0}\ 00\ 1 \cdot 1$ (faulty). First, b is lowered: $0\ 01\ 00\ 1 \cdot 1$ (correct), $0\ 0\mathbf{0}\ 00\ 1 \cdot 1$ (faulty). Note that the both bit lines in the faulty cell are low. Then w is raised: $1\ 01\ 00\ 1 \cdot 1$ (correct), $1\ 0\mathbf{0}\ 00\ 1 \cdot 1$ (faulty). As a result, s changes to 0 in the correct cell: $1\ 01\ 00\ 1 \cdot 0$. In the faulty cell, since both b and $\bar{b}$ are low, s becomes metastable: $1\ 0\mathbf{0}\ 00\ 1 \cdot \mathbf{I}$. Next, w is lowered: $0\ 01\ 00\ 1 \cdot 0$ (correct), $0\ 0\mathbf{0}\ 00\ 1 \cdot \mathbf{I}$ (faulty). Finally, both b and $\bar{b}$ are raised: $0\ 11\ 00\ 1 \cdot 0$ (correct), $0\ 1\mathbf{0}\ 00\ 1 \cdot 0|\mathbf{1}$ (faulty); in the faulty cell, s eventually becomes 0 or 1.

An exhaustive analysis of all operations is given in [12]. 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 the presence of a fault. Unfortunately, this detail of monitoring is not feasible; thus, a more abstract model was necessary.

4. The FSM 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, derived from the "event-sequence" model. We represent the behavior CAM cell as a finite-state machine (FSM), which is shown in Fig. 3. The input x

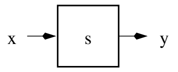

Figure 3. Simplified behavioral model.

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 $b/\bar{b}$ lines or m. State s represents the value stored by the cell. An automaton model for all input stuck-at faults has been constructed. An example is shown in Fig. 4.

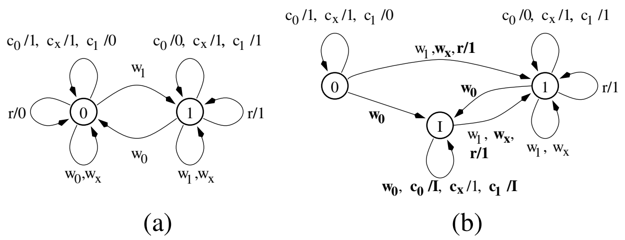

Figure 4. Cell behavior: (a) correct (b) $\bar{b}$-*sa-0*.

5. Sidorowicz & Brzozowski Test

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. 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 *w-sa-1* fault automaton is identical to that of the fault-free CAM cell; therefore, no test exists for a single cell. This is the only input stuck-at fault that is not independently testable. However, it is detectable in conjunction with cells that belong to other words in the array. 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*. Partly with the aid of the OBSERVER program, we have found a test

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

of length 9 that detects all independently testable input stuck-at faults in a single cell.

This test was extended to a march test $T_{S\&B}$, shown below, for an $(n \times l)$ CAM [11, 12]. Two assumptions were made about the peripheral circuitry of the fault-free CAM: 1) The address decoder can raise at most a single write line, 2) Match detection is done through a *hit* line, a *multi-hit* line, and an encoder that returns the address k of the highest priority match line[2].

$$\begin{aligned} T_{S\&B} = \\ & (w^i_{0\ldots0})^{\updownarrow}(w^i_{1\ldots1} w^i_{\times\ldots\times} c_{1\ldots1})^{n\downarrow1}[c_{\times^{j-1}0\times^{l-j}}]^{\leftrightarrow} \\ & (w^i_{0\ldots0} w^i_{\times\ldots\times} c_{0\ldots0})^{n\downarrow1}[c_{\times^{j-1}1\times^{l-j}}]^{\leftrightarrow} \\ & (w^n_{1\ldots1} c_{1\ldots1} w^n_{0\ldots0} w^u_{1\ldots1} c_{1\ldots1}), \end{aligned}$$

where $0\ldots0$ denotes the all-0 word, etc., $\times^{j-1}0\times^{l-j}$ denotes 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), and $1 \le u \le n-1$. 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 symbol $(\ )^{\updownarrow}$ denotes n operations done in order either from n to 1 or from 1 to n; hence the $\updownarrow$. The symbol $(\ )^{n\downarrow1}$ 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.

This test of length $7n+2l+5$, detects 100% of the faults with respect to the input stuck-at fault model. It has two parts. 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 $k=i$, where k is the output of the match line priority encoder. 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.

In this paper we utilize the above approach to determine the fault coverage of the tests reported in [4] and [8].

[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 added by P. Kwiatkowski and P. R. Sidorowicz.

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

6. Giles & Hunter Test

One of the first tests for CAMs, which we call $T_{G\&H}$, of length $8n + 2l$, was presented by Giles and Hunter [4]. 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 cell of Fig. 1 to determine the fault coverage with respect to our input stuck-at fault model. 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. We represent this test using our notation. Since we index the words in our CAM from 1 to n, (1 being the top, and n the bottom of the array) the value of k written into each word i is equal to $i - 1$; $\bar{k}$ denotes the 1's complement of k.

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

The analysis of the $T_{G\&H}$ test in our automaton model of input stuck-at faults shows that, in the cell of Fig. 1, $T_{G\&H}$ does not reliably detect the *b-sa-0* and $\bar{b}$*-sa-0* faults. This is due to a metastable state caused by w_1 and w_0 operations in the presence of these two faults respectively; thus any subsequent output is not reliable. There are $2l$ such faults in an $(n \times l)$ 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 about 84% of faults are detected.

We next modified $T_{G\&H}$ to detect the *b-sa-0* and $\bar{b}$*-sa-0* faults. We have shown that, in the presence of these two faults, a 'write $\times$' operation forces the faulty cell to a determinate but erroneous state that is easily detectable. Therefore, by judiciously inserting 'write $\times$' 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 $11n + 2l$.

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

The details of our analysis are shown in Appendix A.

7. Kornachuk et al. Test

The test, which we call $T_{C\text{-}SM}$, presented by Kornachuk et al. [8] is a built-in self test (BIST) for embedded CAMs that use the cell of Fig. 1. It is an adaptation of the SMARCH test, of length $24nl$, originally developed for testing embedded SRAMs [10]. SMARCH has been shown to detect stuck-at and coupling faults. In fact, SMARCH is a serialized version of the 'March C-' test [5]. This serialization was necessary in order to utilize the scan-path facilities of the CAM circuit.

The adaptation of the SMARCH test to the CAM circuit was possible due to the dual-port nature of this particular CAM design, where ($b/\bar{b}$) lines are used only for the 'read' and 'write' operations, and the ($c/\bar{c}$) lines are 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; hence the length of the test remains $24nl$, although twice as many operations are actually executed. To be precise, 'compare' operations are performed concurrently with the 'read' operations, but they occur immediately after the completion of the 'write' operations; thus the newly stored value is compared.

$T_{C\text{-}SM}$ presented in [8] can be represented using our notation. Since the 'read', 'write' and 'compare' operations are performed on entire words, their subscripts represent the decimal equivalents of l-bit binary numbers. The subscripts of 'read' operations reflect the expected output of these operations. The subscript '$-$' denotes an irrelevant value used in the initialization step of the test. The concurrent 'compare' operations are listed directly below the 'read' and 'write' operations.

$T_{C\text{-}SM} =$

$$\left(\begin{bmatrix} r^i_- w^i_0 \\ c_- c_- \end{bmatrix}^0_{j=l-1} \begin{bmatrix} r^i_0 w^i_0 \\ c_0 c_0 \end{bmatrix} \right)^{1\downarrow n} \tag{1}$$

$$\left(\begin{bmatrix} r^i_{(2^l-2^{(j+1)})} \; w^i_{(2^l-2^j)} \\ c_{(2^l-2^j)} \; c_{(2^l-2^j)} \end{bmatrix}^0_{j=l-1} \begin{bmatrix} r^i_{(2^l-1)} w^i_{(2^l-1)} \\ c_{(2^l-1)} c_{(2^l-1)} \end{bmatrix} \right)^{1\downarrow n} \tag{2}$$

$$\left(\begin{bmatrix} r^i_{(2^{(j+1)}-1)} \; w^i_{(2^j-1)} \\ c_{(2^j-1)} \; c_{(2^j-1)} \end{bmatrix}^0_{j=l-1} \begin{bmatrix} r^i_0 w^i_0 \\ c_0 c_0 \end{bmatrix} \right)^{1\downarrow n} \tag{3}$$

$$\left(\begin{bmatrix} r^i_{(2^l-2^{(j+1)})} \; w^i_{(2^l-2^j)} \\ c_{(2^l-2^j)} \; c_{(2^l-2^j)} \end{bmatrix}^0_{j=l-1} \begin{bmatrix} r^i_{(2^l-1)} w^i_{(2^l-1)} \\ c_{(2^l-1)} \; c_{(2^l-1)} \end{bmatrix} \right)^{n\uparrow 1} \tag{4}$$

$$\left(\begin{bmatrix} r^i_{(2^{(j+1)}-1)} \; w^i_{(2^j-1)} \\ c_{(2^j-1)} \; c_{(2^j-1)} \end{bmatrix}^0_{j=l-1} \begin{bmatrix} r^i_0 w^i_0 \\ c_0 c_0 \end{bmatrix} \right)^{n\uparrow 1} \tag{5}$$

$$\left(\begin{bmatrix} r_0^i w_0^i \\ c_0 c_0 \end{bmatrix}_{j=l-1}^{0} \begin{bmatrix} r_0^i w_0^i \\ c_0 c_0 \end{bmatrix} \right)^{n\uparrow 1} \quad (6)$$

In (1), the array is initially filled with 0s. In (2), starting with the leftmost bit in the top row, 1s are "shifted right" into the array, row by row, until the rightmost bit in the bottom row is 1. This procedure is repeated for shifting in 0s in (3). Steps (2) and (3) are repeated in (4) and (5), except the process starts from the bottom row. In (6) only the initial read is relevant; the array may be filled with arbitrary values. We assumed 0s for notational simplicity.

Since a 'read' always precedes a 'write', the expected output value reflects the previous value held by the cell. Note that in each step, at the end of every word, an additional 'read'-'write' is performed so that a newly written value to the rightmost bit can be verified with a 'read'. It is also important to note that the comparison word is identical to the one that is about to be written, and thus a 'compare' concurrent with a 'read' produces a mismatch, and a 'compare' concurrent with a 'write' produces a match. As multiple matches can be expected, a unique address is determined by a priority encoder.

We have formally verified that $T_{C\text{-}SM}$ detects 100% of the faults in the input stuck-at fault model. This is due to a mutually complementing detection abilities of the concurrent 'read' and 'compare' operations. For example, the *b-sa-0* and $\bar{b}$-sa-0 faults are detected by the erroneous output of a 'read' operation; this operation also forces the cell to an erroneous but determinate state. The concurrent 'compare' operation, in this case, yields an unreliable output. On the other hand, 'compare' operations are necessary for the detection of the *w-sa-0* faults, in the presence of which 'read' operations yield unreliable output. The details are shown in Appendix B.

8. Conclusions

Using the methods of [11] we have analyzed the fault coverage of two CAM tests with respect to our input stuck-at fault model for a CAM based on the cell of Fig. 1. We have shown that:

- The test of [11] is best for input stuck-at faults, as it is the shortest and provides 100% fault coverage with no restrictions on the size of the CAM.
- The test of [4] is longer, requires a modest restriction on the size of the CAM, and fails to detect certain faults. It can be modified to achieve 100% fault coverage, but then it is longer still.
- Finally, the test of [8] also provides 100% fault coverage with no restrictions on the size of the CAM; however, it is significantly longer. This length is dictated by the requirements of BIST.

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 Radu Negulescu and to all the 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, Aug. 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. v. d. Goor. *Testing Semiconductor Memories*. John Wiley & Sons, 1991.

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

[7] A. J. v. d. Goor and B. Smit. Automating the verification of memory tests. In *IEEE VLSI Test Symposium*, pages 312–318, Cherry Hill, NJ, Apr. 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 Workshop on Memory Technology, Design and Testing*, pages 40–45. IEEE, Aug. 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] B. Nadeau-Dostie, A. Silburt, and V. Agarval. Serial interfacing for embedded-memory testing. *IEEE Design & Test of Computers*, 7(2):56–64, Apr. 1990.

[11] P. R. Sidorowicz and J. A. Brzozowski. An approach to modeling and testing memories and its application to CAMs. In *IEEE VLSI Test Symposium*, pages 411–416. IEEE Computer Society Press, Apr. 1998.

[12] P. R. Sidorowicz and J. A. Brzozowski. Testing for input stuck-at faults in content-addressable memories. Technical Report CS-98-02, University of Waterloo, Apr. 1998. ftp://cs-archive.uwaterloo.ca/cs-archive/CS-98-02/CS-98-02.ps.Z.

A. Analysis of $T_{G\&H}$

For the sake of simplicity we assume a CAM where $n = 2^l$. In this manner, the word address 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, coincidently, in the first three steps of the test, the decimal representation of the word content).

$$T^{i,j}_{G\&H} =$$

$$w^{i,j}_{((i-1)\ \mathrm{div}\ 2^j)\ \mathrm{mod}\ 2}\ [c^j_{(p\ \mathrm{div}\ 2^j)\ \mathrm{mod}\ 2}]^{0}_{p=n-1} \tag{1}$$

$$w^{i,j}_{1-(((i-1)\ \mathrm{div}\ 2^j)\ \mathrm{mod}\ 2)}\ [c^j_{1-((p\ \mathrm{div}\ 2^j)\ \mathrm{mod}\ 2)}]^{0}_{p=n-1} \tag{2}$$

$$w^{i,j}_{((i-1)\ \mathrm{div}\ 2^j)\ \mathrm{mod}\ 2}\ [c^j_{(p\ \mathrm{div}\ 2^j)\ \mathrm{mod}\ 2}]^{n-1}_{p=0} \tag{3}$$

$$w^{i,j}_0\ [c^j_{I[p,j]}]^{l-1}_{p=0} \tag{4}$$

$$w^{i,j}_1\ [c^j_{1-I[p,j]}]^{l-1}_{p=0} \tag{5}$$

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

It has been shown [12] that 'compare' operations, even when faulty, do not affect the cell's state, so interleaving 'write' operations with any 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: $w_0w_1w_0w_0w_1$ or $w_1w_0w_1w_0w_1$. In the case of $n = 2^l$, each of the initial three 'write' operations is followed by $n/2$ c_0 operations and $n/2$ c_1 operations. The order of 'compare' operations depends on the column j in which the given cell is located, and is irrelevant to the test. 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 $w^{i,j}_0$ would precede a sequence of c^j_0's or a $w^{i,j}_1$ would precede a sequence of c^j_1'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.

The proofs presented below indicate that $T^{i,j}_{G\&H}$, 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 A test that detects this fault is $w_1w_\times c_1$. The sequence $w^{i,j}_1 w^{i,j}_\times$ does not occur in $T^{i,j}_{G\&H}$. Since after the initial $w^{i,j}_1$ 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 A test that detects this fault is $w_1w_0c_0$. The sequence $w^{i,j}_1 w^{i,j}_0$ occurs during the execution of (1) and (2), or (2) and (3) in $T^{i,j}_{G\&H}$ and is guaranteed to be followed by at least one c^j_0. A similar reasoning holds for the symmetric fault $\bar{b}$*-sa-1*.

w-sa-0 A test that detects this fault is $w_1c_1w_0c_0$. The sequence $w^{i,j}_1 w^{i,j}_0$ occurs during the execution of (2) and (3), or (3) and (4) in $T^{i,j}_{G\&H}$, and each $w^{i,j}_1$ ($w^{i,j}_0$) is guaranteed to be followed by at least one c^j_1 (c^j_0).

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 A test that detects this fault is w_0c_1. This test occurs in (4) of $T^{i,j}_{G\&H}$, as the $w^{i,j}_0$ is guaranteed to be followed by a c^j_1. A similar reasoning holds for the symmetric fault $\bar{c}$*-sa-0*.

c-sa-1 A test that detects this fault is w_0c_0. See proof for *b-sa-1*. 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. Tests that detect these faults are c_1c_0 or c_0c_1. Either $c^j_1c^j_0$ or $c^j_0c^j_1$ occurs in (4) or (5) of $T^{i,j}_{G\&H}$ after $w^{i,j}_0$ or $w^{i,j}_1$, 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.

B. Analysis of $T_{C\text{-}SM}$

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 irrelevant value used during initialization. The subscripts of 'read' operations denote the expected output of the operation.

$$
\begin{aligned}
T^{i,j}_{C\text{-}SM} = & \\
& \begin{bmatrix} r^{i,j}_{-} w^{i,j}_{0} \\ c^{j}_{-}\, c^{j}_{-} \end{bmatrix}^{((l-1)-j)} \begin{bmatrix} r^{i,j}_{-} w^{i,j}_{0} \\ c^{j}_{-}\, c^{j}_{-} \end{bmatrix} \begin{bmatrix} r^{i,j}_{0} w^{i,j}_{0} \\ c^{j}_{0}\, c^{j}_{0} \end{bmatrix}^{(j+1)} & (1)\\
& \begin{bmatrix} r^{i,j}_{0} w^{i,j}_{0} \\ c^{j}_{0}\, c^{j}_{0} \end{bmatrix}^{((l-1)-j)} \begin{bmatrix} r^{i,j}_{0} w^{i,j}_{1} \\ c^{j}_{1}\, c^{j}_{1} \end{bmatrix} \begin{bmatrix} r^{i,j}_{1} w^{i,j}_{1} \\ c^{j}_{1}\, c^{j}_{1} \end{bmatrix}^{(j+1)} & (2)\\
& \begin{bmatrix} r^{i,j}_{1} w^{i,j}_{1} \\ c^{j}_{1}\, c^{j}_{1} \end{bmatrix}^{((l-1)-j)} \begin{bmatrix} r^{i,j}_{1} w^{i,j}_{0} \\ c^{j}_{0}\, c^{j}_{0} \end{bmatrix} \begin{bmatrix} r^{i,j}_{0} w^{i,j}_{0} \\ c^{j}_{0}\, c^{j}_{0} \end{bmatrix}^{(j+1)} & (3)\\
& \begin{bmatrix} r^{i,j}_{0} w^{i,j}_{0} \\ c^{j}_{0}\, c^{j}_{0} \end{bmatrix}^{((l-1)-j)} \begin{bmatrix} r^{i,j}_{0} w^{i,j}_{1} \\ c^{j}_{1}\, c^{j}_{1} \end{bmatrix} \begin{bmatrix} r^{i,j}_{1} w^{i,j}_{1} \\ c^{j}_{1}\, c^{j}_{1} \end{bmatrix}^{(j+1)} & (4)\\
& \begin{bmatrix} r^{i,j}_{1} w^{i,j}_{1} \\ c^{j}_{1}\, c^{j}_{1} \end{bmatrix}^{((l-1)-j)} \begin{bmatrix} r^{i,j}_{1} w^{i,j}_{0} \\ c^{j}_{0}\, c^{j}_{0} \end{bmatrix} \begin{bmatrix} r^{i,j}_{0} w^{i,j}_{0} \\ c^{j}_{0}\, c^{j}_{0} \end{bmatrix}^{(j+1)} & (5)\\
& \begin{bmatrix} r^{i,j}_{0} w^{i,j}_{0} \\ c^{j}_{0}\, c^{j}_{0} \end{bmatrix}^{((l-1)-j)} \begin{bmatrix} r^{i,j}_{0} w^{i,j}_{0} \\ c^{j}_{0}\, c^{j}_{0} \end{bmatrix} \begin{bmatrix} r^{i,j}_{0} w^{i,j}_{0} \\ c^{j}_{0}\, c^{j}_{0} \end{bmatrix}^{(j+1)} & (6)
\end{aligned}
$$

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 A test that detects this fault is $w_1 r$. The $w^{i,j}_1 r^{i,j}_1$ sequence occurs in (2) 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. A similar reasoning holds for the symmetric fault $\bar{b}$*-sa-0*.

b-sa-1 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 (2) and (3) 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. A test that detects this fault is $w_1 c_1 w_0 c_0$. The sequence $w^{i,j}_1 w^{i,j}_0$ occurs during the execution of (2) and (3) in $T^{i,j}_{C\text{-}SM}$. Although $c^j_1 c^j_0$ 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 A test that detects this fault is $w_0 c_1$. This test occurs during the execution of (1) and (2) in $T^{i,j}_{C\text{-}SM}$, before the first $w^{i,j}_1$. During (1) the cell is subject to a $w^{i,j}_0$. During (2), a c^j_1 occurs concurrently with the $r^{i,j}_0$, that precedes the first $w^{i,j}_1$. 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 A test that detects this fault is $w_0 c_0$. See proof for *w-sa-0*. 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 at some point during the execution of a march element. A comparison with an identical key-word that results in a single match is performed (See [8]). Tests that detect these faults are $c_1 c_0$ or $c_0 c_1$ while the state of the cell remains unchanged. The sequence $c^j_0 c^j_1$ occurs during the execution of (1) and (2), and during the execution of (3) and (4) in $T^{i,j}_{C\text{-}SM}$ concurrently with $w^{i,j}_0 r^{i,j}_0$ that immediately precedes a $w^{i,j}_1$. Symmetrically, the sequence $c^j_1 c^j_0$ occurs during the execution of (2) and (3) and during the execution of (4) and (5) in $T_{C\text{-}SM}$ concurrently with $w^{i,j}_1 r^{i,j}_1$ that immediately precedes a $w^{i,j}_0$.