A Comprehensive Test and Diagnostic Strategy for TCAMs

by

Derek Warren Wright

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Master of Applied Science
in
Electrical and Computer Engineering

Waterloo, Ontario, Canada, 2004

I hereby declare that I am the sole author of this thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

Derek Warren Wright

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 thesis, a fault analysis is carried out on an industrial ternary CAM (TCAM) design, and search path test algorithms are designed. The proposed algorithms are able to test the TCAM array, multiple-match resolver (MMR), and match address encoder (MAE). The tests represent a 6x decrease in test complexity compared to existing algorithms, while dramatically improving fault coverage.

Acknowledgements

First and foremost, I would like to thank my supervisor **Dr. Manoj Sachdev** for his guidance and support throughout my graduate studies. His kindness and generosity, along with his experience and expertise has pointed my life in a direction of research and continuous learning, for which I am grateful.

I would also like to thank my CAM Group team members, **Nitin Mohan** and **Wilson Fung**, who are both very helpful, intelligent, and enjoyable to work with.

Special thanks to **Dr. Arokia Nathan** and **Dr. Andrew Kennings** for reviewing this thesis. Their time, energy, and willingness to sit down and help me regardless of the task are greatly appreciated.

Finally, I thank my family. Most people are not lucky enough to know someone as supportive, encouraging, and intelligent as my fiancée, **Amy LeRoij**, who truly is my lab partner for life. My **parents** and **brother**, who are continually supportive, and **Amy's parents**, who treat me like one of their own, all deserve my true and sincere gratitude because they always have my best interests in mind.

Table of Contents

Abstract ... **iii**

Acknowledgements ... **iv**

List of Figures ... **vii**

List of Tables ... **viii**

Chapter 1 Introduction ... **1**

1.1 Motivation ... 2

1.2 Thesis Organization ... 2

Chapter 2 TCAMs: Background ... **4**

2.1 TCAM Overview ... 4

2.2 TCAM Architecture ... 6

2.2.1 TCAM Array ... 8

2.2.2 Peripheral Components ... 10

2.2.3 DfT Components ... 14

Chapter 3 TCAM Spice Model and Cell Fault Analysis ... **15**

3.1 TCAM Spice Model ... 15

3.2 TCAM Cell Fault Analysis ... 16

Chapter 4 Testability Issues in TCAMs ... **22**

4.1 Previous Work on TCAM Testing ... 22

4.2 Overview of TCAM Testing ... 22

4.2.1 Proposed Test Algorithms ... 23

4.2.2 Design for Test ... 24

4.2.3 BIST Architecture ... 26

4.3 DRAM Testing ... 26

4.4 Search Path Testing ... 27

4.4.1 MAE Testing ... 28

4.4.2 MMR Testing ... 30

4.4.3 TCAM Array Testing ... 35

4.4.4 Test Summary ... 47

4.5 Algorithm Validation ... 48

4.6 Algorithm Benefits and Weaknesses ... 50

4.6.1 Existing Test Algorithm Complexity 51

4.6.2 Comprehensive TCAM Test Strategy Complexity 52

4.6.3 Comparison of Test Algorithm Complexities 56

4.6.4 Impact of BIST on Test Complexities 57

Chapter 5 Conclusions 59

References 60

List of Figures

Figure 2.1 – a) DRAM Cell and b) TCAM Cell....................4

Figure 2.2 – Simplified TCAM Search Architecture....................6

Figure 2.3 – TCAM Architectural Overview....................7

Figure 2.4 – TCAM Array Organization....................9

Figure 2.5 – TCAM Word During Search Operation....................10

Figure 2.6 – MMR Tree Configuration....................12

Figure 2.7 – 3-bit MAE....................13

Figure 2.8 – Delay Chain Signal Generation....................14

Figure 3.1 – TCAM Spice Model....................15

Figure 3.2 – Simplified MLSA....................17

Figure 3.3 – Example Defect and Circuit Equivalence....................17

Figure 3.4 – Transistor Level Defects Modelled....................18

Figure 3.5 – Write Time vs. Defect #2 Resistance....................21

Figure 4.1 – DfT Structures....................25

Figure 4.2 – Detailed View of MMR/MAE Scan Chain....................29

Figure 4.3 – MAE Input Test Vectors....................30

Figure 4.4 – L1 MMR Test Bus Configuration....................32

Figure 4.5 – L2 MMR Test Bus Configuration....................33

Figure 4.6 – MMR Input Test Vectors....................35

Figure 4.7 – Comparison Logic Symbolic Representation....................36

Figure 4.8 – Comparison Logic with a) SOP and b) SON Faults....................37

Figure 4.9 – Memory Space with Logical Columns....................39

Figure 4.10 – Search Patterns for Increasing Values....................39

Figure 4.11 – Intra-cell Test Algorithm Overview....................40

Figure 4.12 – Influence of Surrounding TCAM Cells....................44

Figure 4.13 – Inter-cell Test Algorithm Overview....................45

Figure 4.14 – Existing Test Procedure and Complexity....................52

Figure 4.15 – Proposed TCAM Test Procedure and Complexity....................55

Figure 4.16 – Comparison of Proposed and Existing Test Algorithms....................57

List of Tables

Table 2.1 – TCAM Storage and Search States 5

Table 3.1 – Possible TCAM Intra- and Inter-cell Defects and Detection Methods 19

Table 4.1 – MMR/MAE Mux Positions and Corresponding Functionality 29

Table 4.2 – MMR Mux Positions and Corresponding Functionality 31

Table 4.3 – TCAM Test Summary 48

Chapter 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. In essence, this function can be thought of as the inverse of RAM: In a RAM the user provides the address, and an operation (read or write) is performed on the data. In a CAM the user provides the data and an operation is performed that returns the address. This functionality could be achieved 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. A ternary CAM (TCAM) adds masking capabilities by storing and searching a third “don’t care” state.

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 TCAMs, which are used in network processors and routers. TCAMs have recently gained popularity as a means of speeding up the destination address search function and quality of service (QoS) lookup required by network routers. The masking capabilities are extremely useful in classless inter-domain routing (CIDR) Internet applications where longest-prefix matches (LPM) are used.

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

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

1.1 Motivation

The motivation for this thesis lies in the fact that TCAM testing has been relatively ignored in comparison to RAM testing. Thus, this area of research is ripe for new developments, and given the increasing prevalence of TCAMs, advancements in TCAM testing will be welcomed in industry. Existing TCAM test methods are often an afterthought of designers, but with test costs dramatically rising [5] this is no longer an acceptable practice. This thesis represents an attempt by the author to characterize the major test necessities of TCAMs from both a functional and fault perspective, and to present methods to detect and diagnose faults in an effective and timely manner.

1.2 Thesis Organization

This thesis is organized as follows: In Chapter 2, an overview of TCAMs is given. It focuses on the unique properties of TCAMs versus RAMs and binary CAMs. In Chapter 3, a spice model of the TCAM is presented. The spice model is used to perform a transistor level fault analysis. In Chapter 4, the connections between the results of the defect analysis of Chapter 3 and the algorithmic aspects of TCAM testability are described. A MATLAB model is utilized to demonstrate the effectiveness of the resulting algorithm. Finally, in Chapter 5, conclusions are drawn and future research directions are suggested.

The overall goal of this thesis is to develop a comprehensive test and diagnostic strategy for TCAMs. The test strategy is developed by starting with a clear understanding of TCAM architectures, and by performing a defect analysis of a TCAM cell. The possible failure modes of the cell, along with the unique configuration of peripheral components, enable the development of a test strategy consisting of on-chip test circuits and accompanying algorithms. This test strategy is designed to diagnose specific faulty cells and components. The results of this diagnosis could be used along with on-chip redundancy to repair some faulty TCAM chips despite their defects. Redundancy is not covered in this thesis.

Chapter 2

TCAMs: Background

2.1 TCAM Overview

Figure 2.1a) shows a 1-T dynamic RAM (DRAM) cell and Figure 2.1b) shows a 6-T dynamic TCAM 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 dynamic TCAM, there are two DRAM storage cells, (M3 and M6), and the read and write functionality remains the same. The additional four transistors (M1, M2, M4, and M5) make up the comparison logic used for the match operation.

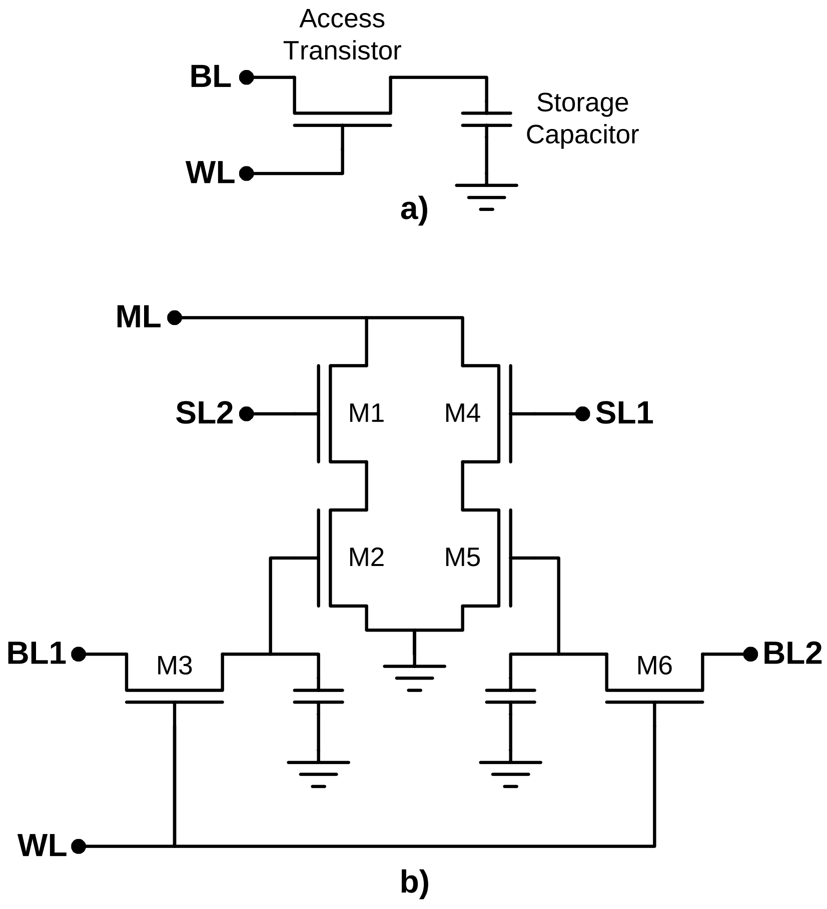


Figure 2.1 – a) DRAM Cell and b) TCAM Cell

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 comparison logic essentially performs an XNOR operation. In the case of SL data matching the stored data, the ML does not discharge. If the bits mismatch, then the ML is discharged through two of the four search transistors. When l bits are placed in parallel with a common ML, if any one bit mismatches, the ML will discharge. Only if all stored (l) bits and complement match l-pair search lines will the ML remain charged.

The ternary nature of the TCAM cell is evident when "0" is stored on both capacitors. This turns off both the lower transistors in the comparison logic. Regardless of the values on the SLs, the ML 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 2.1 shows the different states that can be stored in the TCAM cell. The SLs can both be set to "0" as well. This is equivalent to searching for a "don't care" condition, which will always match since the ML cannot discharge through either discharge path in the comparison logic.

Table 2.1 – TCAM Storage and Search States

Value	**BL1/SL1**	**BL2/SL2**
Zero (0)	0	1
One (1)	1	0
Don't Care (X)	0	0
Not Used	1	1

Figure 2.2 shows a simplified schematic of a TCAM search architecture. In the event of multiple matches, a priority encoder (PE) allows the ML with the highest priority to be encoded as a matching address. The PE is composed of a multiple-match resolver (MMR) and an address encoder (MAE). The MMR allows only the ML with the highest priority to pass to the MAE, and outputs a multiple-match detection signal if there is more than one match present. The MAE is a read-only memory (ROM) that encodes the address of the output from the MMR. These additional components are not

found in RAMs, hence special attention must be paid to testing both these components and the comparison logic.

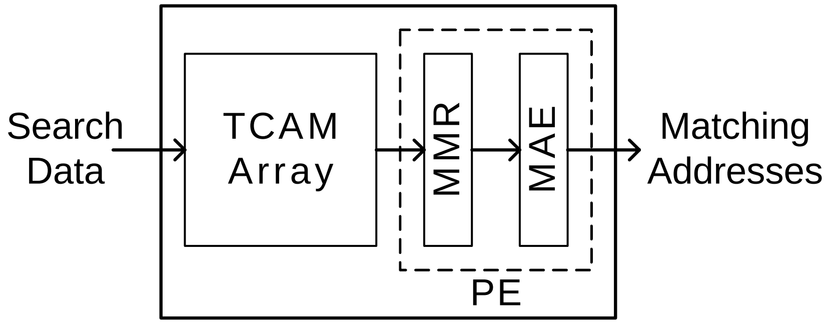


Figure 2.2 – Simplified TCAM Search Architecture

2.2 TCAM Architecture

A typical TCAM integrated circuit (IC) consists of three major types of circuits:

1. TCAM arrays to store ternary data
2. Peripheral components to facilitate read, write, and search functionality
3. Design for Test (DfT) components that enable testing of the various other components

Figure 2.3 shows a simplified block diagram of a TCAM illustrating the first two types of circuits. DfT components will be treated in detail in Chapter 4. The TCAM array consists of TCAM cells that make up TCAM words. Peripheral to the array is its associated address decoders, line drivers, buffers, sense amplifiers, etc. Also, the peripheral circuitry includes the MMR and MAE. The DfT components include the row/column redundancy, on-chip testing circuits, scan chains, and any other circuits specifically included for testing purposes.

In most applications, a TCAM search can result in multiple matches. An MMR is used to resolve the multiple match output into a single match that has the highest priority. The output is then encoded as a matching address by the MAE. Usually the priority is hard-coded based on the physical address of the TCAM word. For example, a matching word located at a lower address has higher priority than a matched word located at a higher address. Some schemes employ programmable priority, which removes the restriction of physical address-based priority and allows the user to assign each word its own priority. The MMR also generates a multiple-match detection (MMD) signal that indicates the presence or absence of multiple matches. Thus, the TCAM can be designed to output all the matching addresses of a search in order of priority by using the MMD signal to indicate to the user that more matching addresses are available.

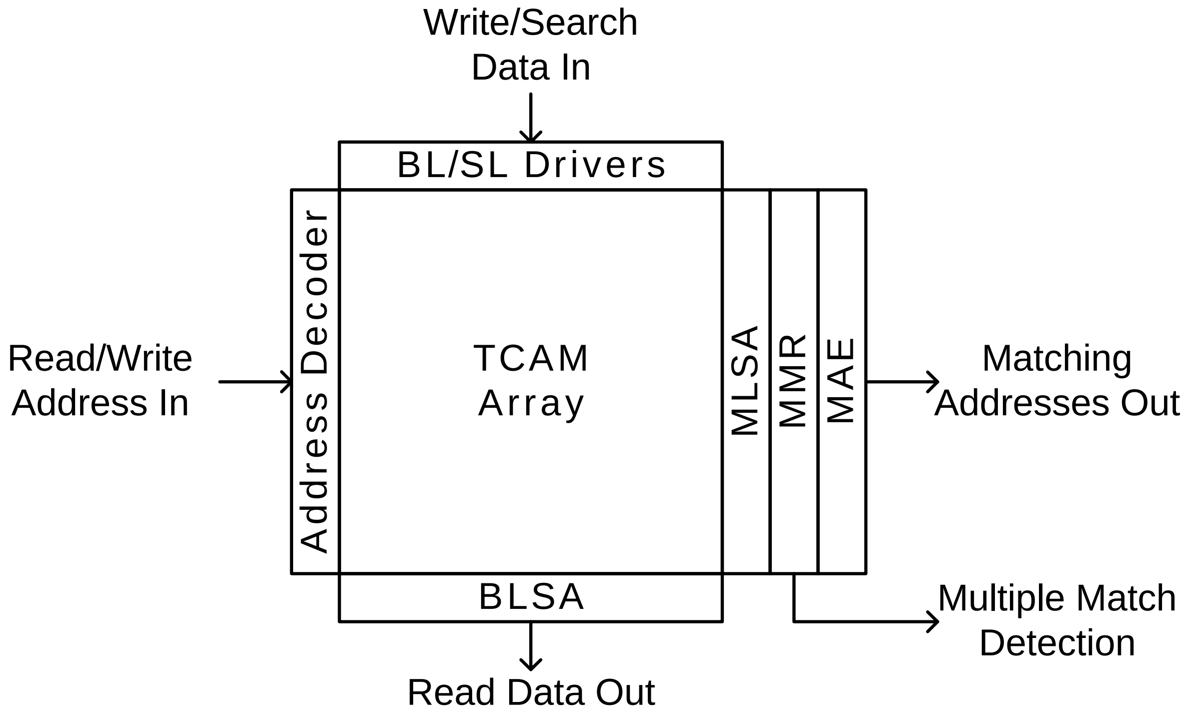


Figure 2.3 – TCAM Architectural Overview

In memory semiconductors, it is common practise to segment the entire memory into blocks and banks. This allows power to be saved by disabling portions of the memory that are not in use. Also, signal drivers can be made smaller since they only have to drive local busses, and not global busses. Typically in a TCAM with 144-bit words, one block will contain 128 or 256 words, which maintains a reasonable layout aspect ratio. The IC may be partitioned into two, four, or eight banks to allow clock gating and bank disabling to save power [6], but this level of abstraction will not be addressed in this thesis. In existing stand-alone TCAMs, the chip level may contain several hundred-thousand words.

A densely-packed large size memory chip requires some degree of DfT to allow observability and controllability without consuming a large number of I/O pins for testing [5]. The goal of the testing is to identify faulty components and replace them with redundant on-chip components, and also to determine whether or not the chip under test can be repaired or if it should be rejected. DfT has the potential to make testing faster and to dramatically increase fault coverage when compared to completely off-chip algorithmic solutions.

2.2.1 TCAM Array

Figure 2.4 shows a more detailed block diagram of the TCAM array. Each memory block in a TCAM chip is implemented as an array of TCAM cells, where each TCAM cell is capable of storing a ternary state. Typically, a horizontal row forms a word of a TCAM-based table. Within a word, a bit is located by its column number. All the TCAM cells in a row share a common WL and a common ML. Similarly, all the TCAM cells in the same column share a common set of BLs and a common set of SLs. There are *l* bits and *n* addresses as shown in the figure.

2.2.1.1 TCAM Word

A TCAM word consists of *l* TCAM cells all connected to a common WL and ML. During a write operation, an *l*-bit word is stored in the TCAM cells at the address specified by the user. The read operation returns an *l*-bit word stored at the given address. The search operation compares an *l*-bit search word against the *l*-bits stored in the TCAM word, and if the two match, the address of the TCAM word is returned.

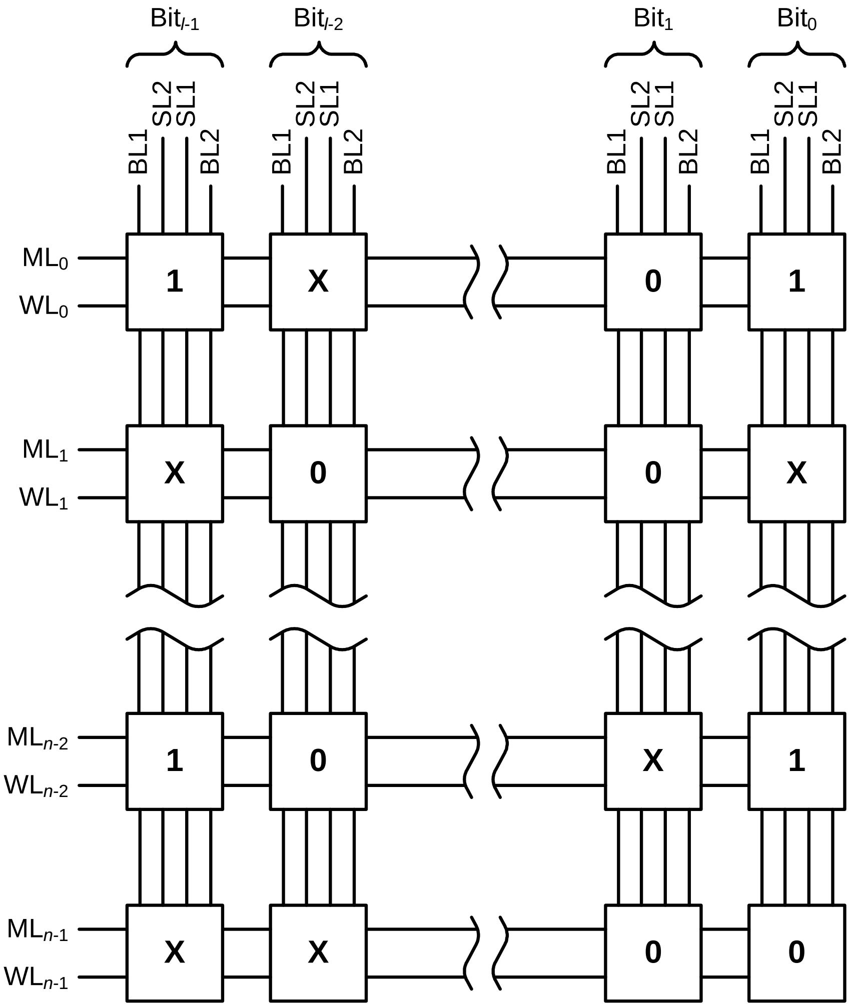


Figure 2.4 – TCAM Array Organization

During a search operation, the ML is precharged and the *l*-bit search word is applied to the TCAM cells on the SLs. Any mismatching cells cause the ML to discharge, indicating a mismatch. This is illustrated in Figure 2.5. The “X” (“don’t care”) condition shuts off the search path for that cell, thus preventing an ML discharge regardless of the applied data. If the ML remains charged, the stored word and search word match.

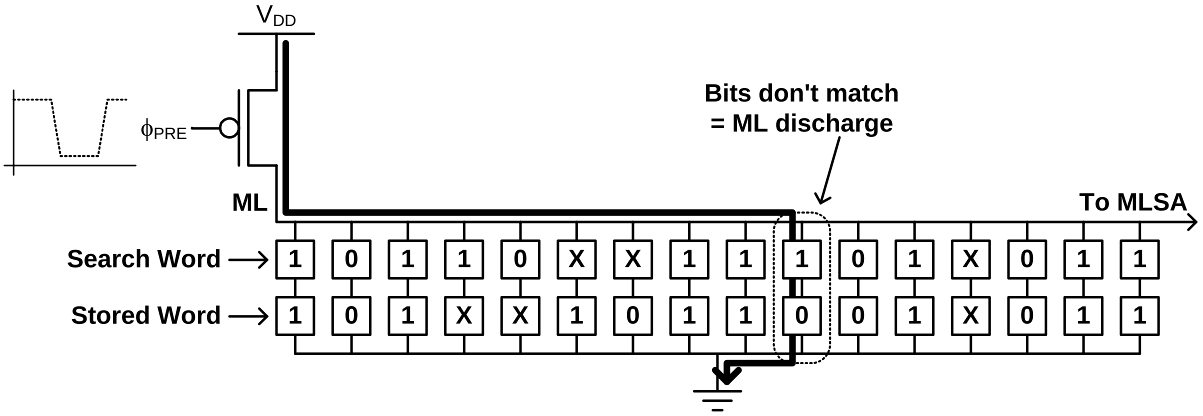


Figure 2.5 – TCAM Word During Search Operation

2.2.2 Peripheral Components

2.2.2.1 WL and BL Drivers and BLSAs

The WL drivers are the same as in any RAM. The WLs are driven by large buffers activated by an address decoder. There are existing test algorithms that are quite suitable for testing address decoders with a high fault coverage [7]. The BL drivers are tristate buffers that can be activated during write operations, and set to a high-Z state during read operations. The BL sense amplifiers (BLSAs) can be voltage or current mode, and are single-ended in dynamic memories or double ended (differential) in static memories. All these components are highly established and researched, and are identically implemented in TCAMs as they are in RAMs. Thus, they will not be treated any further in this thesis.

2.2.2.2 SL Drivers and MLSA

The SL drivers are buffers very similar to the BL drivers, except that they do not need to be tristate since the SLs are never read from, only written to. The ML sense amplifiers (MLSAs) are responsible for detecting whether or not its associated ML has charged or discharged during a search operation. Since they play a large role in determining the power consumption of TCAM ICs, MLSAs are an area of increasingly intense research. The MLs are charged during every search operation. Limiting the minimum charge necessary to make an accurate detection of the ML's state is critical to reducing the energy required per search operation. Consequently, there are numerous MLSA designs, all aimed at lowering the charge used on the ML during the search operation. For the purposes of TCAM array testing, the MLSA is assumed to provide correct results since a malfunctioning MLSA would result in grossly incorrect test results (always matching, or always mismatching), which is easily detected.

2.2.2.3 MMR

Due to the technical impracticalities of directly creating an *n*-bit MMR using traditional designs (the Boolean expressions are too cumbersome), a tree structure of smaller MMRs is used to create a distributed *n*-bit MMR. Each node of the tree is typically a small 16- or 32-bit MMR. This tree configuration is depicted in Figure 2.6 for a distributed 256-bit MMR. A second MMR level (L2 MMR) interacts with the first level (L1 MMR) to disable lower priority L1 MMRs in the case of multiple matches. The figure shows a typical configuration at the block level for a typical block size of 256 bits. The MMR is implemented at the chip level by introducing higher levels of MMRs which enable and disable the L2 MMRs (L3, L4, etc.).

For example, assume in Figure 2.6 that address 0 has the highest priority and address 255 is the lowest. If addresses 3, 15, and 240 are all matches, both L1 MMR_0 and L1 MMR_{15} indicate matches to the L2 MMR. The L2 MMR detects the conflict and disables L1 MMR_{15}. The L1 MMR_0 then chooses address 3 as higher priority than address 15, and thus the only match passed to the MAE is address 3, the highest priority address. If each L1 MMR has an “enable” input and a “match found” output, then the L2 MMR can be functionally the same as the L1 MMRs, using the L1 “match found” signals as inputs and outputting the “enable” signal to the highest-priority L1 block.

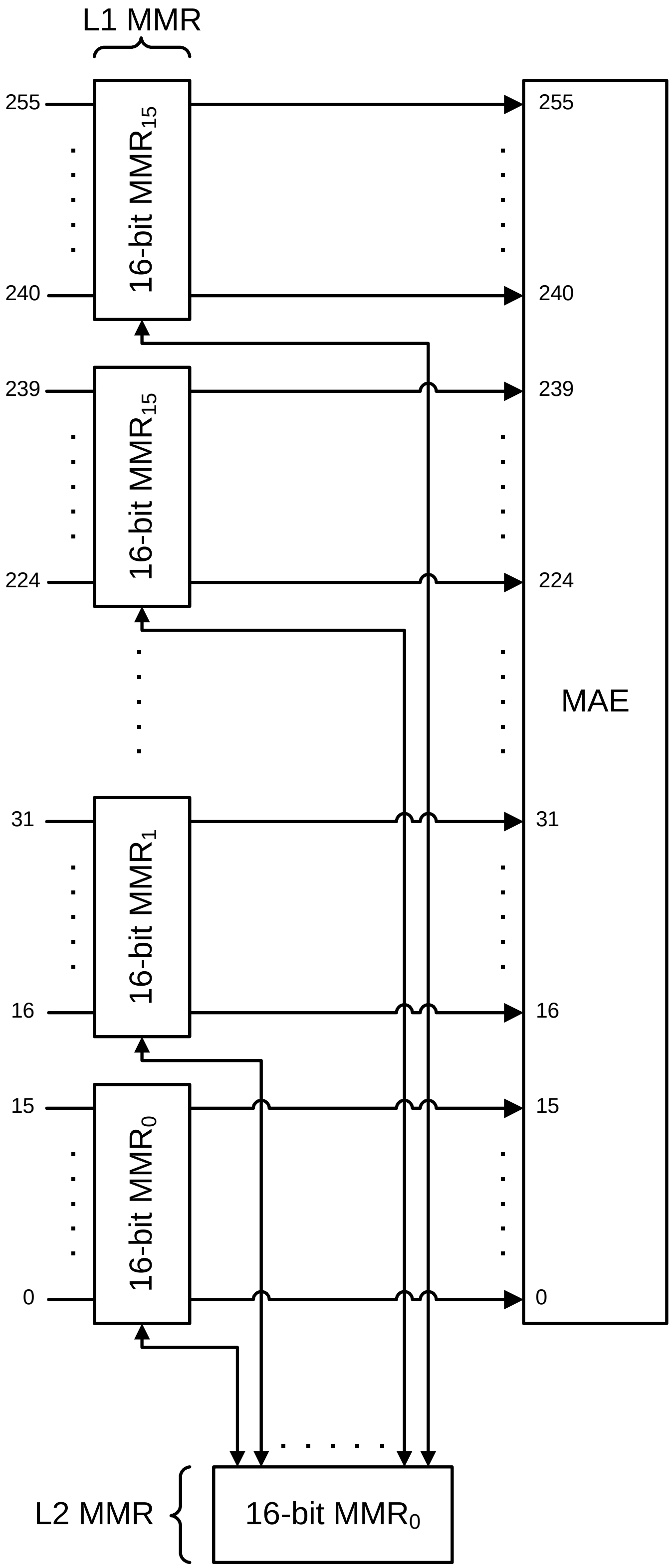


Figure 2.6 – MMR Tree Configuration

2.2.2.4 MAE

The MAE is essentially an n-input ROM with $\log_2(n)$ outputs. Each input line results in a hard-coded output that corresponds to that input line's address. A simple 3-bit example is depicted in Figure 2.7. It is a dynamic circuit, and thus requires a precharge clock signal (ϕ_{PRE}). All output lines are precharged to V_{DD} (logic "1") through the PMOS transistors, and when the input signal arrives on one of the n lines, the gates of NMOS transistors discharge the appropriate outputs to zero. For example, in Figure 2.7 if the input line 6 was "1" and the rest were "0", the outputs would first be charged to $\{Out_2\ Out_1\ Out_0\}$ = "111". Then the corresponding NMOS transistor would discharge the Out_0 line to "0", resulting in an encoded output of "110" = 6.

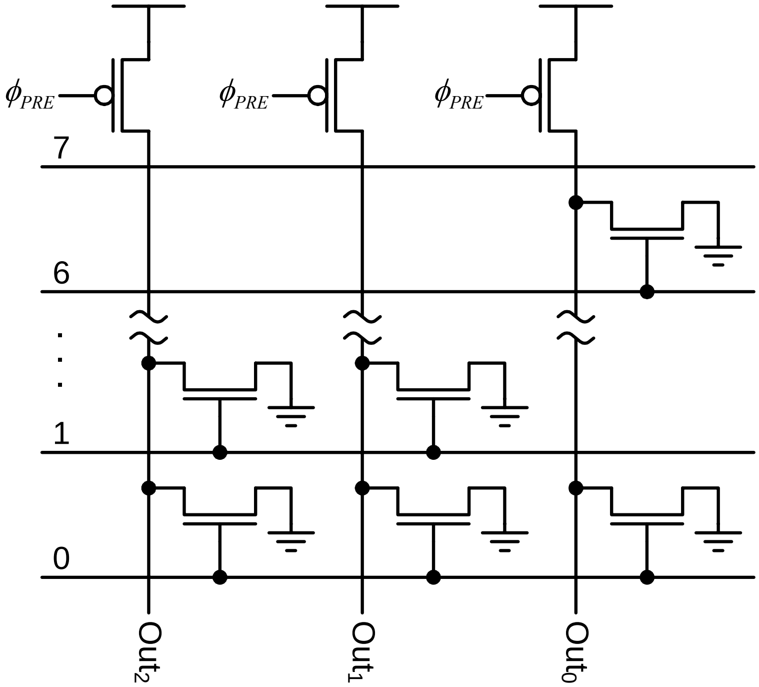


Figure 2.7 – 3-bit MAE

2.2.2.5 Timing Controllers

Timing controllers are typically implemented as delay chains, which are made up of a chain of inverters whose delays are tuned to meet internal signal timing specifications. Figure 2.8 depicts an example of one possible delay chain configuration, and how gates can be used to generate a pulse of a specific width.

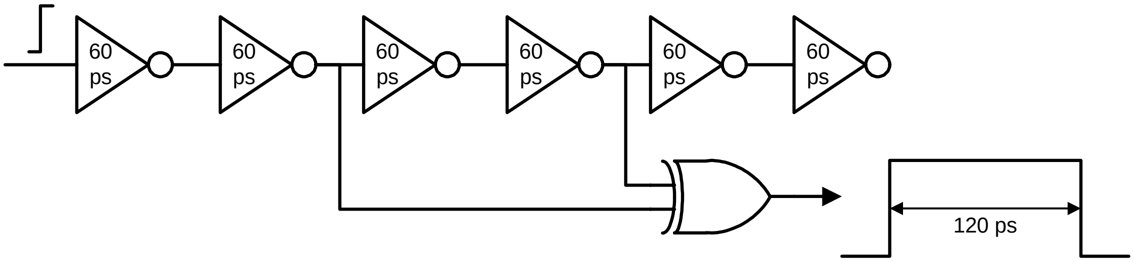


Figure 2.8 – Delay Chain Signal Generation

2.2.3 DfT Components

DfT components encompass all subsystems used only during testing, and not during normal operation. These structures include scan chains, multiplexers, test busses, test controllers and interfaces, test pins, built-in self test (BIST) structures, and all other subsystems included to aid testing. Scan chains, multiplexers, and test busses are used in great detail in Chapter 4 to increase access to the MAE and MMR. Test controllers and interfaces are not addressed in this thesis because they do not directly affect the test procedures or other DfT components. Test controllers and interfaces (and sometimes the I/O pins required) are often specified by organizations to help standardize testing interfaces [8, 9], but say nothing about the tests being performed. BIST moves some portion of the test algorithm execution directly on-chip with the goal of reducing the burden on the tester, and allowing certain test parallelization to reduce test time. The impact of BIST on the algorithms presented in this thesis will be examined in Chapter 4.

Chapter 3
TCAM Spice Model and Cell Fault Analysis

3.1 TCAM Spice Model

A complete spice model of a dynamic TCAM array was constructed for fault analysis in 0.18 μm CMOS technology. The block diagram of this model is illustrated in Figure 3.1. This model is based on a commercial design [10]. It contains an array of 16 words by 144 bits dynamic TCAM.

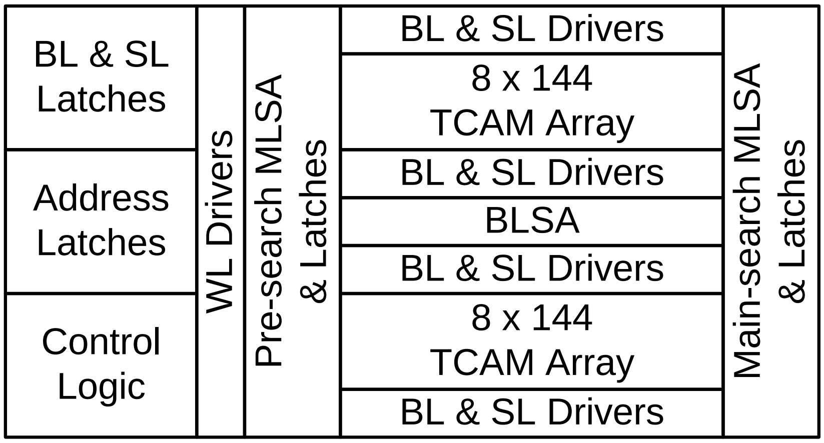


Figure 3.1 – TCAM Spice Model

The basic cell is the same as shown in Figure 2.1b). The BLs and SLs run vertically throughout the array and the WLs and MLs run horizontally. The model consists of the 16 words divided into two groups of eight with one dummy word above and below the BLSA block to aid in MLSA timing and operation. 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 TCAM 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 delay chain 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 has an un-optimized minimum cycle time of 7.2 ns, or 139 MHz. The model was not optimized for cycle time since performance was not the goal of this study.

The BLSA is a simple voltage-mode sense amplifier consisting of two tristate cross-coupled inverters. The MLSA is a current-mode sense amplifier, and a simplified diagram is shown in Figure 3.2. 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 thus saves 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, the 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. If the pre-search is a mismatch, the main-search operation is not executed on that word, thus saving energy. 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.

3.2 TCAM Cell Fault Analysis

Transistor level faults modeled included gate, source, and drain contact failures, subthreshold conduction due to poor L_{eff} control and wide V_t spread, gate oxide failures that lead to gate-source or gate-drain conduction, and inter-cell shorts due to metal spot defects. 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 3.3). Even though the functionality may be preserved, these faults are more insidious since the timing is affected by increased RC time constants.

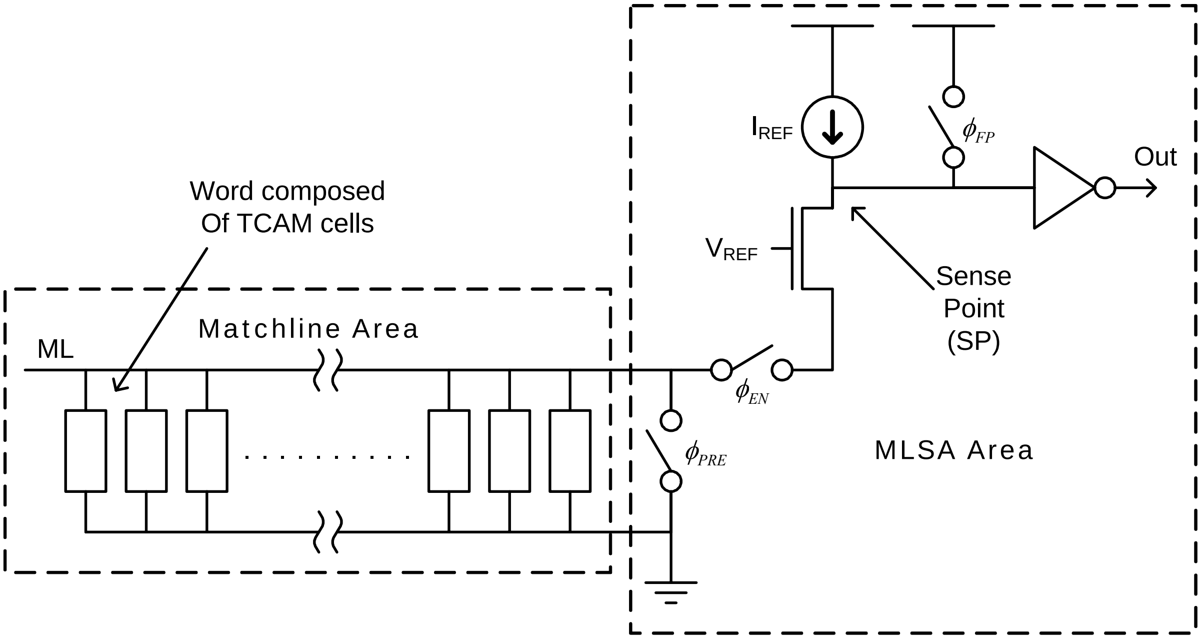


Figure 3.2 – Simplified MLSA

A transistor level fault analysis was performed on the 6-T TCAM cell. Examining possible transistor level faults yielded five possible circuit level representations for the faults, which are shown in Figure 3.4. These five circuit level fault representations were applied to each of the six transistors. Adding the possibility of a storage capacitor ground fault revealed 16 unique intra-cell faults to be examined. Inter-transistor coupling defects (caused by spot defects) resulted in nine additional faults for a total of 25 possible faults.

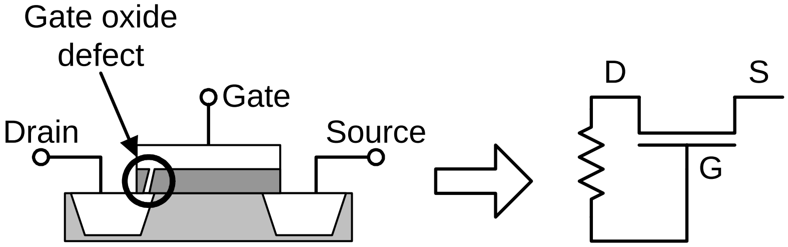


Figure 3.3 – Example Defect and Circuit Equivalence

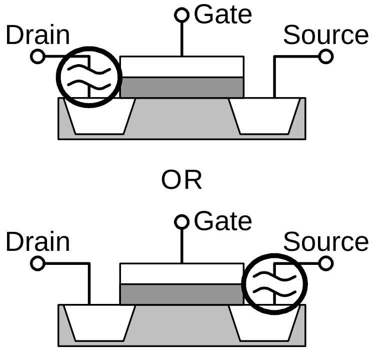
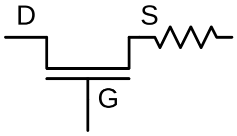
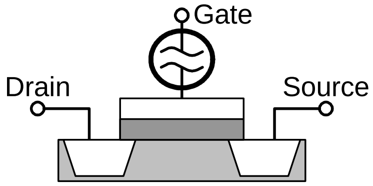
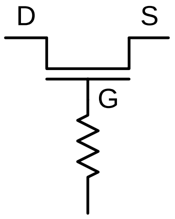
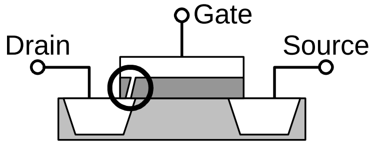
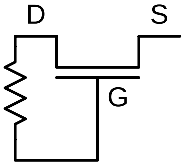
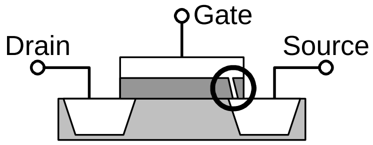
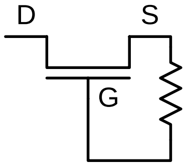
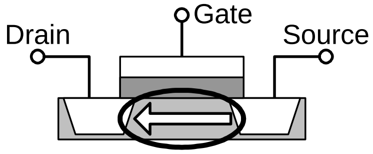
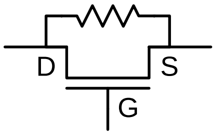
Defect Cause	Defect	Circuit Equivalent
Drain/Source Contact Defect		
Gate Contact Defect		
Gate to Drain Oxide Failure		
Gate to Source Oxide Failure		
Subthreshold Conduction		

Figure 3.4 – Transistor Level Defects Modelled

Analysis and simulation of these faults gave a detailed look into the possible failure modes of the TCAM cell and also helped determine how easily these faults can be detected. Some faults resulted in total failure of the cell, whereas other faults resulted in failure only under very specific operating conditions. This allowed the development of test methods that attempt to catch the subtle faults.

These methods are presented in Table 3.1. The Transistors, M1 through M6, refer to the cell depicted in Figure 2.1b). The last operation in each method in the column "Detection Method" refers to the result under correct operating conditions. For example, if Defect #11's test method results in a "mismatch", 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.

Table 3.1 – Possible TCAM Intra- and Inter-cell Defects and Detection Methods

#	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
17	M1 Gate to M2 Gate Short	(a) Write "0"; (b) SL2 = "1"; (c) Read "0"
18	M1 Drain to M4 Drain Short	(a) Write "1"; (b) SL1 = "1"; (c) Search for Mismatch
19	M1 Gate to M4 Drain Short	(a) Write "1"; (b) SL1 = "1", SL2 = "0"; (c) Search for Mismatch
20	M2 Gate to M4 Drain Short	(a) Write "1"; (b) SL1 = "1", SL2 = "0"; (c) Search for Match
21	M1 Gate to M4 Gate Short	(a) Write "1"; (b) SL1 = "1", SL2 = "0"; (c) Search for Match
22	M1 Gate to M5 Gate Short	(a) Write "0"; (b) SL1 = "1", SL2 = "0"; (c) Search for Mismatch
23	M2 Gate to M5 Gate Short	(a) Write "1"; (b) Read "1"
24	M1 Gate to M3 Gate Short	(a) Write "1"; (b) SL1 = "1", BL1 = "0"; (c) Search for Mismatch
25	M1 Drain to M3 Gate Short	(a) Write "0"; (b) SL2 = "1"; (c) Search for Mismatch

Using these defect models, simulations were performed to verify the effectiveness of the detection methods (Table 3.1) 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.

As an example, Figure 3.5 illustrates the detection method for Defect #2 (Table 3.1). 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 behaviour is illustrated in the graph in Figure 3.5. As shown in the figure, at the resistance value of 10 kΩ and above, M2 exhibits the stuck-open (SOP) behaviour for write times of 2.5 ns, which is limited by the control signal timing.

Most of the techniques presented in Table 3.1 require precise timing of on-chip control signals to control lines such as WLs, BLs, SLs, and MLs in order 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 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 3.5.

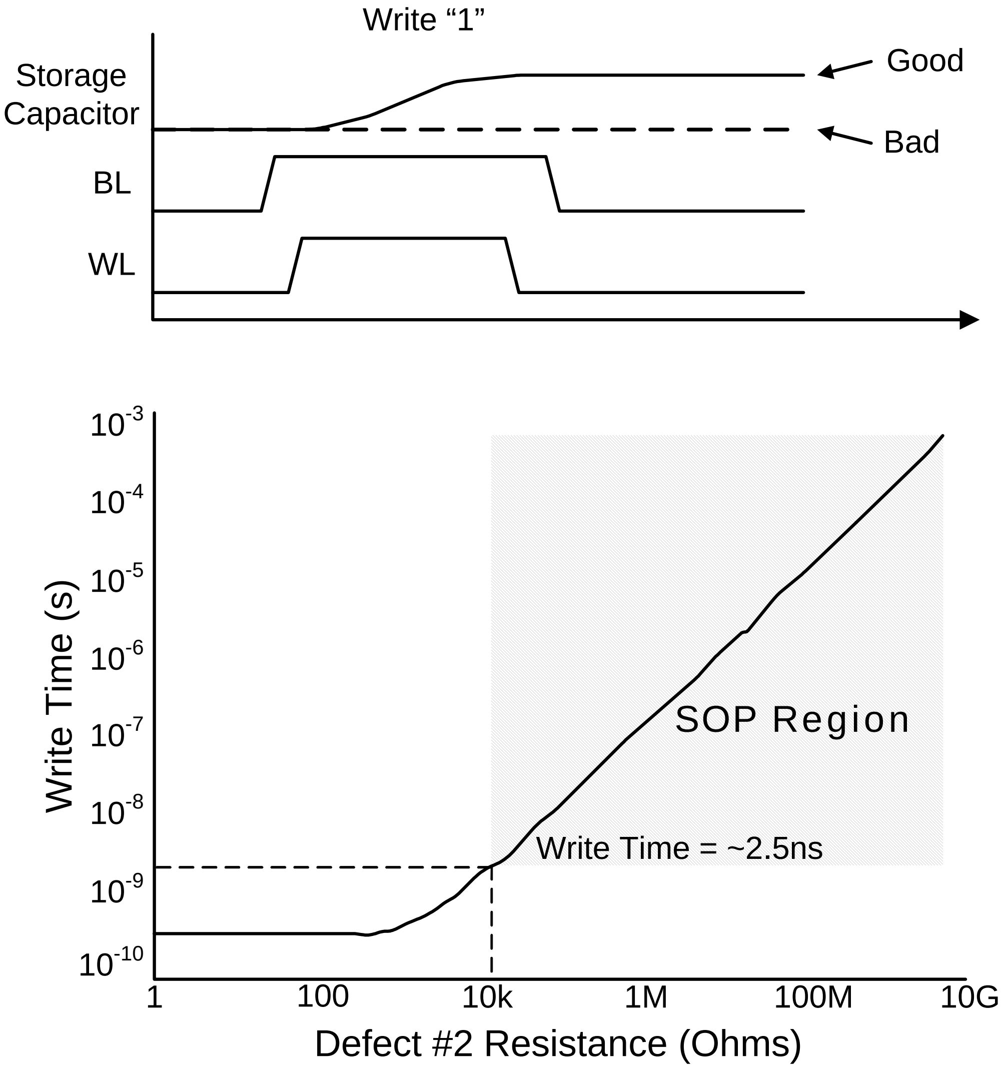


Figure 3.5 – Write Time vs. Defect #2 Resistance

Chapter 4
Testability Issues in TCAMs

In this chapter, we examine the testability aspects of TCAMs. 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 dynamic TCAMs, it is equally well suited to static implementations since the comparison logic can be used in conjunction with any static or dynamic storage cell. New test patterns and test structures are described that provide a full test suite designed with the special complexities of TCAMs in mind.

4.1 Previous Work on TCAM Testing

Until recently, TCAMs 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 TCAMs. The only previous works of note are [11-14] which focus on testing static binary CAMs, and [15] which focuses on TLBs, which are also static binary CAMs.

A recent patent [16] describes a TCAM built-in self test (BIST) scheme. From a BIST perspective, the patent deals only with the TCAM array and does not mention the MMR or MAE. The patent simply describes several possible permutations of methods used to get test data in and out of the TCAM array. From an algorithmic standpoint, a brute-force approach is taken and each bit is tested individually, resulting in the worst case complexity of O(nl). Thus, this patent serves as a good starting point for further research, but fails to make any unique or truly innovative claims.

4.2 Overview of TCAM Testing

TCAM testing presents new challenges, both in terms of test algorithm development and DfT. Similar to RAMs, a TCAM also contains address decoders, storage cells, and latches. Hence, there is some overlap between traditional RAM testing and TCAM testing. However, TCAMs also contain additional components such as bit-level comparison logic, MLSAs, wide-input MMRs, and MAEs.

These components require the development of new test strategies. As a result, a complete TCAM test strategy will incorporate both existing RAM tests, and new tests designed specifically for TCAMs.

The largest commercial TCAMs currently available have tens of millions of storage cells. Such a large number of cells makes TCAM testing very time consuming, thus the tests must be as efficient as possible. The tests must also have access to individual search-path components working both together and in isolation. Finally, due to the ubiquitous nature of defects in deep sub-micron technologies along with the large critical area of TCAMs, test strategies must be developed using defect analysis and not only using the simple stuck-at fault model.

TCAM 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 MLs. All ML discharge paths of a given word (2*l*) are connected to the corresponding ML. Therefore, each discharge path must be uniquely tested. In a complex TCAM, there could be 36 M such paths [17], making TCAM testing expensive.

The additional difficulty of TCAM testing lies in identifying and repairing faulty cells. In TCAMs it is easier to identify the faulty address (row), but it is more time consuming to find the individual faulty bit (column). Testing the TCAM cells using higher-level functionality presupposes that the MMR and MAE are functional and fault free. Therefore the MAE and MMR must first be tested and deemed functional to exploit the high-level Read, Write, and Search functions.

4.2.1 Proposed Test Algorithms

The test algorithms must strive towards full fault coverage, high efficiency, and ease of implementation to support existing testers. RAM testing is a mature research area since RAM applications are so prevalent. Hence, existing RAM tests, such as March-based algorithms, can be used to test the storage cells in TCAMs [18]. These algorithms require slight modifications to account for the two storage cells per TCAM cell, or a read and write mode built into the TCAM which allows decoupling of the individual storage cells in order to execute the RAM test algorithm directly without modification.

The comparison logic contained in every TCAM cell presents new test challenges. Since every TCAM cell can individually discharge its corresponding ML, every single discharge path in the TCAM must be verified. For example, testing an 18 Mb TCAM requires coverage of 36 M discharge paths. Basic TCAM test algorithms result in a complexity of $O(nl)$, where n is the number of words and l is the number of bits per word since each bit must be tested individually. The algorithms described in this thesis have reduced the complexity by exploiting the masking capabilities of TCAMs and result in a complexity of $O(nl / \log_2(n))$. Finally, circuits such as the wide-input priority encoder are more recent and have not had extensive test algorithm development. A basic functional test algorithm, along with the necessary on-chip test circuitry required to facilitate testing, is described later in this chapter.

4.2.2 Design for Test

The number of transistors per I/O pin is dramatically increasing as ICs become more highly integrated and feature sizes shrink. According to the 2003 International Technology Roadmap for Semiconductors, there are now close to 1 M transistors per I/O pin [5]. This makes I/O pins valuable, and hence testing should occupy as few pins as possible. If only a few I/O pins are available, off-chip testing suffers from reduced observability of the IC from the perspective of the tester.

Design for Test (DfT) seeks to alleviate these problems by incorporating circuits and structures onto an IC during the design phase that make testing easier, faster, and more effective. One aspect of DfT is that it can achieve low-speed testing of high-speed ICs using on-chip circuitry. Thus, older testers can be used and do not need to be replaced as often. Another aspect of DfT is the inclusion of on-chip structures that increase the accessibility and observability of resources that need to be tested. On-chip test circuitry’s view of the chip is not limited by the narrow window of available I/O pins.

Finally, BIST seeks to implement the necessary test algorithms directly on the die. The tests can be executed in parallel, require less tester bandwidth, and be fully customized to the IC. BIST can identify faulty components and replace them with the on-chip TCAM redundancy, and can also help the tester to determine whether the TCAM IC under test can be repaired or if it should be rejected. External testers are still required for other essential tests, like I_{DDQ} and parametric testing.

Figure 4.1 shows the DfT architecture that will be used in the remainder of this thesis for algorithm development. Two scan chains allow test vectors to be scanned in, and results to be scanned out, and several multiplexers control the movement of test and regular operational data throughout the IC.

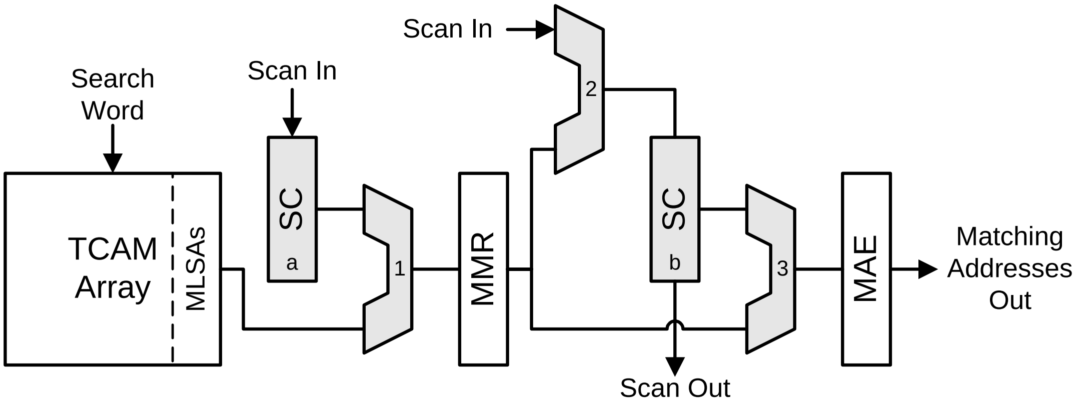


Figure 4.1 – DfT Structures

Because the search operation propagates from TCAM array to MMR to MAE, it is logical to test these structures in reverse order. The multiplexers allow the inputs and outputs of the various units to switch between the output of the previous stage and a scan chain. Since the multiplexers can be implemented using pass transistors, there is little area or delay penalty for including them into the search path of the TCAM. The multiplexers in front of scan chain “b” (SC-b) allow the scan chain either to input test vectors, or to store the output of the previous stage (MMR). The outputs are then shifted out serially to the test circuitry. For example, the outputs of the MMR can be examined before they enter into the MAE.

The TCAM array is tested using the MAE's output once the MMR and MAE have been tested. This compacts the results of the TCAM array test into matching addresses, eliminating the need for an additional scan chain to serially shift out the TCAM array test results. Using the MAE output compacts the results of the TCAM array test algorithm from n-bit vectors into $\log_2(n)$-bit matching addresses. The following sections assume that scan chains "a" and "b" (SC-a and SC-b), and multiplexers "1", "2", and "3" (Mux-1, Mux-2, and Mux-3) shown above in Figure 4.1 are implemented on the TCAM. An additional multiplexer, Mux-TB, is also implemented but not shown in the figure for simplicity. Its operation is discussed in subsequent sections.

4.2.3 BIST Architecture

Test algorithms can be run entirely on a tester with few test provisions designed into the chip. This option necessitates an advanced tester that is capable of executing all the necessary test algorithms and most likely would need to operate at the frequency of the IC. Including BIST circuits on the IC allows test vectors to be generated and results examined directly on chip. This reduces the burden on the tester, potentially allowing more ICs to be tested at once, and enables test parallelization.

Without BIST, testing typically must occur at the chip level (~64-128k words). With BIST, certain portions of the test algorithm can be executed at the block level (~128-256 words), on all blocks in parallel. This reduces the test complexity of the algorithms executed at the block level by a factor of x, where x is the number of blocks per chip. For example, an 18 Mb TCAM with 128 k words and blocks of 128 words each benefits from a test time reduction of 1024x. Given the size and complexity of TCAMs, DfT and BIST are essential for efficient and cost effective testing.

4.3 DRAM Testing

The storage section of the TCAM 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 [19]. An example of a modern test interface for embedded DRAMs is [20].

The testing of DRAMs applies equally to the access transistors and storage capacitors of the TCAM. 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 types of 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 TCAM cell. If a static implementation is used, SRAM tests should be applied to the TCAM.

When choosing a RAM test algorithm, it should be noted that there are two storage cells per TCAM cell. Thus, storing a logical “1” in the TCAM actually results in the physical storage of a “1” and a “0”, and vice versa. The chosen algorithm must be slightly modified to take these differences into account. Otherwise, a read and write mode which allows decoupling of the individual storage cells must be incorporated into the TCAM in order to execute the RAM test algorithm directly without modification.

4.4 Search Path Testing

The search path consists of the comparison logic, the MMR, and the MAE. The comparison logic consists of the ML and the four discharge transistors in every 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 thesis 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 comparison logic transistor SON and SOP faults. It was assumed during the development of this algorithm that the TCAM is able to return all matched addresses sequentially, as was the TCAM being modeled. In the case of a TCAM 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.

The MAE is tested at the chip level by examining its ability to encode every possible address. As previously discussed, the MMR is a tree-structure of smaller 16- or 32-bit MMRs. It could be efficiently tested initially at the block level, but must also be tested at the chip level. The TCAM array can be tested at any level, but testing at the block level would enable time saving parallelism. Each of these aspects will be discussed in the following sections.

4.4.1 MAE Testing

4.4.1.1 MAE DfT Components

Figure 4.2 shows a detailed view of the test structures in place between the MMR and the MAE. It shows the registers of SC-b, and Mux-2 and Mux-3 that connect and disconnect them, as depicted in Figure 4.1. The blocks labelled “R” are the registers of the scan chain, and the switches represent multiplexers. The MAE is tested by encoding every possible address and examining the output’s validity. Even at the chip level this results in only *n* tests, one for each address, and ensures that the complete hierarchical MAE is fully functional.

When the multiplexers are in position “a”, the scan chain is capable of storing all the outputs of the MMR in parallel. The values can then be shifted out serially to a tester by switching Mux-2 to position “b”. Alternately, test vectors can be shifted into the scan chain by keeping Mux-2 in position “b”. The MAE inputs are connected to multiplexers that allow either the MMR’s outputs to pass directly to the MAE (Mux-3 position “a”), or the scan chain’s values to pass (Mux-3 position “b”). Thus, for regular operation, the switch would be in position “a” and the scan chain would not be used. These modes are described in Table 4.1.

Table 4.1 – MMR/MAE Mux Positions and Corresponding Functionality

Mux-2 Position	Mux-3 Position	Function
a	a	Normal operation / Store MMR outputs in SC-b
b	a	Normal operation / Shift SC-b In/Out
a	b	Not used
b	b	MAE connected to SC-b / Shift SC-b In/Out

4.4.1.2 MAE Test Algorithm

An MAE is configured similarly to the one shown in the 3-bit example in Figure 2.7. To test the MAE, the test controller (either an off-chip tester or a BIST controller) connects the MAE's inputs to SC-b using Mux-2 and Mux-3 in position "b", as they are shown in Figure 4.2.

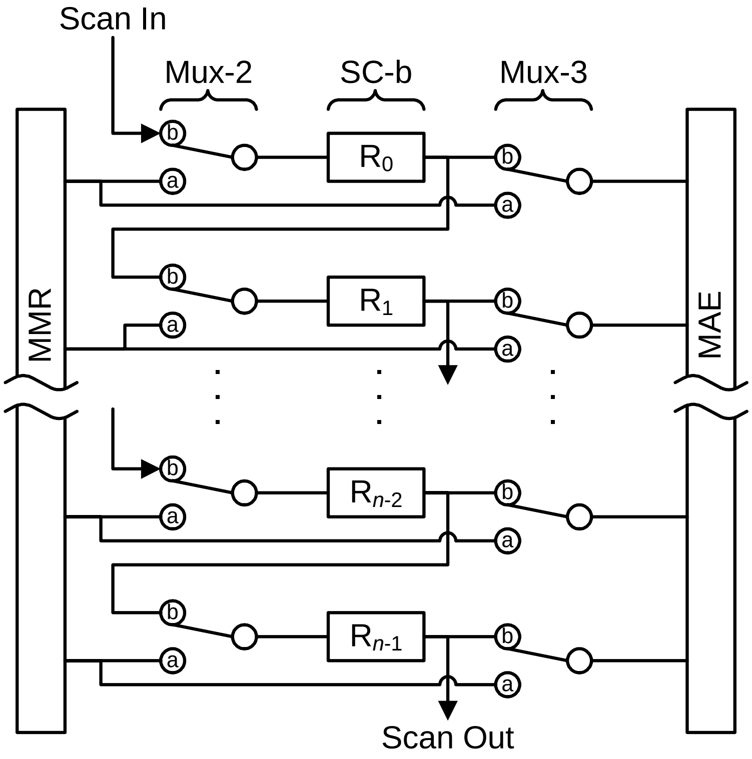
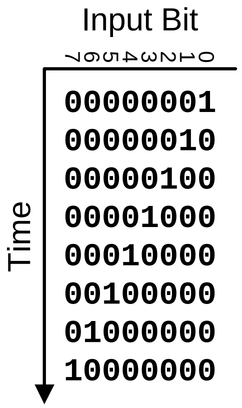


Figure 4.2 – Detailed View of MMR/MAE Scan Chain

A string of zeros is first shifted into the scan chain, or the scan chain registers are simultaneously reset to zero depending on the kind of registers used. A "1" is shifted into the scan chain followed by more zeros. These input test vectors are shown in Figure 4.3 for the first eight input bits of the MAE. Every time the “1” is shifted, the output is examined to ensure that the address has been correctly encoded. Typically the address would begin at zero and increment each shift until the last address is detected at *n* – 1, where *n* is the number of words. Any time the output address does not match the expected address, there is a fault present at the expected address in the MAE.

This procedure exhaustively tests every address. Due to the nature of the MMR, it is unnecessary to apply multiple high (“1”) inputs during testing of the MAE because this input condition should never occur, and the MAE is not typically designed to handle this type of erroneous input. The MAE test is of complexity O(*n*). For example, in a 64k-word TCAM, the test would take 64k shifts to move the “1” through the scan chain plus the amount of time it would take to reset the scan chain to all zeros.



```
00000001
00000010
00000100
00001000
00010000
00100000
01000000
10000000
```

Figure 4.3 – MAE Input Test Vectors

4.4.2 MMR Testing

4.4.2.1 MMR DfT Components

Due to the technical impracticalities of directly creating an *n*-bit MMR using traditional designs, a tree structure of smaller MMRs is used to create a distributed *n*-bit MMR, as shown in Figure 2.6. This tree-structure of smaller MMRs must eventually be tested at the chip level to ensure higher-level functionality. However, by first testing the nodes of the MMR tree, the complexity of testing the full MMR tree is greatly reduced, eliminating special test cases between nodes of the MMR tree. Using appropriately connected multiplexers, the MMR tree is disconnected to separate each node. Then each node, which is typically a small 16- or 32-bit MMR, is tested in isolation. The tree is then reconnected and tested as a whole.

This scan chain and multiplexer configuration is depicted in Figure 4.4 for the L1 MMR. Without any DfT components in place, the outputs of the MLSAs would be directly connected to the MMR. Mux-1 is inserted to allow the inputs of the MMR to switch between the MLSA outputs for regular operation in position "a", and scan chain register outputs for MMR testing in position "b". These test modes are summarized in Table 4.2.

Table 4.2 – MMR Mux Positions and Corresponding Functionality

Mux-TB Position	Mux-1 Position	Function
a	a	Normal operation / SC-a parallel load from TB
b	a	Normal operation / SC-a serial load from TB
a	b	MMR connected to SC-a / SC-a parallel load from TB
b	b	MMR connected to SC-a / SC-a serial load from TB

The input of every sixteenth bit of SC-a is either a 1-bit MMR test bus (TB), or the output of the previous scan chain register. The Mux-TB serves to switch between the resulting parallel 16-bit scan chains in position "a", and one long serial scan chain in position "b". When Mux-TB is in position "a", every sixteenth scan chain register shifts in new test values from the 1-bit MMR TB, thus acting as numerous parallel 16-bit scan chains. When Mux-TB is in position "b", every scan chain register loads its new value from the previous register, thus acting as one long serial scan chain. The scan chain registers in Figure 4.4 correspond with SC-a in Figure 4.1.

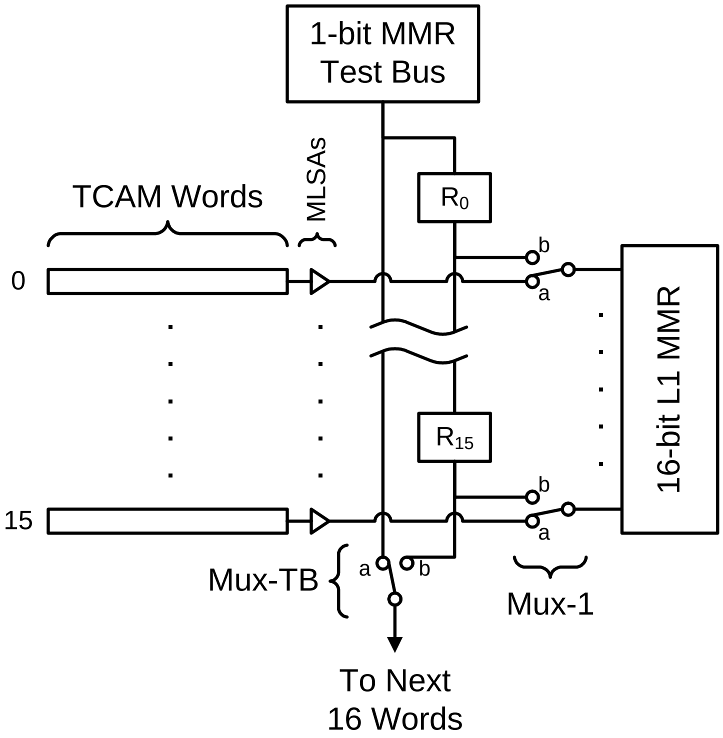


Figure 4.4 – L1 MMR Test Bus Configuration

Figure 4.5 shows the scan chain and multiplexer configuration for the L2 MMRs. It is essentially the same as for L1 MMRs with the exception that the input signals are normally the “match found” signals from the L1 MMRs, and the outputs are connected to the enable inputs of each L1 MMR. Thus, when the MMR is being tested as disconnected nodes, all MMR nodes must be individually enabled to circumvent deactivation by higher-level MMR nodes.

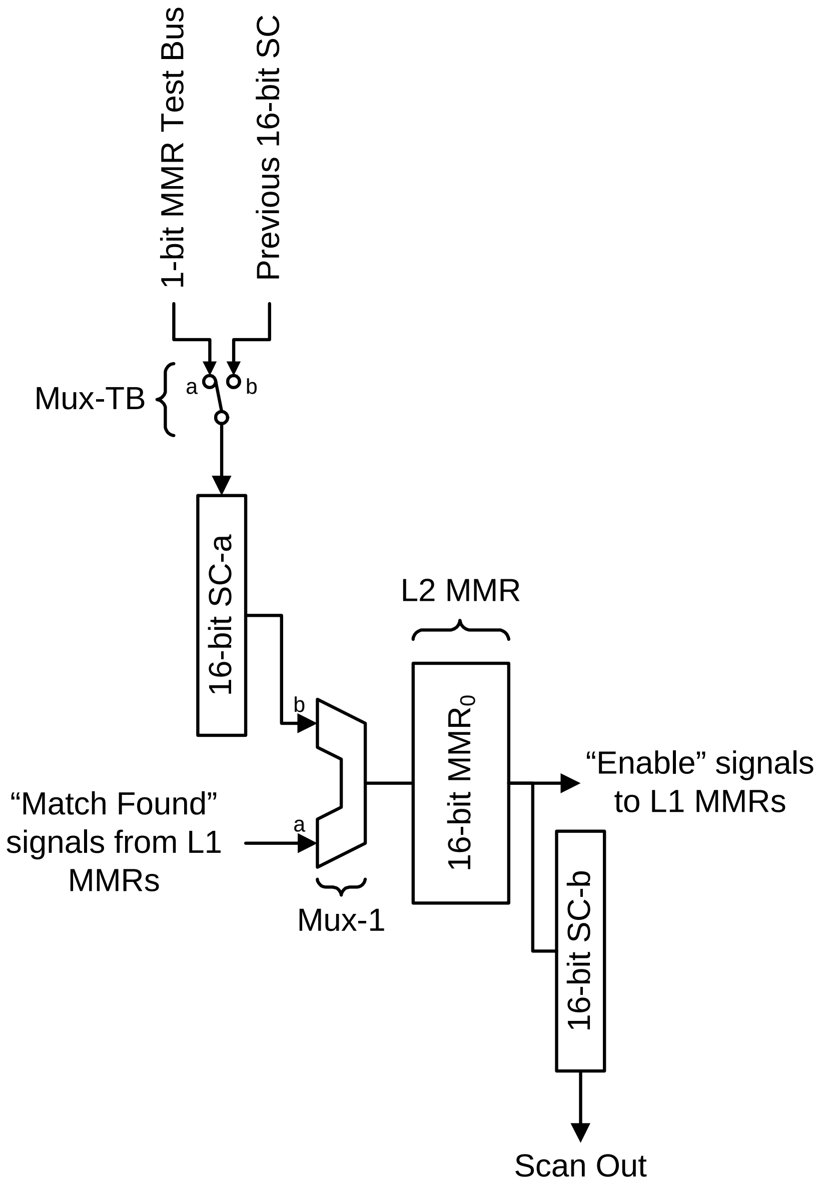


Figure 4.5 – L2 MMR Test Bus Configuration

4.4.2.2 MMR Test Algorithm

Initially, the entire SC-a is reset to "0", Mux-TB is set to position "a" (i.e. SC-a is configured as parallel 16-bit scan chains), and SC-b is used to capture the output of the MMR. Since exhaustive testing of p-bit L1 MMRs would require 2^p test vectors, and each vector would require p shifts to load in the p-bit parallel scan chains, it would take $p\cdot 2^p$ shifts to load the exhaustive test vectors. This complexity can be reduced to 2^p by using a pseudo-random binary sequence (PRBS) generator [21]. However, shifting out the results would require n shifts of the serially connected SC-b, thus eliminating any benefit of using a PRBS. Exhaustive testing would therefore take $n \cdot 2^p$ total shifts to accomplish. Assuming block level testing is possible, blocks of 128 words, and $p = 16$, this would result in $128 \cdot 2^{16} = 2^{25} = 33.6$ M shifts. Under most circumstances, this is too costly. A basic functional test is presented here that attempts to verify the proper functionality of the L1 MMRs while saving time.

The functional test requires the same initial configuration as stated earlier: Initially, the entire SC-a is reset to "0", Mux-TB is set to position "a" (i.e. SC-a is configured as parallel 16-bit scan chains), and SC-b is used to capture the output of the MMR. A string of "1"s are shifted into the scan chains, and the L1 MMR outputs are checked to verify that each new "1" inputted into the scan chains results in the next higher priority output than the previous "1". This pattern is depicted in Figure 4.6 assuming that the least-significant bit (LSB) is the highest priority. Once all the small L1 MMRs are verified, the tree is reconnected ("match found" and "enable" signals to and from L2 MMRs are re-established), and the inputs to the scan chains are set to the outputs of the previous scan chain (Mux-TB in position "b"). Now, the MAE output can be used to compact the results into matching addresses since the MAE has already been tested and deemed functional. Any error with the MMR blocks connected as an *n*-bit MMR will result in an incorrect address being outputted by the MAE. The MMR tree is tested as a whole by resetting SC-a to all "0"s, then by shifting in a string of "1"s. Again the outputs ensure that each new "1" introduced into the scan chain has a higher priority than all the previous "1". Consequently, the string of "1"s should proceed from lowest priority input to highest.

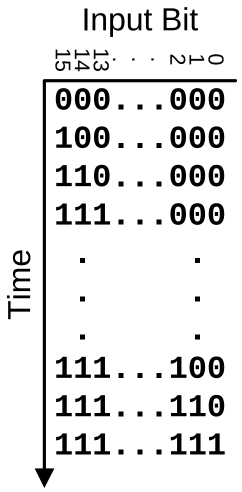


Figure 4.6 – MMR Input Test Vectors

It should be noted that each level of the MMR tree is not limited to using 16-bit MMRs. Theoretically, they can be made of any size MMRs, but typically they are 16- or 32-bits for practical reasons. One level of the tree need not have the same MMR input size as the previous or following level. The test algorithm presented is easily extendable to any sized MMR. This thesis assumes 16-bit MMRs to facilitate algorithm development and presentation. For full MMR testing, the same pattern shown in Figure 4.6 can be extended to test all *n*-bits of the MMR. This results in *n* + 1 input patters for the serial scan-in (full tree) test mode of the MMR.

4.4.3 TCAM Array Testing

Once the MAE and MMR are known to be functional, the following TCAM array test algorithms can be executed without any special DfT components to increase observability and controllability. This algorithm 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 is designed to detect all intra-cell defects leading to SON and SOP faults, and inter-cell coupling faults. Also, the row and column of a faulty cell are simultaneously extracted from the results of this algorithm. Thus, it is equally amenable to both row and column redundancy.

The algorithm exploits the masking capabilities of the TCAM to test many bits at once instead of existing methods that test only one bit at a time. The test was designed based on the preceding defect analysis of a commercial TCAM, and the results were found to be applicable to any TCAM that uses an ML architecture. Commercial standalone TCAMs use ML architectures, so these results are very portable. An example of a non-ML architecture would be a software program that uses RAM to mimic the functionality of a TCAM.

To help in visualizing the match operation, Figure 4.7a) shows a symbolic representation of the comparison logic transistors. BL1, BL2, SL1, and SL2 represent the transistors located in the comparison logic. For example, if the cell is storing a “1”, then BL1 = “1” and BL2 = “0” as shown in both Figure 4.7b) and Figure 4.7c). The available states are summarized in Table 2.1. In Figure 4.7b), the bit being searched for is also a “1” since SL1 = “1” and SL2 = “0”, so there is a match condition and no path exists for the ML to discharge. However, if a “0” is being searched for, as in Figure 4.7c), then SL1 = “0” and SL2 = “1”, creating a path for the ML to discharge, thus indicating a mismatch. Though the BLs are not directly tied to the gates of the transistors in the comparison logic, the symbols BL1 and BL2 are used to indicate the values stored on the capacitors since these values initially came from the BLs. This is the reason why this comparison logic configuration can be used with static memory architectures. As long as the gates of BL1 and BL2 are driven by the values stored in the memory cells, the type of memory cell is irrelevant.

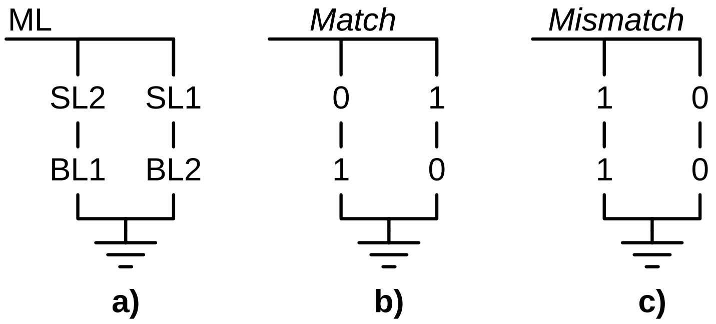


Figure 4.7 – Comparison Logic Symbolic Representation

4.4.3.1 Intra-cell Testing Overview

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 4.8a).

If the expected address is not returned, this indicates a 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 4.8b).

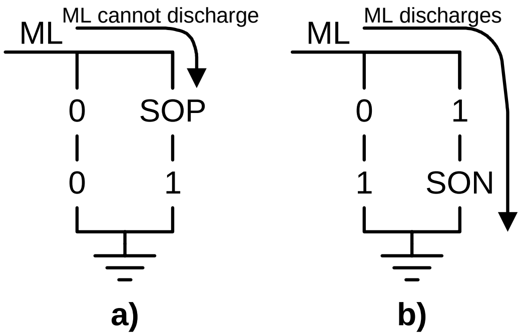


Figure 4.8 – Comparison Logic with a) SOP and b) SON Faults

The intra-cell algorithm tests the four comparison logic transistors of every TCAM cell. 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 BL SON faults by looking for missing expected matching addresses. 2c) Check for SL SON faults at addresses that failed the BL SON test. 3) Repeat the procedure using inverted values when writing and searching.

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, by only assigning unique values 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. Since the number of bits per word, l, typically exceeds the number of bits in the address, $\log_2(n)$, many padding bits are needed per word when writing unique values. However, the above mentioned procedure has some shortcomings. It takes a lengthy procedure to search for SON faults in the padding bits because the locations of SON faults in the padding bits are indistinguishable from each other.

An alternative, efficient arrangement can be realized if we divide the word length into logical columns of 16 bits each (i.e. the same number of bits as the address space, which is $\log_2(n)$). In every logical column the same data pattern is repeated, as shown in Figure 4.9 for a small 16 x 16 TCAM. 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 4.10 for the same small 16 x 16 TCAM. The word is divided into logic columns of 4 bits each.

A diagram of the algorithm’s flow is presented in simplified form in Figure 4.11. 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.

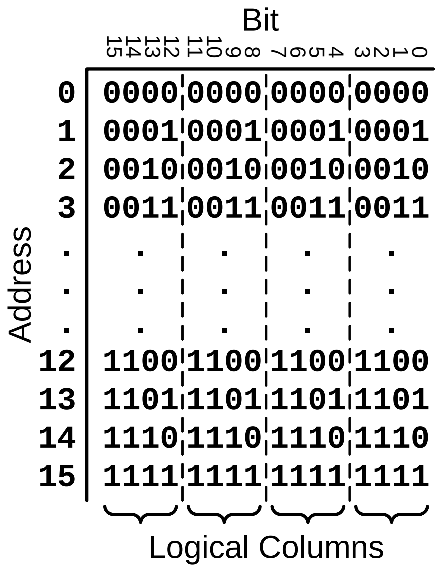


Figure 4.9 – Memory Space with Logical Columns

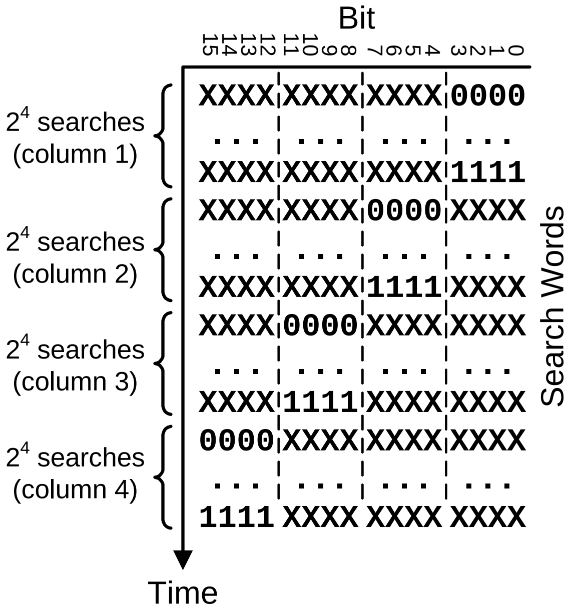
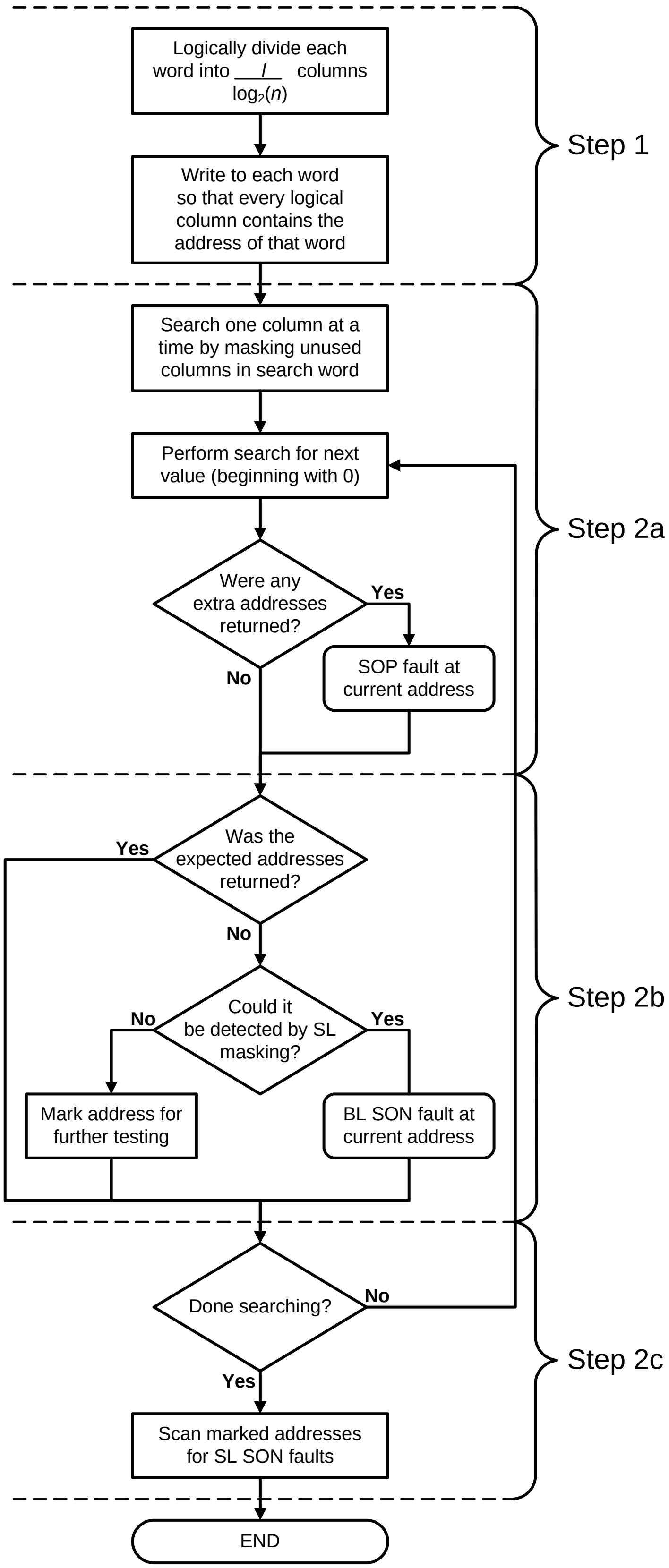


Figure 4.10 – Search Patterns for Increasing Values



Logically divide each word into $\frac{l}{\log_2(n)}$ columns
Write to each word so that every logical column contains the address of that word
Step 1
Search one column at a time by masking unused columns in search word
Perform search for next value (beginning with 0)
Were any extra addresses returned?
Yes
No
SOP fault at current address
Step 2a
Was the expected addresses returned?
Yes
No
Could it be detected by SL masking?
No
Yes
Mark address for further testing
BL SON fault at current address
Step 2b
Done searching?
No
Yes
Scan marked addresses for SL SON faults
Step 2c
END

Figure 4.11 – Intra-cell Test Algorithm Overview

4.4.3.2 Intra-cell Test Algorithm

This algorithm is composed of three steps, two of which are summarized in Figure 4.11. 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 $l/\log_2(n)$ 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. Therefore, 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.

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 TCAM is then searched using 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 located at the unexpected returned addresses. If one of the BL or SL transistors has an SOP fault, then a transistor is 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 located 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 the 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. The consequence is a false mismatch. To find the location of the SON fault, the corresponding search word for the missing address is repeatedly searched. However, each time it is searched, one of the bits of the unmasked portion of the search word is replaced with a "don't care". If the expected address is returned during this process, then the SON fault is located in a BL transistor in the bit that was just masked when the correct address was returned. If the expected address is not returned, the masked bit is unmasked and a different bit is masked. Typically the procedure would start by testing the least-significant bit (LSB) of the logical column and then linearly testing each successive bit. 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 (Step 2c) finds the defective bit. A binary search tree algorithm cannot be used with SL masking since too many matching addresses would be returned. Thus, a linear search is used, masking one bit at a time so that only one extra unnecessary address is returned each search.

Step 2c: The opposite approach of 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 uses BL masking to find the faulty bit location. A binary search tree algorithm can be used, and would begin by rewriting the data at the address being searched, except that half of the word would be masked. When searched using a partially masked stored word and unmasked search word, if the address is successfully returned, then the faulty bit is contained in the masked portion of the stored word. Otherwise it is contained in the unmasked part of the word. The algorithm proceeds recursively by further dividing the faulty half of the stored word 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.

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 (i.e. $l/\log_2(n)$ 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 searched only 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 resulting consequence is that the bits that are contained in both a regular column and the extra column will be tested twice.

4.4.3.3 Inter-cell Testing Overview

Figure 4.12 shows how surrounding cells can influence the operation of a TCAM cell in the presence of coupling faults. These coupling faults can lead to corrupted stored data. Since bridging faults due to excess deposits during fabrication are currently the dominant manufacturing defect [22], therefore the detection of coupling faults is critical in deep sub-micron technologies.

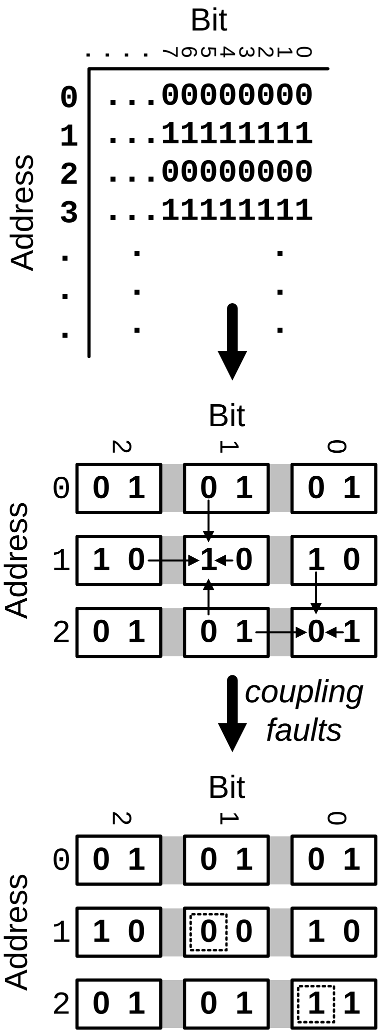


Figure 4.12 – Influence of Surrounding TCAM Cells

4.4.3.4 Inter-cell Test Algorithm

This algorithm is composed of six steps which are summarized in Figure 4.13. Steps that call for themselves to be repeated using inverted search data do not show these repeats in the figure for the sake of clarity. 1) Write a pattern to memory to expose horizontal and vertical inter-cell coupling faults. 2) Detect if a cell has flipped stored value from either “1” → “0” or vice versa. 3) Repeat 1) and 2) respectively using inverted stored data. 4) Write a pattern to memory to expose diagonal inter-cell coupling faults. 5) Detect if a cell has flipped stored value from either “1” → “0” or vice versa. 6) Repeat 4) and 5) respectively using inverted stored and search data. An explanation of each step follows.

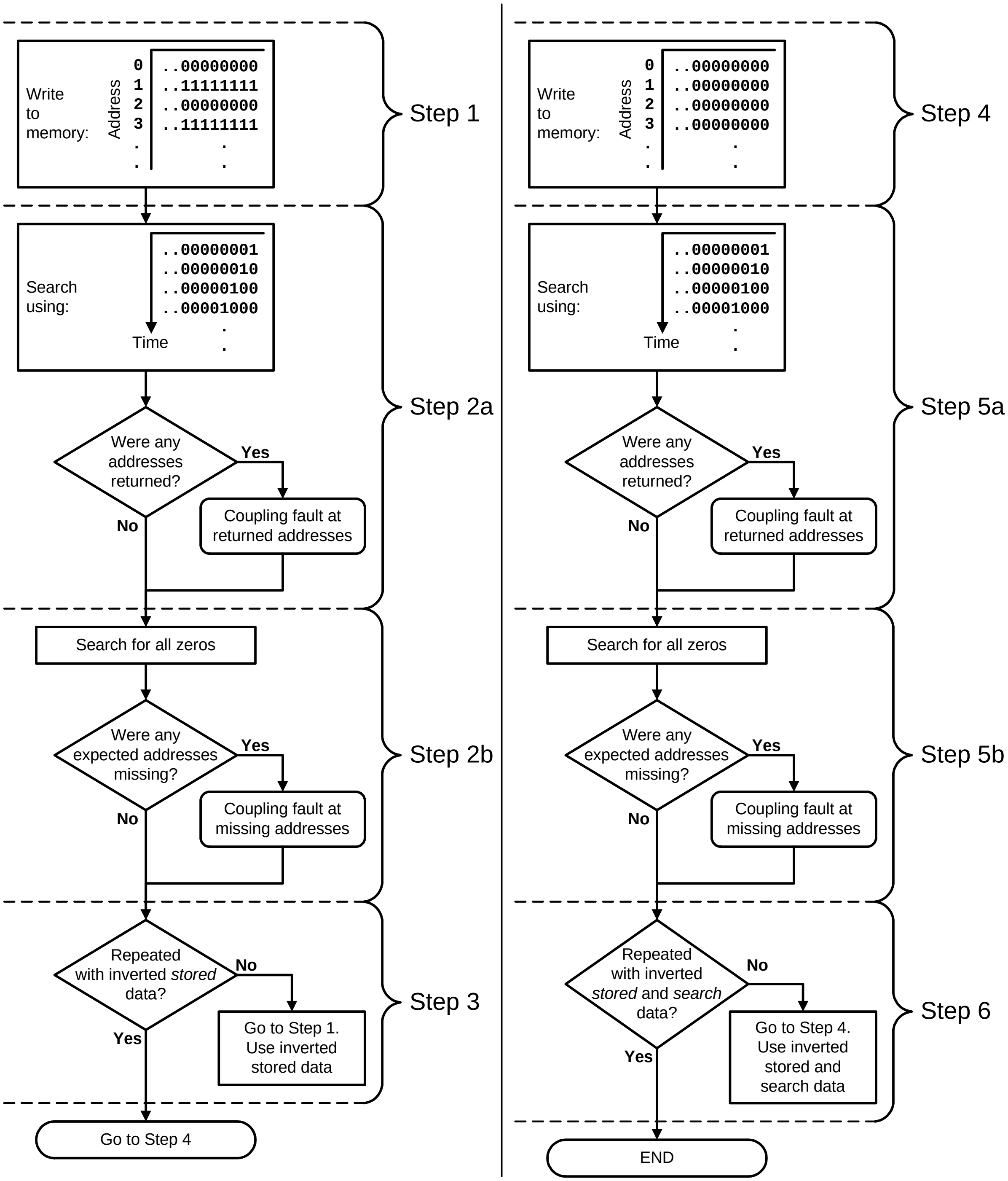

Write to memory:
Address
0 ..00000000
1 ..11111111
2 ..00000000
3 ..11111111
Step 1
Search using:
..00000001
..00000010
..00000100
..00001000
Time
Were any addresses returned?
Yes
No
Coupling fault at returned addresses
Step 2a
Search for all zeros
Were any expected addresses missing?
Yes
No
Coupling fault at missing addresses
Step 2b
Repeated with inverted *stored* data?
No
Yes
Go to Step 1. Use inverted stored data
Step 3
Go to Step 4
Write to memory:
Address
0 ..00000000
1 ..00000000
2 ..00000000
3 ..00000000
Step 4
Search using:
..00000001
..00000010
..00000100
..00001000
Time
Were any addresses returned?
Yes
No
Coupling fault at returned addresses
Step 5a
Search for all zeros
Were any expected addresses missing?
Yes
No
Coupling fault at missing addresses
Step 5b
Repeated with inverted *stored* and *search* data?
No
Yes
Go to Step 4. Use inverted stored and search data
Step 6
END

Figure 4.13 – Inter-cell Test Algorithm Overview

Step 1: This step is responsible for writing a pattern to the storage cells which causes alternating ones and zeros in the storage cells. In Figure 4.13, Step 1 shows a pattern which, when written to memory, exposes horizontal and vertical inter-cell faults. The alternating rows of ones and zeros in the storage cells enable horizontal and vertical inter-cell faults to affect the stored values. When a coupling fault is present it may alter the value of an adjacent cell, possibly pulling it to “0”, which can change the TCAM cell’s value to a “don’t care” condition, or pulling it to “1”, which results in faulty mismatches as was depicted in Figure 4.12.

Step 2: This step looks for cases where horizontal and vertical inter-cell coupling faults have either caused a stored “1” to flip to a “0” (Step 2a), or where a stored “0” has flipped to a “1” (Step 2b).

Step 2a: The search words in Figure 4.13, Step 2a will detect when an inter-cell coupling fault has caused a stored “1” to flip to a “0” in the words containing all zeros. In the fault-free case, there will be no matches to the aforementioned search words. If a match is returned, then there is a coupling fault in the word at the returned address, and in the bit location that corresponds with the location of the “1” in the search word. Repeat this step using inverted search data, this time to detect when an inter-cell coupling fault has caused a stored “1” to flip to a “0” in the words containing all ones.

Step 2b: Search using a search word of entirely zeros. Half of the addresses, namely the words storing all zeros, should match. If any addresses are missing, then there was a faulty mismatch at the missing address. If the exact faulty cell must be identified, then a binary search tree must be used as in Step 2b in 4.4.3.2. Otherwise, the faulty word can be replaced using row replacement. Repeat this step using a search word of entirely ones. This will test the words storing all ones.

Step 3: Steps 1 and 2 are repeated using inverted stored data *only*.

Step 4: This step is responsible for writing a pattern to the storage cells which causes aligned ones and zeros in the storage cells. Writing all zeros to memory exposes diagonal inter-cell faults.

The aligned rows of ones and zeros in the storage cells enable horizontal and vertical inter-cell faults to affect the stored values.

Step 5: This step looks for instances where diagonal inter-cell coupling faults have either caused a stored “1” to flip to a “0” (Step 5a), or where a stored “0” has flipped to a “1” (Step 5b).

Step 5a: The search words in Figure 4.13, Step 5a will detect if an inter-cell coupling fault has caused a stored “1” to flip to a “0”. In the fault-free case, there will be no matches to the aforementioned search words. If a match is returned, then there is a coupling fault in the word at the returned address, and in the bit location that corresponds with the location of the “1” in the search word.

Step 5b: Search using a search word of entirely zeros. All of the addresses should match. If any addresses are missing, then there was a faulty mismatch at the missing address. If the exact faulty cell must be identified, then a binary search tree must be used as in Step 2b in 4.4.3.2. Otherwise, the faulty word can be replaced using row replacement.

Step 6: Steps 4 and 5 are repeated using inverted stored data *and* search words.

4.4.4 Test Summary

The various test algorithms, along with the multiplexer and scan chain states necessary to execute them, are summarized in Table 4.3. The MAE test uses SC-b as the input and checks for correct addresses at the MAE output. Mux-2 and Mux-3 are configured so that SC-b acts as a shift register. The MMR node test uses SC-a to shift in 16-bit test vectors, and uses SC-b to first capture the outputs (Mux-2 in position “a”), and then to shift the data out (Mux-2 in position “b”). Mux-TB is set to position “a” so that SC-a acts as parallel 16-bit scan chains and Mux-1 in position “b” sets the inputs of the MMR to SC-a. For MMR full tree testing, Mux-TB is set to position “b” so that SC-a behaves as a long serial scan chain, and Mux-1 in position “b” sets the inputs of the MMR to SC-a. Mux-3 is set to position “a” so that the MAE can be used to compact the test results. Finally, for TCAM array testing, the TCAM is configured for normal operation by setting Mux-1 and Mux-3 to position “a”. The standard user interface, consisting of address, data in, and search data in are used as inputs, and the outputs are the MAE output and data out (for RAM tests).

Table 4.3 – TCAM Test Summary

Test	Input	Output	Mux Positions
MAE	SC-b	MAE Output	Mux-2 position “b” Mux-3 position “b”
MMR Nodes	SC-a	SC-b	Mux-TB position “a” Mux-1 position “b” Mux-2 alternates (store/shift)
MMR Full	SC-a	MAE Output	Mux-TB position “b” Mux-1 position “b” Mux-3 position “a”
TCAM Array	User Interface (address/data/search)	MAE Output	Mux-1 position “a” Mux-3 position “a”

4.5 Algorithm Validation

The algorithms proposed for the MMR and MAE are functional test algorithms and not defect-oriented. The reasoning behind this is twofold: First, MMRs lack the same level of research and patent publication that TCAM arrays have achieved. This makes choosing a circuit on which to perform a defect analysis difficult, and the resulting algorithm may not be very portable to future designs. Furthermore, the MMR and MAE are not designed to be directly accessible and are only accessible either through regular functionality or through special DfT circuits. As a result, during testing a serial interface is used to load test vectors into an *n*-input device. Thus, every test vector requires *n* shifts to load the vector and to read the test vector’s output. This can become excessively time consuming unless the number of vectors is minimized. To limit the time and complexity of the MMR and MAE testing, simple functional tests designed to verify the correct output of a set of basic test vectors are used. Thus, there is a trade-off between fault coverage and test time.

The algorithms proposed for the TCAM array are defect-oriented, and the bandwidth into the array is not limited by a serial interface as is the case for the MMR and MAE. A MATLAB model was created consisting of scripts and functions that simulate the higher-level functionality of the TCAM array. 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 TCAM array test algorithms.

For example, the search function in MATLAB sets a variable “ML” = 1 to represent precharging of an 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 TCAM hardware, the TCAM’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, the algorithm was verified using 144-bit words and 16-bit addresses.

The functions necessary for the TCAM array test algorithms in this thesis are *Write*, *Search*, and *Find*. The function *Write*(*word*, *addr*) stores *word* at address *addr*. This represents the write functionality of a TCAM. In the algorithm, it is assumed that *Search*(*word*) returns an array of addresses that contain *word*. This represents the search functionality of a TCAM. The basic premise of simulating the search functionality is that the ML 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 2.1. Similarly, BL1 and BL2 are also not necessarily complementary.

Finally, the *Find*(*condition*) function parallels that of the MATLAB “find” function which returns the indices of an array for which the condition is true. For example, find ([3 1 2 4 5 2] == 2) returns [3 6], and find ([4] == 2) returns [ ]. The use of this function dictates that the tester used to test the dies must make decisions during the execution of the test algorithm, either at run time or it must generate a second follow-up round of testing based on the results of the first round. The *Write* and *Search* functions are implemented in hardware on the TCAM.

4.6 Algorithm Benefits and Weaknesses

There are three main algorithms presented in this thesis. They are designed to test the MAE, the MMR, and the TCAM array. These algorithms were developed assuming that specific test structures (scan chains and multiplexers) are included on the chip. The benefits of the MAE test algorithm are that it exhaustively tests the MAE's ability to encode every address, and that it only takes n shifts to accomplish. The MMR test algorithm utilizes the tree configuration of the MMR to individually test each node of the tree (small 16- or 32-bit MMRs) before connecting them as a whole. This enables early detection of faulty MMR blocks and simplifies fault identification. The MMR can then be tested as a whole using test patterns that are designed to stimulate the connections between the nodes of the tree. The MMR node test is limited because it has to shift the outputs out of a scan chain (SC-b) after each pattern is inputted. However, its test complexity is still of the order n. The MMR full tree test uses the MAE to encode the result as an address for quick reading, and testing takes $n + 1$ shifts to accomplish. The TCAM array is tested for intra-cell transistor level faults, which is of the order $nl/\log_2(n)$. It is then tested for inter-cell transistor level faults, which is of the order n.

The TCAM array test 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 can determine the exact faulty transistor by examining the location of the faulty returned addresses relative to the expected returned address; however, it is typically not needed in the redundancy and repair scenario. Arguably, this function may be needed for defect data collection and yield improvement. This information is available and can be collected and used.

The main weakness of the MMR test algorithms presented in this thesis is the serial nature of the scan chains. This increases the number of cycles needed to load each test vector and to read out the results. Parallel busses could be used to speed data entry during MMR node testing, but would require more I/O pins and more routing. A PRBS generator can reduce the scan-in of vectors from $p{\cdot}2^p$ to only 2^p, but the scan-out of results still suffers from $n{\cdot}2^p$ complexity. The major weakness of the TCAM array test algorithm lies in the detection of BL and SL SON faults, which 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 problem would be to take the addresses marked for SL and BL SON fault testing and to test every bit in sequence, lessening the decision making required of the tester. The resulting data could be analyzed offline and no decision making would be required of the tester. The other side effect is that the total test time becomes weakly dependent on the number of SON faults.

An additional benefit of several of the algorithms presented is that they are amenable to block level parallelization through BIST. Each block could include its own BIST controller that applies test vectors and examines test results. The BIST controller could then relay this information to an off-chip tester to enable redundancy replacement, or to indicate that the IC must be discarded.

4.6.1 Existing Test Algorithm Complexity

An existing test algorithm similar to the one used in a recent TCAM testing patent [16] was used as a basis for comparison. It proceeds in two steps: 1) Test the ability of every address to match. 2) Test each bit's ability to mismatch. Step 1 is accomplished as follows: Reset all words to zeros, and then search for all zeros to see if all addresses are returned as a match. Repeat this step using complementary values. Step 2 is accomplished as follows: Write 000…001 to all addresses, and then search for all zeros to ensure that all addresses are mismatches. Left shift the "1" bit (000…010) and write the pattern to all addresses and repeat the search. Repeat using complementary values. This algorithm individually tests each bit's ability to match and mismatch for both ones and zeros, and will have a total number of writes, searches, and address readouts as follows:

Existing Test Algorithm: $2n(l+1)$ writes, $2(l+1)$ searches, and $2n$ address readouts

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 every address is written to. Step 1 requires *n* writes, and Step 2 requires *nl* writes to write the corresponding test patterns. Repeating these steps once results in a total of 2*n*(*l* + 1) writes. Step 1 requires one search, and Step 2 requires *l* searches to test each bit. Repeating these steps once results in a total of 2(*l* + 1) searches. In the fault-free case, Step 1 requires *n* matching address readouts. Step 2 only results in matches in faulty cases. Thus there are a total of 2*n* address readouts. The resulting total complexity is:

Existing Algorithm Complexity (Total): $2[(n+1)(l+1)+n]$

It can be seen that the 2*nl* term dominates, and therefore the existing test algorithm's complexity is O(*nl*). For a 64k-word x 144-bit TCAM, this equals 19,005,440 writes, 290 searches, and 131,072 address readouts, for a total of 19,136,802 operations. Figure 4.14 shows the existing procedure along with its complexity. The total complexity assumes equal penalties for writes, searches, and address readouts in order to sum their complexities.

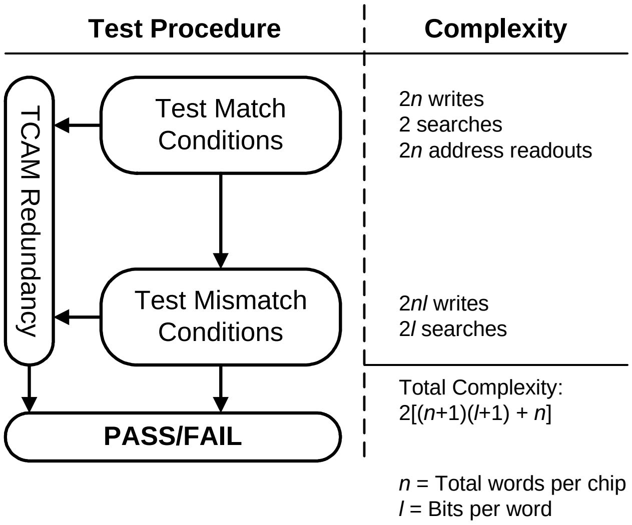


Figure 4.14 – Existing Test Procedure and Complexity

4.6.2 Comprehensive TCAM Test Strategy Complexity

The algorithm presented in this thesis has an average total number of writes, searches, and shifts as follows assuming S/R scan chains (i.e. one clock cycle to reset the scan chain to zero), no SON faults, and the ability to simultaneously shift a scan chain or search, and examine a resulting address (i.e., only one or the other will be counted during one operation):

MAE Test: n shifts

There are n shifts required to test every possible output address of the MAE, as shown in Figure 4.3. The MMR should never output more than one matching address, so this functional test is sufficient. If the MMR did output more than one matching address, an erroneous address would be encoded by the MAE and detected by the tester.

MMR Node Test: $\left(n\sum_{i=0}^{m-1}\frac{1}{p^i}\right)(p+1)$ shifts

where p = number of input signals into each MMR block, and m = number of levels of the MMR being tested. There are p + 1 test vectors needed for the MMR node test, as shown in Figure 4.6. For every test vector, the output must be serially shifted out. The number of outputs (total size of SC-b) is equal to n for the L1 MMRs, plus n/p for the L2 MMRs, plus n/p^2 for the L3 MMRs, etc. For example, for a 64k word TCAM with 16-bit MMRs and 4 levels of testing ($n = 2^{16}$, $p = 16$, $m = 4$), there would be 4,096 L1 MMRs, 256 L2 MMRs, 16 L3 MMRs, and 1 L4 MMR. Thus, the first factor in brackets represents the total length of SC-b for all levels of MMR being tested, and the second factor represents the number of test vectors.

MMR Full Tree Test: $n+1$ shifts

The test pattern in Figure 4.6 can be extended to all n inputs of the full MMR, resulting in $n + 1$ total patterns. There is no readout penalty since the output of the MAE is used to encode the MMR output.

TCAM Array Test (intra-cell): $2n$ writes, and $2n\left(\frac{l}{\log_2(n)}\right)$ searches

For the intra-cell test, the memory space is written to twice (assuming no SON faults) resulting in $2n$ writes. Each address in each logical column is searched twice, resulting in $2n$ searches per column, and $l/\log_2(n)$ columns. This explains why the number of searches is divided into two factors, one representing searchers per logical column, and the other representing the number of logical columns.

TCAM Array Test (inter-cell): $4n$ writes, $6l$ searches, and $4n$ address readouts

Initially, the n addresses are written to, and each bit is tested for SOP faults in both the one and zero cases, requiring $2l$ searches. Next, the same stored test data is used twice to search for SON faults, each time producing $n/2$ address readouts (in the fault-free case) for a total of n address readouts. This is repeated with inverted stored data, yielding $2n$ writes, $4l$ searches, and $2n$ address readouts. Next, the n addresses are re-written with all zeros, and each bit is tested for SOP faults in zero cases, requiring l searches. Next, the same stored test data is used once to search for SON faults, each time producing n address readouts (in the fault-free case). This is repeated with inverted stored and search data, yielding an additional $2n$ writes, $2l$ searches, and $2n$ address readouts. This is a total of $4n$ writes, $6l$ searches, and $4n$ address readouts.

Comprehensive Test Strategy Complexity (Total): $n\left(12+\frac{p(p+1)}{p-1}+\frac{2l}{\log_2(n)}\right)+6l+1$

The above equation takes the summation $\sum_{i=0}^{m-1}\frac{1}{p^i}$ to infinity to simplify the result as $\frac{p}{p-1}$, which is typically valid since the summation drops off so quickly. The minimum number of MMR levels being tested at once (during block-level testing) is two, so if the infinite summation was used in this case, it would also erroneously account for the 16 L3 MMRs, and the 1 L4 MMR used to implement a 64k-word TCAM. Higher-order terms result in non-integer solutions, and can be truncated. Thus, if precision is needed when calculating the complexity of testing a subset of the word space, the summation should be used instead of the infinite series.

For a 64k word by 144-bit TCAM with 16-bit MMRs nodes and 5 levels in the MMR tree being tested, this would result in a total of 1,319,459 shifts, 393,216 writes, 1,180,512 searches, and 262,144 address readouts. This is a total of 3,155,331 operations. Figure 4.15 shows both the proposed test procedure along with its complexity. The total complexity assumes equal penalties for shifts, writes, searches, and address readouts in order to sum their complexities.

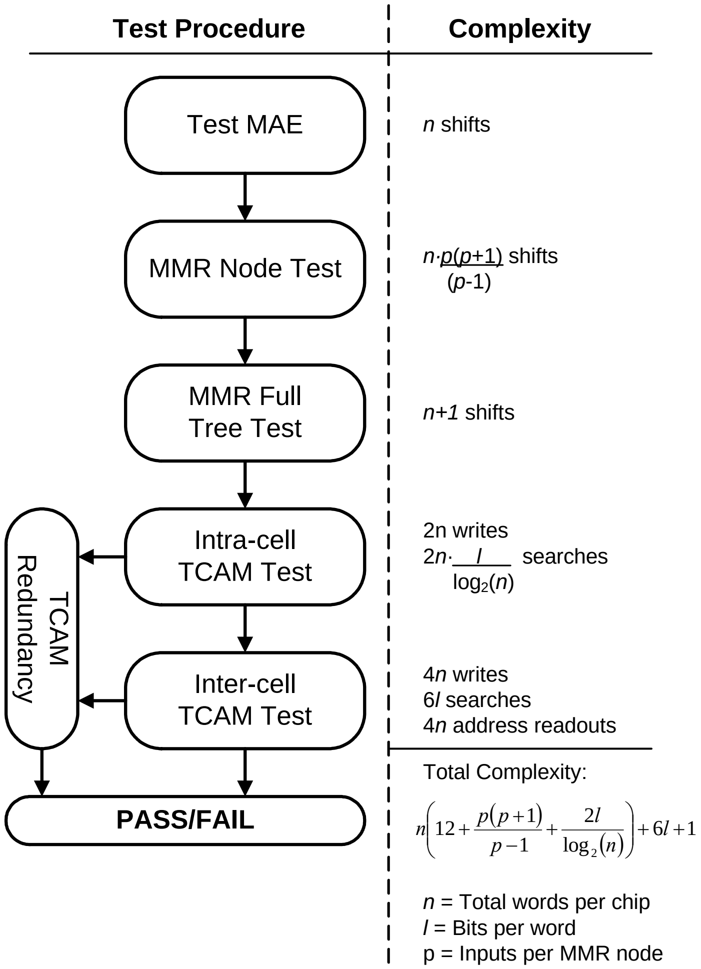


Figure 4.15 – Proposed TCAM Test Procedure and Complexity

The equations given above represent the fixed overhead of the algorithm with no SL or BL SON faults. The penalty for a BL SON fault is on average $\log_2(n)/2$ searches and one address readout (i.e. it takes an average of searching half the logical column width to find the SON fault, and the successful search results in one unnecessary address and the expected address). The penalty for a SL SON fault is $\log_2(l)$ searches and writes plus the initial $\log_2(n)$ searches needed to determine that it was not a BL SON fault. For the aforementioned 64k-word x 144-bit TCAM, this is only an average of 8 searches plus one extra address readout for BL SON faults, and 24 searches plus 8 writes for SL SON faults. Thus, SON faults cause a negligible increase in test complexity when using the stated detection algorithms.

4.6.3 Comparison of Test Algorithm Complexities

Assuming that shifts, writes, searches, and address readouts all take the same number of cycles, the proposed test algorithms achieve a 6x decrease in total test complexity, and at the same time a drastic increase in fault coverage when compared to the existing TCAM test algorithm. Furthermore, the existing algorithm is not designed to test the MAE and MMR, so based on TCAM array testing alone (writes and searches only) this represents a 10x decrease in complexity.

Figure 4.16 shows how the complexities, and consequently the test times, of the two TCAM test algorithms compare as the total number of words increases. It assumes $p = 16$, and $l = 144$. Though the new test flow is more intricate than simply executing an exhaustive algorithm, Figure 4.16 clearly shows the savings by using the new test flow. What is not depicted is the additional coverage of the new test flow by specifically testing the MAE and the MMR.

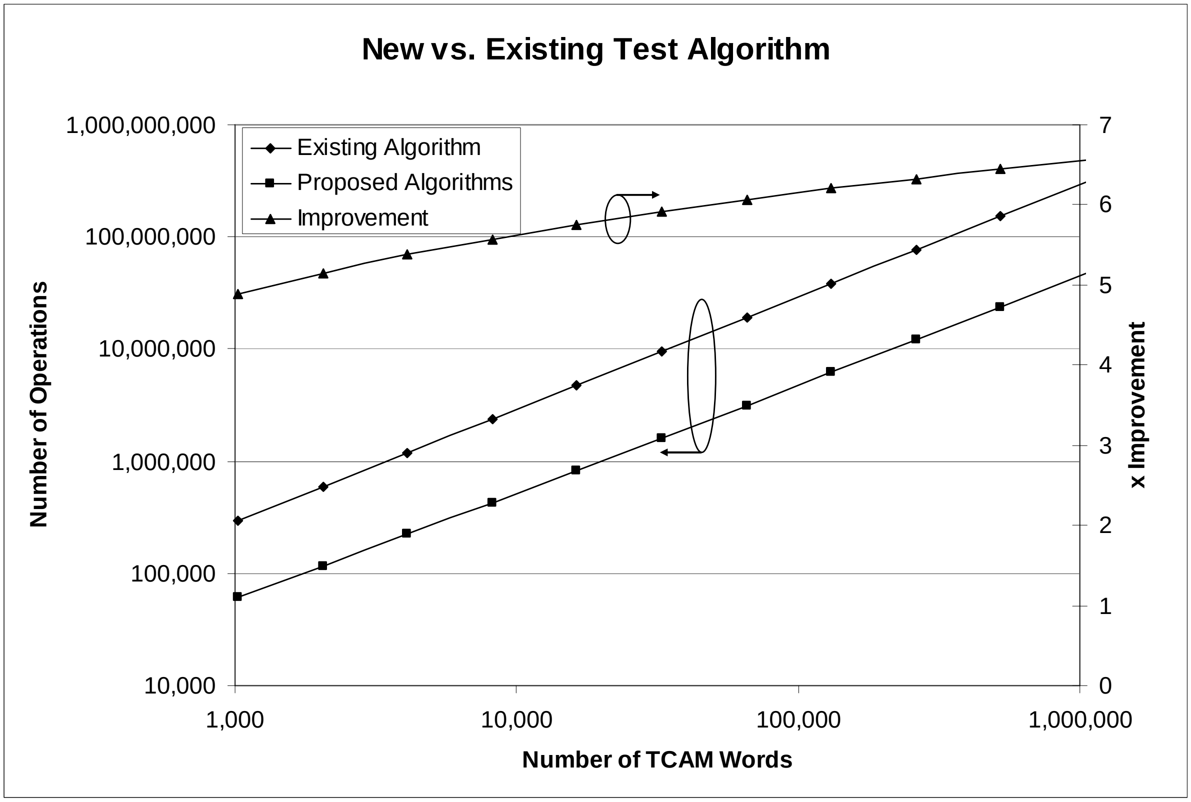


Figure 4.16 – Comparison of Proposed and Existing Test Algorithms

4.6.4 Impact of BIST on Test Complexities

If BIST is included in the design of a TCAM, and the BIST is implemented such that parallel block level testing is possible, then the overall complexity of testing can be reduced. If *n′* is the number of words per block, then any test that can be executed at the block level will have the *n* replaced with *n′* in its complexity equation.

For the existing test algorithm, the entire algorithm could be executed at the block level, resulting in a total complexity of $2[(n'+1)(l+1)+n']$. For blocks of 128 words x 144 bits, this results in a total of 37,666 operations. Comparing the equivalent part of the new test procedure, the TCAM array test algorithms, which can be executed at the block level, the resulting total complexity is $2n'\left(\frac{l}{\log_2(n')}\right)+8n'+6l$. For the same blocks of 128 words x 144 bits, this results in a total of 7,264 operations. The MMR node test can operate at the block level, reducing the scan-out time of test results along SC-b, but the MMR full tree and MAE tests must be executed at the chip level.

These dramatic decreases in test complexity through test parallelization and BIST lead to the conclusion that BIST with parallel block level testing is essential for large stand-alone TCAMs regardless of the algorithms used. That being said, the new test algorithms are still significantly faster than existing TCAM test algorithms.

Chapter 5
Conclusions

In this thesis, the test complexity of a dynamic TCAM was analyzed. A defect-oriented test suite was developed using realistic transistor level defects. The algorithms were validated using a high-level MATLAB model. The new tests, compared to existing methods, are 6x faster owing to reduced complexity. However, the new algorithms also test the MAE and MMR before testing the TCAM array. The following general conclusions summarize this research:

- The TCAM search path is tested from back to front (MAE, MMR, TCAM array) to ensure full operation of the individual components and the full operation of the system as a whole
- DfT components are required to increase the observability and controllability of the MMR and MAE
- The TCAM array test algorithms are executed along with established RAM tests, and its results are amenable to both row and column redundancy
- BIST enables time saving block level test parallelization and is recommended for any large stand-alone TCAM

With respect to future research directions:

- There may be overlap between the intra- and inter-cell test algorithms, and opportunities may exist to prune certain test vectors
- Specific BIST implementations of the new test algorithms could be described
- A solution to the scan-out bottleneck of the MMR node test could be explored
- The new algorithms could be executed in parallel with existing algorithms in a real manufacturing test environment to determine the true benefits of the new test algorithms

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] 2003 International Technology Roadmap for Semiconductors, *Test & Test Equipment* [Online], Available: http://public.itrs.net/Files/2003ITRS/Test2003.pdf

[6] F. Shafai, K. J. Schultz, G. F. R. Gibson, A. G. Bluschke, and D. E. Somppi, “Fully Parallel 30-MHz, 2.5-Mb CAM,” *IEEE Journal of Solid-State Circuits*, vol. 33, no. 11, November 1998.

[7] M. Sachdev, “Open defects in CMOS RAM address decoders,” *IEEE Design & Test of Computers*, vol. 14 , issue 2 , pp. 26-33, April-June 1997.

[8] I. Porneranz, S. M. Reddy, and Y. Zorian, “A test interface for built-in test of non-isolated scanned cores,” *Proceeding of the IEEE VLSI Test Symposium*, pp. 371-376, May 2003.

[9] L. van de Logt, F. van der Heyden, and T. Waayers, “An extension to JTAG for at-speed debug on a system,” *Proceedings of the IEEE International Test Conference*, pp. 123-130, October, 2003.

[10] MOSAID Technologies, “MOSAID Class-IC DC9288 Feature Sheet,” February 2003.

[11] 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.

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

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

[14] 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.

[15] 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.

[16] S. Gupta, and R. Gibson, “Methods and Circuitry for Built-In Self-Testing of Content Addressable Memories”, *U.S. Patent no. 6,609,222*, 2003.

[17] SiberCore Technologies, *Ultra-18M SCT1842 Product Brief* [Online], Available: http://www.sibercore.com/pdf/SCT1842_ProdBrief.pdf

[18] L. Shen, and B. F. Cockburn, “An optimal march test for locating faults in DRAMs,” *Records of the IEEE International Workshop on Memory Testing*, pp. 61-66, August 1993.

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

[20] 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.

[21] S. Kim, M. Kapur, M. Meghelli, A. Rylyakov, Y. Kwark, and D. Friedman, “45-Gb/s SiGe BiCMOS PRBS generator and PRBS checker [pseudorandom bit sequence],” *Proceedings of the IEEE Custom Integrated Circuits Conference*, pp. 313-316, September 2003.

[22] W. Maly, A. J. Strojwas, and S. W. Director, “VLSI Yield Prediction and Estimation: A Unified Framework,” *IEEE Transactions on Computer Aided Design*, vol. CAD-5, no. 1, pp. 114-130, January 1986.