Transistor-Level Fault Analysis and Test Algorithm Development for Ternary Dynamic Content Addressable Memories

Derek Wright and Manoj Sachdev

Dept. of Electrical & Computer Engineering
University of Waterloo, Waterloo, Ontario, Canada
dwright@alumni.uwaterloo.ca, msachdev@uwaterloo.ca

Abstract

Content addressable memories (CAMs) are gaining popularity with computer networks. Testing costs of CAMs are extremely high owing to their unique configuration. In this paper, we carried out a transistor-level fault analysis and devise a search path test algorithm. The proposed algorithm is of the order (nl / log_2n) compared to the brute-force algorithm of complexity (nl). For the analyzed CAM, the search path test complexity is reduced by 30x.

1. Introduction

Content addressable memories (CAMs) are semiconductor memories. They differ from regular static and dynamic random access memories (RAMs) by providing additional searching functionality. In a RAM, information is stored and retrieved at a location determined by an address provided to the memory. A CAM builds on this functionality by introducing searching capabilities. Every location in the memory can be compared to a search pattern and the CAM will respond with either a "match" or "mismatch" signal. This could be accomplished in RAM by successively searching every location in memory until a match is found. However, this is extremely inefficient. CAMs perform the search in parallel on every location at once, thus dramatically reducing the search time.

Though they have been in existence since the 1960s, CAMs have recently become increasingly important. Traditionally, CAMs existed as embedded units in microprocessors as the translation lookaside buffer (TLB). However, the growth of the Internet and the demand for increased bandwidth of networks has necessitated search capabilities that traditional RAMs can barely meet. The solution is being found in stand-alone CAMs used in network processors and routers.

The increasing importance of CAMs is evident from articles such as [1]. There is no search throughput performance penalty when using CAMs with a network processor as opposed to RAMs. This allows wire-speed packet forwarding at OC-192 and OC-768 rates. These data rates require up to 100M searches per second which would be enormously difficult to achieve without a CAM. The CAM's ternary nature allows longest-prefix matches (LPM) for Classless Inter-domain Routing (CIDR). The future of CAMs lie in increasing their density, speed, and word length to support the exponentially increasing demands placed on networks.

In spite of obvious CAM benefits, there are challenges in design and manufacturing of CAMs. A significant amount of resources are spent on reducing the power consumption of CAMs and developing efficient test algorithms. CAMs are power hungry devices owing to sheer complexity, parallel search requirements, and performance specifications [2,3,4]. Similarly, the test complexity of CAMs comes from the combination of memory and logic. The traditional memory test algorithms are necessary but not sufficient for adequate testing of CAMs.

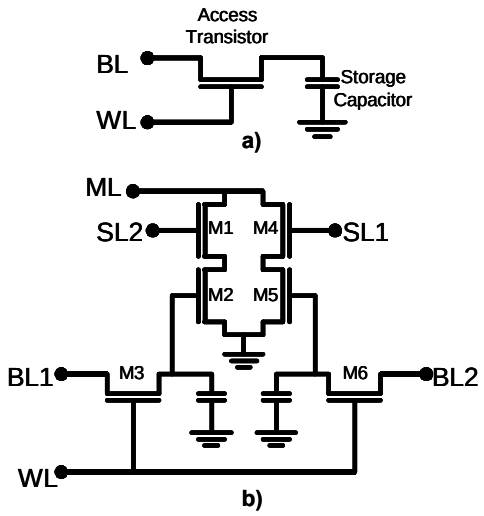

Figure 1 DRAM and DCAM Cells

2. Previous Work on CAM Testing

Until recently, CAMs were mostly considered an exotic type of semiconductor memory. Consequently, most research into memory testing was directed to SRAMs and DRAMs with very little emphasis on CAMs. The only previous works of note are [5] - [8] which focus on testing binary static CAMs, and [9] which focuses on TLBs, which are also static binary CAMs.

In this paper, we examine the testability aspects of ternary CAMs. A realistic defect analysis was carried out on an industrial design. From this analysis an algorithm to identify realistic faults was developed. Although this algorithm was developed for ternary dynamic CAMs, it is equally well suited to ternary static implementations since the search path can be used in conjunction with any static or dynamic storage cell.

This paper is organized as follows: In the next section, an overview of dynamic CAMs is given. It focuses on the unique properties of ternary CAMs. In Section 4, a spice model of the CAM is presented. The spice model is used to perform the transistor-level fault analysis. In Section 5, the algorithmic aspects of CAM testability are described. A MATLAB model is utilized to demonstrate the effectiveness of the algorithm. Finally, in Section 6, conclusions are drawn.

3. Ternary Dynamic CAMs: Background

Figure 1 shows a 1-T dynamic RAM (DRAM) cell and a 6-T dynamic CAM (DCAM) cell. In a DRAM cell, the bit line (BL) is connected to the capacitor when the word line (WL) is high, thus enabling read and write functionality. In a DCAM, the read and write functionality remains the same. The additional four transistors are used for the match operation. The matchline (ML) is precharged to V_{DD}. The storage nodes are loaded with complementary data. The search lines (SL) are charged to the value which is being searched. The four search transistors essentially perform an XNOR operation. In the case of SL data matching the stored data, the matchline does not discharge. If the bits mismatch, then the matchline is discharged through two of the four search transistors. When *n* bits are placed in parallel with a common matchline, if any one bit mismatches, the matchline will discharge. Only if all stored (*n*) bits and complement match *n* pair search lines will the matchline remain charged.

The ternary nature of the CAM cell is evident when "0" is stored on both capacitors. This turns off both the lower transistors in the search transistor paths. Regardless of the values on the search lines, the matchline is unable to discharge. This effectively represents a "don't care" condition being stored, hence the name "ternary": "1", "0", and "don't care". Table 1 shows the different states that can be stored in the ternary DCAM cell. The search lines can both be set to "0" as well. This equates to searching for a "don't care" condition, which will always match since the matchline cannot discharge through either discharge path.

Table 1 **DCAM Storage States**

Value	BL1	BL2
Zero (0)	0	1
One (1)	1	0
Don't Care (X)	0	0
Not Used	1	1

4. DCAM Spice Model and Fault Analysis

A complete spice model of a DCAM was constructed for fault analysis in 0.18 μm CMOS technology. The block diagram of this model is illustrated in Figure 2. This model is based on a commercial design [10]. It contains an array of 16 words by 144 bits ternary dynamic CAM. The basic cell is the same as shown in Figure 1. The bit lines and search lines run vertically throughout the array and the word lines and matchlines run horizontally. The model consists of the 16 words divided into two groups of eight with one dummy word above and below the bit line sense amplifier (BLSA) block. BL and SL drivers are located at the top and bottom of each block of eight words to speed charging and discharging of the highly capacitive lines. WL drivers and address decoders are on the left of the CAM array. Basic control was achieved using an asynchronous state machine with four states: Idle, Read, Write, and Search. Timing was implemented using three separate delay chains with attached static logic for each operation. Although only one is typically used, it was easier to design each delay chain individually and for the purposes of this research it was not critical that only one be used. The model could operate with a minimum operation time of 7.2 ns, or at 139 MHz.

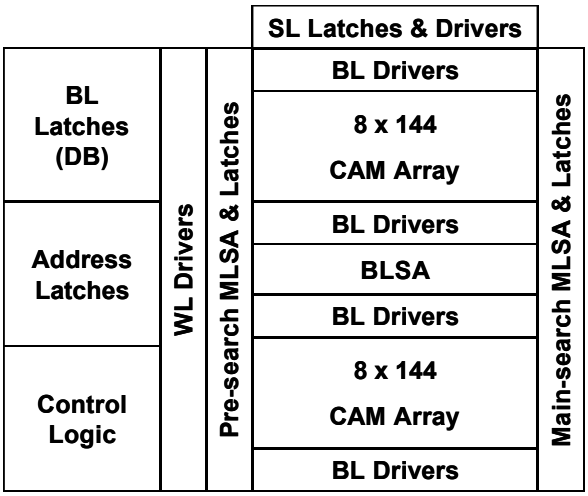

Figure 2 **CAM Spice Model**

The BLSA is a simple voltage-mode sense amplifier consisting of two tristate cross-coupled inverters. The matchline sense amplifier (MLSA) is current-mode and a simplified diagram is shown in Figure 3. During normal operation ϕ_{PRE} is high, tying ML to ground. During a search operation, ML is quickly precharged using ϕ_{FP} to a voltage set by V_{REF}. This voltage is less than V_{DD} in order to reduce the voltage swing on ML and save energy. The I_{REF} is realized using a current mirror set at a current less than a one-bit miss (the minimum current drawn from ML during a mismatch), and greater than the leakage current drawn during a match. If I_{REF} is not supplying enough current then the word is a mismatch and the sense point (SP) will discharge. Otherwise, SP will remain charged indicating a matching word. Further energy savings are achieved by separating the search operation into two phases: Pre-search and Main-search. Pre-search performs the search operation on a small subset of the bits and only if they all match will the main-search proceed. Energy is saved since if the pre-search is a mismatch the main-search operation is not executed on that word. This necessitates the division of the ML into pre- and main-search MLs. The pre-search ML is connected to 36 of the 144 bits and the main-search ML is connected to the remaining 108 bits. If the first 36 bits mismatch then the main-search ML is not precharged. This dramatically reduces power consumption at the cost of a minor increase in complexity.

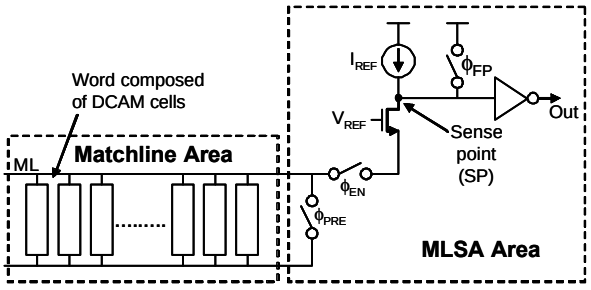

Figure 3 **Current-mode Matchline Sense Amplifier**

Transistor-level faults modeled included gate, source, and drain contact failures, subthreshold conduction due to poor L_{eff} control and wide V_{th} spread, and gate oxide failures that lead to gate-source or gate-drain conduction. Each of these defects in the extreme case is represented as either a short or an open, but intermediate cases occur where the defect is best modeled as a resistance (see Figure 4). Even though the functionality may be preserved, these faults are more insidious since the timing is affected by increased RC time constants.

Transistor-level fault analysis was performed on the 6-T CAM cell. Due to the symmetry of the cell, only one storage node, its access transistor and two search path transistors were analyzed (see Figure 5). Examining possible transistor-level faults yielded five possible circuit-level representations for the faults. These five circuit-level fault representations were applied to each of these three transistors. Adding the possibility of a storage capacitor ground fault revealed 16 unique faults to be examined.

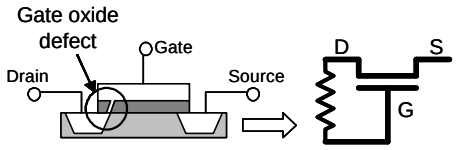

Figure 4 **Example Defect and Circuit Equivalence**

Analysis and simulation of these faults gave a detailed look into the possible failure modes for the DCAM cell and also helped determine how easily detectable these faults were. Some faults resulted in total failure of the cell whereas other faults resulted in failure only under very specific operating cases. This allowed the development of test methods that attempt to catch the subtle faults. These methods are presented in Table 2. The last operation in each method in the column "Detection Method" refers to the result under correct operating conditions. For example, if for Defect #11's method a "mismatch" is detected, then there is a fault. When a "wait" is required, no time is specified since the length of time waited changes the range of faults detected. Sometimes a long wait period is not achievable due to the dynamic nature of the circuit, the control circuitry used, etc.

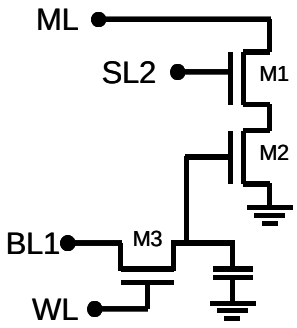

Figure 5 **Fault Analysis Half-Cell**

Using these defect models, simulations were performed to verify the effectiveness of the detection methods (Table 2) for corresponding defects. For some defects, there are multiple ways to detect the defect. However, certain methods were able to detect a wider range of defect resistances resulting in a more robust test method.

Figure 6 illustrates the detection method for Defect #2 (Table 2). In a defect free case, when BL and WL have appropriate values the storage capacitor should charge to V_{DD}. This voltage controls the gate of the M2 transistor.

In the presence of the defect, the stored charge becomes a function of the defect resistance. As the defect resistance increases, lesser charge is stored in a given time reducing the drive of M2. At a certain resistance value, M2 fails to conduct. This behavior is illustrated in the graph in Figure 6. As shown in the figure, at the resistance value of 10kΩ and above, the M2 exhibits the stuck-open (SOP) behavior for write times of 2.5 ns which is limited by the control signals.

Table 2 Possible DCAM Cell Faults

#	Description	Detection Method
1	Storage Capacitor Defect	(a) Write "1"; (b) Wait; (c) Read "1"
2	M3 Source/Drain Contact Defect	(a) Write "0"; (b) SL2 = "1"; (c) Search for Match
3	M3 Gate Contact Defect	(a) WL = "1"; (b) Wait; (c) WL = "0", BL1 = "1", SL2 = "1"; (d) Search for Match
4	M3 Gate to Drain Oxide Failure	(a) Write "0"; (b) Read "0"
5	M3 Gate to Source Oxide Failure	(a) Write "0"; (b) Read "0"
6	M3 Subthreshold Conduction	(a) Write "0"; (b) WL = "0", BL1 = "1",SL2 = "1"; (c) Wait (d) Search for Match
7	M2 Source/Drain Contact Defect	(a) Write "1"; (b) SL2 = "1"; (c) Search for Mismatch
8	M2 Gate Contact Defect	(a) Write "1"; (b) SL2 = "1"; (c) Wait; (d) Write "0"; (e) Search for Match
9	M2 Gate to Drain Oxide Failure	(a) Write "1"; (b) SL2 = "1"; (c) Wait; (d) Read "1"
10	M2 Gate to Source Oxide Failure	(a) Write "1"; (b) Wait; (c) Read "1"
11	M2 Subthreshold Conduction	(a) Write "0"; (b) SL2 = "1"; (c) Search for Match
12	M1 Source/Drain Contact Defect	(a) Write "1"; (b) SL2 = "1"; (c) Search for Mismatch
13	M1 Gate Contact Defect	(a) Write "1"; (b) SL2 = "1"; (c) Wait; (d) SL2 = "0"; (e) Search for Match
14	M1 Gate to Drain Oxide Failure	(a) SL2 = "0"; (b) Search for Match
15	M1 Gate to Source Oxide Failure	(a) Write "1"; (b) SL2 = "1"; (c) Search for Mismatch
16	M1 Subthreshold Conduction	(a) Write "1"; (b) SL2 = "0"; (c) Search for Match

Most of the techniques presented in Table 2 will require precise timing of on-chip control signals to control lines such as word lines, bit lines, search lines, and matchlines to achieve a wide range of fault detection. However, timing is usually fixed owing to nominal operating conditions, so alternate algorithms were created that use the higher functions typically available to the tester: Read, Write, and Search. The high-level test algorithms developed in this paper are designed assuming that weak defects will ultimately result in stuck-on (SON) or SOP faults. This assumption limits the robustness of the fault detection methods because the timing of the internal control signals is fixed. However, even with these timing limits in place, most faults can be grouped into SON or SOP as was shown in Figure 6.

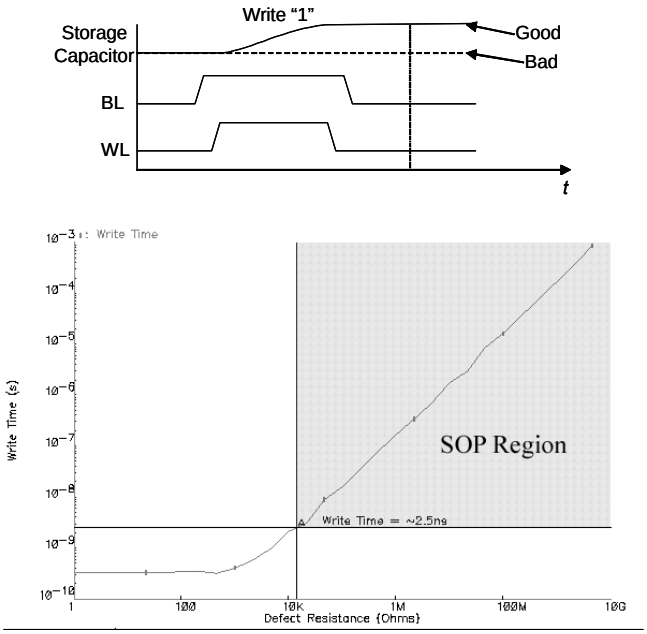

Figure 6 Write Time vs. Defect #2 Resistance

5. Testability Issues in CAMs

CAM cell testing can be roughly divided into two categories: (i) The first part of the complexity is the same as that of any DRAM. It encompasses the access transistor and the storage capacitor. This is a mature research topic with a significant amount of research done on the subject. (ii) The second part of the test complexity comes from the search operation. The output of the search is available on the matchlines. All search paths (144x2) of a given word are connected to the corresponding matchline. Therefore, each search path must be uniquely tested. In a complex CAM, there could be 18 M such paths [10]. Therefore, CAM testing is expensive.

The additional CAM test complexity is to identify and repair a faulty cell. In CAMs it is easier to identify the faulty address (row), but it is more time consuming to find the individual faulty bit (column).

5.1 DRAM Testing

The storage section of the CAM cell is identical to a DRAM cell. Since DRAMs have been ubiquitous for

decades, a great effort has been directed towards discovering and solving their testability issues. A look at the various possible defects that can occur in a DRAM is documented in [6]. An example of a modern test interface for embedded DRAMs is [7].

The testing of DRAMs applies equally to the access transistors and storage capacitors of the CAM. If the storage capacitor is faulty, it may not be able to retain charge. Gate oxide capacitors may have cracks in the oxide which could allow conduction to ground. Other capacitors may have faults with similar effects. The access transistors may suffer from any combination of the faults previously covered. Tests for DRAM cells can be equally applied to testing these portions of the CAM cell.

5.2 Search Path Testing

The search path consists of the matchline and the four discharge transistors of the cell. A successful test algorithm will detect if any one of these transistors is SON or SOP, but an efficient test algorithm will do it quickly.

The algorithm presented in this paper exploits the fact that the search function returns address information. It detects discrepancies between the expected returned addresses and the actual returned addresses. This algorithm will detect all SL and BL transistor SON and SOP faults. It was assumed during the development of this algorithm that the CAM is able to return all matched addresses sequentially, as was the CAM being modeled. In the case of a CAM that only returns the highest priority match, all matching addresses can be returned in sequence by writing mismatching values to the returned address. When a matching address is returned, the complement of the search value is written to the address, thus ensuring a mismatch. The next highest priority address will then be returned. Once the searching is complete the original values can be re-written to the affected addresses.

5.2.1 Algorithm Overview

To help in visualizing the match operation, Figure 7a) shows a symbolic representation of the search path transistors. BL1, BL2, SL1, and SL2 represent the transistors located in the search path. For example, if the cell is storing a “1”, BL1 = “1” and BL2 = “0” as shown in both Figures 7b) and 7c). In Figure 7b), the bit being searched for is also a “1” since SL1 = “1” and SL2 = “0”, so there is a match condition and there is no path for the ML to discharge. However, if a “0” is being searched for as in Figure 7c), SL1 = “0” and SL2 = “1” and there is now a path for the ML to discharge indicating a mismatch. Though the bit lines are not directly tied to the gates of the transistors in the search path, the symbols BL1 and BL2 are used to indicate the values stored on the capacitors since these values initially came from the bit lines. This is also why this search path configuration can be used with static memory architectures. As long as the gates of BL1 and BL2 are driven by the value stored in the memory cells, the type of memory cell becomes irrelevant.

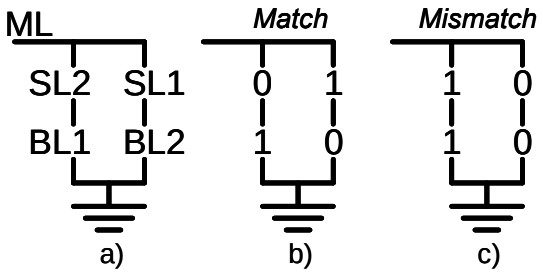

Figure 7 **Search Path Symbolic Representation**

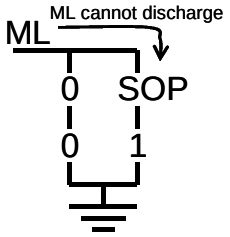

Figure 8 **Search Path with SOP Fault**

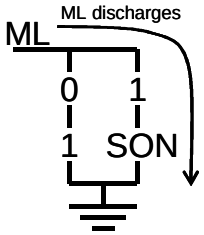

Figure 9 **Search Path with SON Fault**

If every address contains a unique word, then searching for one of these unique words should return only one address. However, if an unexpected address is returned, this indicates an SOP fault. This occurs because the ML could not discharge through the SOP fault and returns a false match. This is illustrated in Figure 8.

If the expected address is not returned, this indicates an SON fault. Even though all the stored bits in the expected address match the bits on the SLs, the SON fault allows the ML to discharge indicating a false mismatch. This is illustrated in Figure 9.

The algorithm will test every CAM cell’s four search path transistors. The algorithm proceeds as follows: 1) Write unique values to every address. 2a) Check for SOP faults by looking for erroneous matching addresses. 2b) Check

for SON faults by looking for missing returned addresses. 3) Repeat the procedure using inverted values when writing and searching.

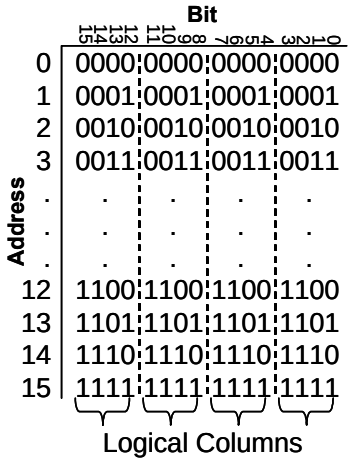

Figure 10 Memory Space with Logical Columns

An easy way to assign a unique word to each address would be to write increasing values to each address starting with "0". Hence, if the address contains 16 bits and the word length is 144 bits, only by assigning a unique value to the first 16 bits, one can have a unique pattern in the entire word space. The rest of the bits (128 padding bits) in every word can be kept as all zeros or ones. However, the above mentioned procedure has some short comings. In order to search for SOP faults in the padding bits, it will take a long procedure to identify the faulty location.

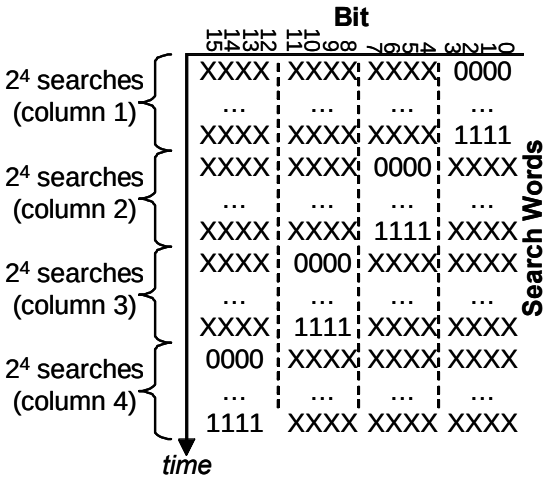

Figure 11 Search Patterns for Increasing Values

An alternative, efficient arrangement can be realized if we divide the word length into logical columns of 16 bits (same as address space) each. In each logical column the same data pattern is repeated, as shown in Figure 10 for a small 16 x 16 CAM. For example, if the address is 16 bits and the word length is 144 bits, there will be 144/16 = 9 columns of increasing (or decreasing) 16 bit numbers.

Once these numbers are stored in memory, each column will be searched individually for the correct increasing (or decreasing) values by masking out the other columns in the search word. This concept is illustrated in Figure 11 for a small, 16 x 16 CAM. The word is divided into logic columns of 4 bits each.

A diagram of the flow of the algorithm is presented in simplified form in Figure 12. It shows the general steps and decisions executed to successfully implement the first pass (ascending values) of the test algorithm. This procedure would be repeated using descending values with ascending address instead – essentially inverting all test data.

5.2.2 Search Path Test Algorithm

This algorithm is composed of three steps, two of which are summarized in Figure 12. Step 3 is to repeat Steps 1 and 2 respectively except that descending values are stored and searched. An explanation of each step follows.

Step 1: This step is responsible for logically dividing the word into $w/\log_2 n$ columns and filling each column of each word with a value equal to that word's address. This is accomplished by taking the modulus of the actual bit position in the word with respect to the number of bits in the address. This yields the bit's relative position within its logical column. For example, if the logical column width is 16 bits, bits 0-15, 16-31, 32-47, etc. are all columns within the word. This means that bits 0, 16, 32, etc. all have the same bit position relative to their column: relative bit position 0. Once the relative bit position is known, the value of that bit can be determined based on the current address location. This is accomplished an "if" statement that chooses either to place a zero or a one in that bit location.

Step 2: Now that the address space is completely filled with unique values, the search function will be used to return useful information. Each column is searched individually in ascending address order. To accomplish this, the logical column is searched and the rest of the search word is set to "don't care" ("X"). The CAM is then searched for the word contained in the search word. Two tests are performed based on the address information returned: A test for SOP faults and a test for SON faults.

Step 2a: The test for SOP faults checks the returned matching addresses to see if addresses other than the one expected are returned. If there are, then they are caused by SOP faults in the words contained in the unexpected returned addresses. If one of the BL or SL transistors has

an SOP fault, there is a transistor currently off that should be on. Thus, the ML of that word cannot discharge through that path. This results in that bit being perceived as a "don't care". The exact location of the bit with the SOP fault can be determined from the returned address. It is at relative bit location $\log_2$(expected address XOR unexpected address). For example, if address 9 was expected, but address 13 was also returned, the defective bit is at relative bit location $\log_2(1001_2 \text{ XOR } 1101_2) = \log_2(0100_2) = \log_2(4) = 2$. This means that the third relative bit inside word with address 13 has an SOP fault.

Step 2b: The test for SON faults is slightly more complicated than the test for SOP faults. If the returned addresses do not contain the expected address at all, then there is an SON fault that is causing the ML to discharge when it should not. It causes a false mismatch. To find the location of the SON fault, the same search word is repeatedly searched, but each time one of the bits is replaced with a "don't care". If the expected address is returned during this process, then the SON fault is due to one of the BL transistors in the bit that was masked when the correct address was returned. However, if the expected address is not returned during this process, this indicates that one of the SL transistors at some location in the entire word is SON and cannot mask properly. This address is noted and a subsequent portion of the algorithm finds the defective bit.

Step 2c: The opposite approach as Step 2b is used to find the remaining SON faults in the addresses that were noted. The search word is completely unmasked and set to the expected contents at the faulty address and the stored word will use masking to find the faulty bit location. A number of search algorithms could be used to find the faulty bit. Successively masking one bit at a time would take on average $l/2$ searches. A binary tree search would require $\log_2(l)$ searches and so is more efficient. The binary search tree algorithm would begin by re-writing the data at the address being searched, except that half of the word would be masked. When searched using an unmasked search word, if the address is successfully returned then the faulty bit is contained in the masked half of the word. Otherwise it is contained in the unmasked part of the word. The algorithm proceeds recursively by further dividing the faulty half in to masked and unmasked quarters, eighths, etc., until the faulty bit is found.

Step 3: Repeat Steps 1 and 2, but write and search using complementary values. This ensures that each transistor is tested for both SOP and SON faults.

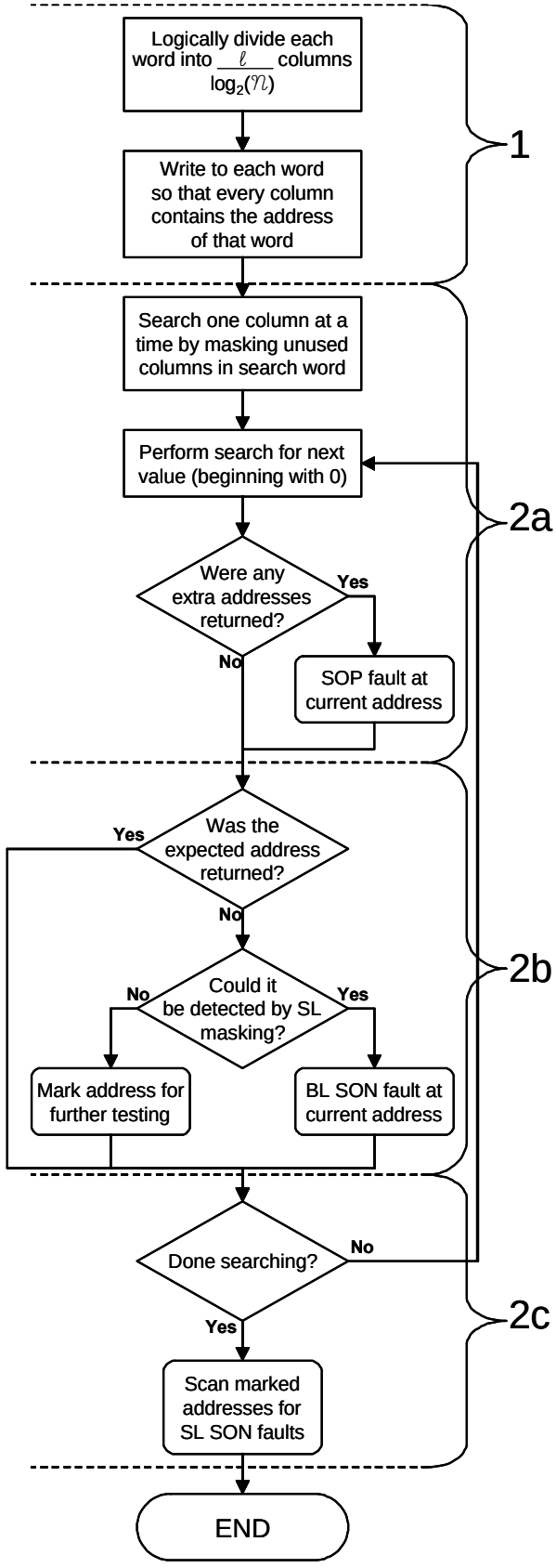

Figure 12 Algorithm Overview

An assumption was made in the algorithm that the number of bits in a word divided by the number of address bits is an integer. This may not be the case, but the algorithm needs only a simple modification to function properly. The remaining bits will be correctly written to in Step 1. Since bits are only searched as a part of a logically divided column, an extra column with the same width as the others must overlap the extra bits and also be searched. The only consequence of this is that the bits that are contained in both a regular column and the extra column will be tested twice.

5.3 Algorithm Validation

A MATLAB model was created consisting scripts and functions that simulate the higher-level functionality of the CAM. All BLs and SLs are represented as arrays of ones and zeros. Addresses are realized as array indices. This higher level model allows fast debugging and evaluation of possible search path test algorithms.

For example, the search function in MATLAB sets a variable "ML" = 1 to represent precharging of a ML. It then compares the SLs to the BLs bit by bit at an address and sets "ML" = 0 if a mismatch occurs. All addresses are searched sequentially and each matched address is returned. Thus, though the MATLAB search function is not a true representation of the CAM hardware, the CAM's functionality is mimicked in a thorough manner to allow quick algorithmic validation. Algorithms were validated in MATLAB using a 20-bit word and a 5-bit address. Once debugged in the MATLAB, the algorithm was verified using 144-bit words and 16-bit addresses.

The functions necessary for the algorithm presented in this paper are *write, search*, and *find*. The function *write* (*word*, *addr*) stores *word* at address *addr*. This represents the write function in a CAM. In the algorithm, it is assumed that *search* (*word*) returns an array of addresses that contain *word*. This represents the search function built into a CAM. The basic idea behind simulating the *search* functionality is that the matchline of a given word will discharge if (SL2 AND BL1) OR (SL1 AND BL2) is true. This cannot be abbreviated as an XOR statement since SL1 and SL2 are not necessarily complementary as was shown in Table 1. Similarly, BL1 and BL2 are also not necessarily complementary.

Finally, the *find* (condition) function parallels that of the MATLAB *find* function which will return the indices of an array for which the condition is true. For example, *find* ([3 1 2 4 5 2] == 2) would return [3 6], and *find* ([4] == 2) would return [ ]. The use of this function dictates that the tester used to test the dies must make decisions during the execution of the test algorithm. The *write* and *search* functions are implemented in hardware on the CAM.

5.4 Algorithm Benefits and Weaknesses

The algorithm identifies both the row and the column location of a faulty bit by exploiting returned address information by the search function. Furthermore, the algorithm could also determine the exact faulty transistor. However, it is typically not needed in the redundancy and repair scenario. Arguably, this function may be needed in defect data collection and yield improvement. This information is available and one may collect and use it.

A brute force algorithm similar to the one currently used to test MOSAID CAMs was used as a basis for comparison. It proceeds in two steps: 1) Test the ability for an address to match. 2) Test each bit's ability to mismatch. Step 1 is accomplished as follows: Write all zeros to the current address being tested, and then search for all zeros to see if the current address is returned as a match. Clear the address by writing all ones and proceed to the next address. Repeat using complementary values. Step 2 is accomplished as follows: Write 000…001 to the current address being tested, and then search for all zeros to ensure that the current address is a mismatch. Left shift the "1" bit (000…010) and write the pattern to the current address and repeat the search. Once all bits in the current address have been searched, write all ones to that address and proceed to the next address. This algorithm individually tests each bit's ability to match and mismatch for both ones and zeros will have a total number of writes and searches as follows:

$$2n(l+5) \text{ writes} + 2n(l+1) \text{ searches} \qquad (1)$$

where n = number of addresses and l = number of bits in each word. The factor "2" arises from having to test both the ones and zeros cases. This doubles the test complexity. There are n writes required each time the memory is cleared which occurs four times. Step 1 requires n writes to write test patterns, and Step 2 requires nl writes to write test patters. Repeating this once results in a total of $2n(l+5)$ writes. Step 1 requires n searches to test each address, and Step 2 requires nl searches to test each bit. Repeating this once results in a total of $2n(l+1)$ searches. For 64k addresses and 144 bit words, this equals 19,529,728 writes and 19,005,440 searches.

The algorithm presented in this paper has an average total number of writes and searches as follows:

$$2n + \log_2(l) \times SON \text{ writes and}$$

$$2n\left(\frac{l}{\log_2(n)}\right) + \log_2(l) \times SON \text{ searches,} \qquad (2)$$

where SON = number of stuck-on faults in the SL transistors. The fixed overhead of the algorithm with no SL SON faults is $2n$ writes and $2n\left(\frac{l}{\log_2(n)}\right)$ searches. The factor "2" arises from having to test both the ones and zeros cases. This doubles the test complexity. Each pass of the algorithm requires n writes to write test patterns, and $n\left(\frac{l}{\log_2(n)}\right)$ searches where the number of logical columns is $\frac{l}{\log_2(n)}$. The terms of the equations multiplied by SON represent the average additional writes and searches due to the SL SON fault. It is assumed that the binary search tree algorithm is used. With no SON faults and the same 64k x 144 bit structure, this equals 131,072 writes and 1,179,648 searches. Each SON fault requires an additional 8 writes and searches to be detected. Assuming that writes and searches take the same amount of time, this results in the test complexity reduction by a factor of 30.

The major weakness of this algorithm is that the detection of SL SON faults requires some decision making by the tester. The other algorithm steps can be executed as a script and the results can be analyzed offline. One possible solution to this would be to take the addresses marked for SL SON fault testing and to test every bit in sequence. The resulting data could be analyzed offline and no decision making would be required of the tester.

6. Conclusions

In this paper, we analyzed the test complexity of a 9Mb ternary dynamic CAM. A search path algorithm was developed using realistic transistor-level defects. This algorithm was validated using a high-level MATLAB model. The new algorithm compared to the brute force method is 30 times faster owing to reduced complexity.

7. Acknowledgements

This research is based on MOSAID's 9Mb ternary dynamic CAM and consequently would not have been possible without resources provided by MOSAID. These include technical documents, and more importantly, numerous correspondences and conversations with several key MOSAID employees. Most notably, thanks go to Tomasz Wojcicki for giving his time and resources to this project, Jin-Ki Kim for his in-depth technical knowledge and helpfulness, and to Brent Taylor for detailed information regarding CAM testing.

8. References

[1] K. Schultz, "A CAM Memory for 10Gbit and Terabit Routers and Switches", *Electronic Engineering*, vol. 72, no. 880, pp. 57-62, May 2000.

[2] E. Shen, and J. B. Kuo, "0.8V CMOS Content-Addressable-Memory (CAM) Cell Circuit With A Fast Tag-Compare Capability Using Bulk PMOS Dynamic-Threshold (BP-DTMOS) Technique Based On Standard CMOS Technology for Low-Voltage VLSI Systems", *IEEE International Symposium on Circuits and Systems*, pp. 583-586, 2002.

[3] C. A. Zukowski, and S.-Y. Wang, "Use of Selective Precharge for Low-Power Content-Addressable Memories", *IEEE International Symposium on Circuits and Systems*, pp. 1788-1791, 1997.

[4] G. Thirugnanam, N. Vijaykrishnan, and M. J. Irwin, "A Novel Low Power CAM Design", *14th Annual IEEE International ASIC/SOC Conference*, pp. 198-202, 2001.

[5] P. Sidorowicz, and J. Brzozowski, "Verification of CAM Tests for Input Stuck-at Faults", *International Workshop on Memory Technology, Design, and Test*, pp. 76-82, 1998.

[6] P. Sidorowicz, and J. Brzozowski, "An Approach to Modelling and Testing Memories and its Application to CAMs", *VLSI Test Symposium*, pp. 411-416, 1998.

[7] P.Sidorowicz, "Modelling and Testing Transistor Faults in Content-Addressable Memories", *International Workshop on Memory Technology, Design, and Test*, pp. 83-90, 1999.

[8] K. Lin, and C. Wu, "Testing Content-Addressable Memories Using Functional Fault Models and March-Like Algorithms", *IEEE Transactions on Computer Aided Design of Integrated Circuits and Systems*, pp. 577-588, 2000.

[9] S. Kornachuk, L. McNaughton, and R. Gibbins, B. Nadeau-Dostie, "A High Speed Embedded Cache Design with Non-Intrusive BIST", *International Workshop on Memory Technology, Design, and Test*, pp. 40-45, 1994.

[10] MOSAID Technologies, "MOSAID Class-IC DC9288 Feature Sheet", February 2003, http://www.mosaid.com/semiconductor/dc9288fs.pdf

[11] H.-D. Oberle, M. Maue, and P. Muhmenthaler, "Enhanced Fault Modelling for DRAM Test and Analysis", *VLSI Test Symposium*, pp. 149-154, 1991.

[12] S. Miyano, K. Sato, and K. Numata, "Universal Test Interface for Embedded-DRAM Testing", *IEEE Design and Test of Computers*, pp. 53-58, January-March 1999.