A Formal Framework for Modeling and Testing Memories

by

Piotr Roald Sidorowicz

A thesis
presented to the University of Waterloo
in fulfilment of the
thesis requirement for the degree of
Doctor of Philosophy
in
Computer Science

Waterloo, Ontario, Canada, 2000

I hereby declare that I am the sole author of this thesis.

I authorize the University of Waterloo to lend this thesis to other institutions or individuals for the purpose of scholarly research.

I further authorize the University of Waterloo to reproduce this thesis by photocopying or by other means, in total or in part, at the request of other institutions or individuals for the purpose of scholarly research.

The University of Waterloo requires the signatures of all persons using or photocopying this thesis. Please sign below, and give address and date.

Abstract

Testing is essential to VLSI circuit production. In the case of memory circuits, the cost of testing often exceeds the cost of manufacture. Current memory testing methods rely on fault models that are inadequate to accurately represent potential defects that occur in modern, often specialized, memories.

We present a formal framework for modeling and testing memories. Simple fault models are created, based on potential circuit-level defects in a given memory. This framework is demonstrated using a content-addressable memory (CAM) as an example. CAMs are used in integrated circuits where searching is a key operation.

A CAM cell is analyzed at the transistor-network, event-sequence and finite-state machine levels. A fault model is defined; it comprises input stuck-at, transistor and bridging faults. We show that functional tests can reliably detect all input stuck-at faults, most transistor faults (including all stuck-open faults), and about 50% of bridging faults. The remaining faults are detectable by parametric tests. A test, of length $7n + 2l + 9$, that detects all the reliably testable faults in an n-word by l-bit CAM was designed. DFT suggestions that reduce the length of this test to $2l + 11$ are proposed. Two CAM tests, by Giles & Hunter and by Kornachuk et al., are evaluated with respect to the input stuck-at faults. It is shown that the former test fails to detect certain faults; it can be modified to achieve full coverage at the cost of increased length.

To demonstrate the general applicability of our framework, an input stuck-at fault model of a word-oriented, static random-access memory (SRAM) is also given. Several commonly known tests are evaluated; some fail to detect close to 50% of faults in this model.

Acknowledgements

I would like to thank my supervisor, Dr. Janusz (John) Brzozowski for his support and guidance. His commitment to his work and to his students, attention to detail and professional standards are without equal; they will always remain a source of inspiration for me. I am very grateful to him.

I am thankful to the other members of my defence committee, Drs. Helmut Jürgensen, Mike McCool, Manoj Sachdev, Johnny Wong, and Yervant Zorian, for the time spent on reading my thesis and for all the valuable comments and suggestions.

Sincere thanks go to Dr. Kenneth Schultz, formerly of Nortel Corporation's Memory Development Team, for providing extensive information regarding various defects and their effects on a CAM.

I would like to express my gratitude to the all the members of the Maveric Group, particularly to Drs. Robert Berks, Tracey Bogue, and Radu Negulescu for their insightful comments and suggestions.

Many day-to-day tasks would have become bureaucratic nightmares, if it were not for the wonderful support and assistance of the members of the departmental staff, particulrly of Debbie Mustin, Jane Prime, Wendy Rush and Ursula Thoene. Thank you for all your help.

I am grateful to my fiancée Ewa Madey for her patience, love, understanding, and for always being there for me.

Finally, I would like to acknowledge the support of the Natural Sciences and Engineering Research Council of Canada under grant No. OGP0000871.

Dedication

To my late grandparents, Irena and Lucjan, whose love, guidance and understanding have always made such a difference in my life. Wish you were here. Then again, you probably are...

Contents

List of Figures

List of Tables

Chapter 1

Introduction

"If you don't test it, it won't work! Guaranteed." [54]

This adage embodies the undisputed necessity of testing. Particularly, in the realm of VLSI circuits where millions of devices are located in an area of approximately 1 cm^2 and no fabrication method, no matter how meticulous, can guarantee perfect results, testing is an essential and integral part of the production process. There are numerous reasons for VLSI circuit failure. During chip fabrication, defective silicon substrate, improper doping, mask misalignments or imperfections, contamination with dust particles, or etching flaws may result in outright faulty devices, or devices which fail soon after being put into operation. Even correctly operating chips may, in time, fail due to electro-migration, oxidation, corrosion, or mechanical stress. Moreover, such aging processes can be accelerated by extreme environmental conditions. It is, therefore, necessary to test integrated circuits throughout their life-cycle, from fabrication to in-service maintenance.

Memory circuits constitute a large part of all integrated circuit production. High bit-densities of modern memory chips render them particularly prone to failure, thus magnifying the need for thorough testing methods that are efficient enough to be economically justifiable. Not surprisingly, a substantial research effort has been devoted to testing random-access memories (RAMs). The development and utilization of fault models for RAMs has led to qualitative comparisons among existing testing methods. It also facilitated the construction of more efficient testing methods and the establishment of lower bounds on the lengths of test sequences for various faults.

As technologies and cell designs change and as the circuits become smaller, it is no longer clear if fault models designed some ten years ago are still applicable to the memories of today. This is particularly true for special-purpose memories that incorporate additional, non-standard circuitry. Although new tests for these memories are being developed [21], the approach taken is still rather *ad-hoc*, and results in an ever increasing number of highly abstract fault models, completely detached from the design and the technology for which they were developed.

In this thesis we demonstrate the need for establishing simple formal fault models that are created for specific memory cell designs. As a representative of a special-purpose memory we chose a CAM, a word-oriented storage device that is being utilized increasingly often in digital designs for its parallel search abilities. Although CAMs have been known since the sixties, their higher complexity and cost effectively excluded them from use by the digital systems industry. By the same token, research devoted to testing CAMs has also been limited. Today, due to the never-ending quest for increased processing power and advancement in VLSI technology, CAMs are usually found embedded in various high performance application-specific integrated circuits (ASICs).

We have developed a cell-specific fault model based on a transistor circuit of a static CMOS CAM cell utilized by Nortel Corporation in their telecommunication ASICs. We have found that only a fraction of the faults that can occur in the CAM cell do, indeed, correspond to the well-established fault types such as *cell stuck-at* faults.

Our fault model provides a basis for comparisons among test methods currently used for testing CAMs in terms of coverage of faults that are specific to a particular CAM cell circuit design. It provides a precise mathematical foundation for determining shortest test sequences for various faults in the model. Short test sequences are particularly relevant to embedded memories, where testing is accomplished by built-in self test (BIST) schemes.

By applying our formal framework for fault modeling, test generation and evaluation to an SRAM cell we demonstrate that our approach can be easily applied to any type of memory.

The remainder of this thesis is organized as follows: General issues regarding testing are discussed in Chapter 2. Chapter 3 is devoted to common methodologies used for testing random access memories. Formal memory models are described in Section 4. CAM fundamentals are the topic of Section 5. A survey of existing CAM tests is given in Chapter 6. Section 7 presents the formal model of the three types of faults contained in our fault model. The development of an efficient CAM test is described in Chapter 8. An evaluation of some of the CAM tests of Chapter 6 is presented in Chapter 9. An application of the framework to an SRAM circuit is given in Chapter 10. Chapter 11 contains concluding remarks and future areas of investigation.

Chapter 2

Testing

The goal of testing is to distinguish good circuits from malfunctioning ones. A more detailed analysis leading to localization of the cause of a malfunction is known as diagnosis; hence, testing is only a part of a diagnostic process.

Digital electronic circuits may be tested in several ways. The analysis of various electrical properties, such as voltages, currents, noise, power dissipation, etc. is called *parametric* testing. Another approach to testing digital integrated circuits relies on a comparison of the behavior of the circuit to its functional specification without any regard for the structural implementation of the circuit. This type of analysis is referred to as *functional* testing. The study of how deviations in the structural implementation of a circuit may affect its logical functionality is called *logical* testing. Our efforts concentrate on the last approach.

2.1 Logical Testing

Any time a circuit's behavior deviates from that prescribed by its specification, we say that the circuit is malfunctioning, or that an *error* has occurred. Every error is the result of some physical *defect* or design miscalculation. A variety of different physical defects may occur in a circuit. Severed connections, shorts to ground or to supply voltage V_{dd}, any flaws caused by improper masking, or poor semiconductor doping during the manufacturing process, even an incorrect design, are all considered defects.

The large number of diverse physical defects renders any circuit analysis in terms of these defects impractical. It is much easier to analyze circuits in terms of *faults*. A fault is a term used to describe a group of physical defects which manifest themselves in the same way. For example, let us consider a group of defects that cause a node in a circuit to be permanently connected to a high voltage. Such a node can be referred to as being subject to a stuck-at-1 fault. Stuck-at-1 and stuck-at-0 faults are the best known and the most commonly analyzed faults. If under certain input conditions a fault causes an erroneous output value, the given fault is called *detectable*. Some faults are not detectable, which means that there are physical defects that do not cause a circuit to malfunction, but affect its performance. Other faults manifest themselves only if the circuit is tested at speeds approaching its design extrema; they are referred to as *dynamic* faults.

Logical testing of digital circuits is performed by applying a sequence of logical values to the circuit's input ports and observing the resulting output. If the output differs from the expected one, a fault is detected.

The nature of a fault determines the *fault model*; in case of stuck-at faults, the stuck-at fault model. In order to reflect faults of a different nature — for example, transistor stuck-on(open) or bridging faults — appropriate fault models have to be defined. This is particularly important in the case of CMOS circuits where certain defects may convert a combinational circuit into a sequential one or may cause the circuit to generate logically indeterminate output. Sequential behavior of a faulty combinational circuit can be detected by an application of an appropriate sequence of test patterns. Faults that cause logically indeterminate output, on the other hand, are often detectable by parametric tests.

One well known fault model is the *external stuck-at fault model.* It describes faults that may cause inputs as well as outputs of a gate to be permanently set to logic 0 or logic 1. The definition of this fault model has facilitated reasoning about forks as separate entities within the circuit, with their own respective faults [25]. Figure 2.1 depicts locations of faults for the external stuck-at fault model.

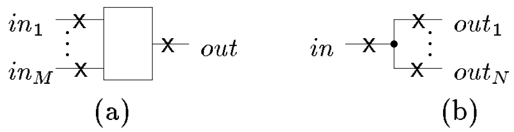


Figure 2.1: External stuck-at fault model for: (a) gate, (b) fork.

A simpler, *output stuck-at fault model* considers only those faults that occur on outputs of gates, thus effectively, on inputs of forks. Locations of faults for the output stuck-at fault model are presented in Fig. 2.2.

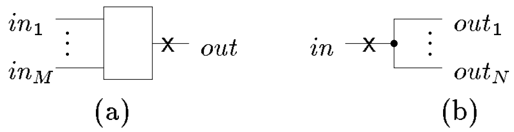


Figure 2.2: Output stuck-at fault model for: (a) gate, (b) fork.

It is clear that digital circuits need to be tested in terms of an appropriate fault model. The fact that not all faults (for a given fault model) are detectable in a particular circuit brings about the issue of *testability.* A circuit is completely testable if every fault in a given fault model can be detected. Detection of a fault requires that it be *activated* and the resulting error *propagated.* Consider the circuit in Fig. 2.3. The fault *Z= Sa-1* is said

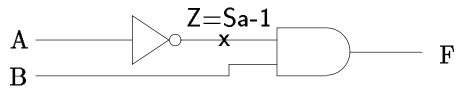


Figure 2.3: Activation and propagation of faults.

to be activated when input A of the circuit is set to logic 1. Consequently, node Z is set to logic 0 in a fault-free circuit. This process is often referred to as *fault sensitization.* In effect an erroneous input value is presented on the upper input to the AND gate. This error will be propagated to the output F when input B is set to logic 1. Therefore, fault $Z=$ *Sa-1* is detectable because, for the input vector $AB = 11$, a fault-free circuit would produce a 0 on the output, whereas the faulty circuit yields a 1. In this case it can be said that the node Z can be *controlled* and *observed.* Clearly, testability of a circuit is determined by the observability and controllability of all its internal nodes. A comprehensive study of these issues can be found in [1].

Let us now look at the testability of circuits under the stuck-at fault model.

Digital circuits are classified as either combinational or sequential. The issue of testing combinational circuits is a well studied one [1, 17]. The simplest circuits to test are the *fan-out free* circuits, such as the one presented in Fig. 2.3. The reason for this simplicity can be explained by following Theorem [8]:

> "In fanout-free circuits (i.e., circuits where each primary input and each gate output are inputs to at most one gate), there exists a single stuck fault test set of minimal cardinality which detects all multiple stuck faults."

Unfortunately, most combinational circuits contain *reconvergent fan-out*, which means that an arbitrary fault could be activated, and the resulting error propagated through multiple paths. It may also be the case that coherent input values can be determined for detection of a particular fault only for a subset of possible paths. The selection of appropriate paths is done algorithmically. Many such algorithms have been developed for testing for stuck-at faults. These algorithms include the D-algorithm, Podem, Fan, etc.

Testing of sequential circuits is a different matter. Consider the circuit of Fig. 2.4. The next state of this sequential circuit depends on the present input values and on the

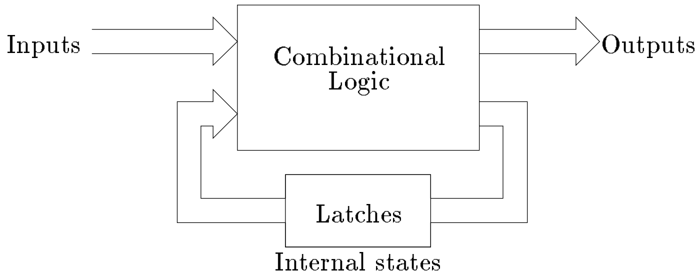


Figure 2.4: Sequential circuit overview.

current internal state of the circuit. A common way of testing a sequential circuit is to visualize it as testing a series of combinational circuits which in addition to inputs and outputs from the sequential circuit, have additional inputs denoting the current state, and additional outputs denoting the next state of the sequential circuit. Unfortunately, the internal state is often not observable and its controllability is limited. In order to determine a sequential circuit's adherence to its specification, a sequence of input values has to be applied from a known initial state. This implies that the circuit has to be somehow initialized. Unless the circuit has a resetting capability, an initializing sequence of input values has to be applied beforehand. This sequence is also known as a *homing* sequence. It has been shown that input sequences for determining the internal state of a sequential circuit are in the worst case exponential in length [29]. In general, exhaustive testing of sequential circuits is an NP-hard problem [17].

As stated earlier, the testability of a circuit is determined by its controllability and observability. By introducing circuit design modifications that improve either of these two factors, the complexity of necessary tests can be greatly reduced. To date, a substantial research effort has been dedicated to design for testability (DFT). *Scan test* design is a popular method of DFT.

2.2 Scan Test

Scan design is a method of increasing sequential circuit testability. To illustrate it, we present the sequential circuit of Fig. 2.5. It consists of five registers interleaved with some combinational logic. The only inputs that are directly controllable are those attached to

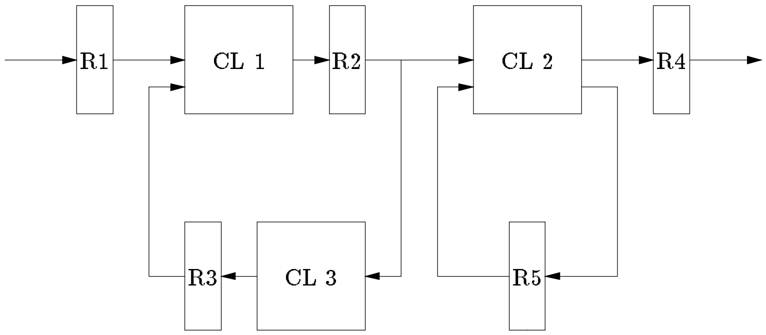


Figure 2.5: Arbitrary sequential circuit.

register $R1$, and the only outputs that are directly observable are those from register $R4$. The remaining three registers are internal and, therefore, practically inaccessible.

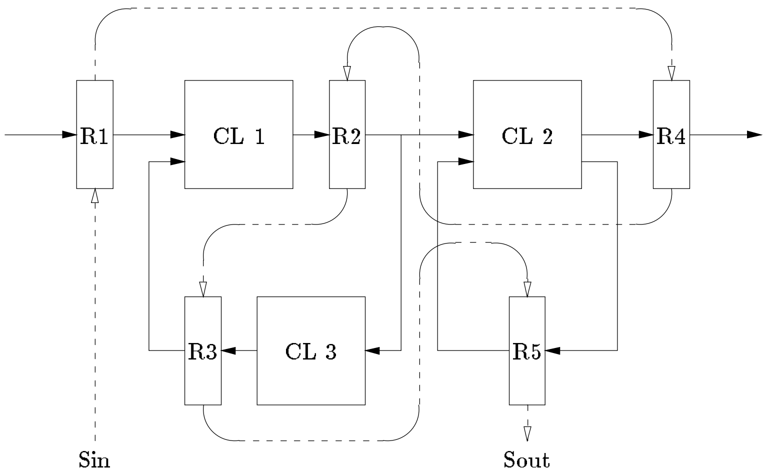


Figure 2.6: Scan Test design.

Now, suppose that all the registers are *scan registers* (which are similar to shift registers) and are connected together with a serial line to form a single scan path, as depicted in Fig. 2.6. Through this path arbitrary values can be scanned in and out of every register. In this manner all the registers are made fully controllable and also fully observable. Since every register can be now loaded with useful test vectors, the hard problem of testing sequential circuits has been reduced to that of testing combinational circuits.

In the above example every register is made scannable, which constitutes a *full scan test* design. Circuits designed in this manner are fully testable. This ability, however, comes at a price. Scan registers are slower and more complex than their non-shiftable counterparts. Additional serial and control lines are also required. Consequently, fully scannable circuits are larger, more complex, slower, and use more energy than their non-scannable counterparts. Furthermore, full scan designs entail extensive scan paths resulting in prohibitively long scan-in/scan-out times.

Limiting the number of registers included in the scan path in order to reduce overhead constitutes a *partial scan test* design. These registers must be selected in a way that does not impede the circuit's testability. Significant amount of research has been devoted to establishing optimal partial scan tests [6, 12, 23, 27, 51].

Scan path techniques are not limited to intra-chip testing. At the system level, test sequences can be provided to individual chips on a circuit board via a serial path line. This method, called *boundary scan*, has proven so useful that it has been standardized by the IEEE[1] [54].

Scan path techniques, although very useful in improving the testability of a sequential circuit, have the disadvantage of being serial in nature. This imposes a practical upper bound on the length of the test vectors. Hence the goal is to obtain maximal fault coverage with shortest possible tests.

[1]IEEE 1149 Boundary Scan architecture.

2.3 Summary

Fundamental aspects of testing digital circuits have been described using the most common fault model, the stuck-at fault model, as an example. This fault model is insufficient to describe all possible faulty behaviors of a digital circuit, particularly one implemented in CMOS technology. Other fault models are required and appropriate tests have to be devised. Moreover, due to the existence of non-logical faults, logic tests have to be complemented with parametric tests.

Logic testing, in itself is a complex issue, particularly when applied to sequential circuits. Exhaustive testing is an NP-hard problem in general, so fault models have to be tailored to the circuits under test in order to be accurate, and for the respective tests to be short enough to be useful.

Although circuits should be designed to include features that improve testability, such as scan paths (DFT), test lengths should be minimal while providing optimal fault coverage in the fault model.

Chapter 3

Memory Testing

Memory-based products make up a significant percentage of the total production of integrated circuits. As technologies change and bit densities continue to increase, new causes of failure emerge; defects that were of no consequence in earlier generations have to be dealt with now. Consequently, testing methods have to evolve continuously in order to maintain the expected levels of quality control with efficiency.

Testing of random-access memories (RAMs) has been the topic of extensive research. The content of this chapter is a summary of fundamental aspects of functional RAM testing, based on the comprehensive studies on testing RAMs presented in [20, 42].

3.1 RAM Test Classification

RAMs are large sequential circuits used for storing binary information. Their large state space and regular structure has led to the development of memory-specific testing methodologies. Testing methodologies can be divided into two categories: *traditional* and *modern*; the latter is also known as *reduced*.

Traditional tests such as 'Zero-One', 'Checkerboard', 'GALPAT', 'Walking 1/0', 'Sliding Diagonal' and 'Butterfly' were not designed on the basis of any fault model. Fault coverage of these tests could not be determined and qualitative comparisons between these tests were difficult. Moreover, these tests were often $O(n^2)$ in length, where n is the number of bits stored in the chip. Since modern memory chip capacities reach the $1Gb$ range, any tests of such length are unacceptable.

Modern functional memory testing techniques define their coverage in terms of faults, such as: (cell) stuck-at, toggling, coupling and pattern-sensitivity. Examples of modern tests are: 'MATS', 'MATS+', 'MATS++', 'March C-', etc. These tests are also known as *march tests.* They derive their name from the fact that memory cells are analyzed one by one in ascending or descending order. The sequence of operations that are performed on each consecutive cell is referred to as the *march element.* March tests can be easily generated and thus are often used in built-in self-test designs.

Van de Goor presents a study of several march tests in terms of their fault coverage and complexity, followed by a qualitative comparison to traditional ones [20]. These results are summarized in Table 3.1. This table contains the fault coverages of several tests. The listed faults include both those of the memory array (i.e., cell stuck-at faults, transition faults and coupling faults) and those of the peripheral circuitry (i.e., address decoder faults, refresh faults, sense amplifier recovery faults and write recovery faults). The latter three faults are dynamic, as they occur when the memory is operated at speeds approaching its design extrema. For instance, a slow sense amplifier may fail to read a 0 after reading several consecutive 1s, or a slow address decoder may fail to access a correct memory location in time for a subsequent operation. It should be noted that traditional tests not only detect faults, but also locate them within the memory array; however, this diagnostic ability comes at a substantial increase in test time and is usually not required, since a faulty chip will often be discarded.

Two types of storage mechanisms are used in current RAMs. When data is stored

Table 3.1: Comparison of memory tests for n-bit RAMs [20].

Algorithm	Fault coverage					Test time
	AF	SAF	TF	CF	Others	
Zero-One	-	L	-	-		$O(n)$
Checkerboard	-	L	-	-	Refresh	$O(n)$
Walking 1/0	L	L	L	L	Sense amplif. rec.	$O(n^2)$
GALPAT	L	L	L	L	Write recovery	$O(n^2)$
GALROW	LS	L	L	L	Write recovery	$O(n \cdot \sqrt{n})$
GALCOL	LS	L	L	L	Write recovery	$O(n \cdot \sqrt{n})$
Sliding Diag.	LS	L	L	-		$O(n \cdot \log_2(n))$
Butterfly	LS	L	L	-		$O(n \cdot \log_2(n))$
MATS	DS	D	-	-		$O(n)$
MATS+	D	D	-	-		$O(n)$
Marching 1/0	D	D	D	-		$O(n)$
MATS++	D	D	D	D		$O(n)$
March X	D	D	D	D	Unlinked CFins	$O(n)$
March C-	D	D	D	D	Unlinked CFins	$O(n)$
March A	D	D	D	D	Unlinked CFs	$O(n)$
March Y	D	D	D	D	Linked TFs	$O(n)$
March B	D	D	D	D	Linked CFs	$O(n)$

AF = Address decoder Fault L = Locate
SAF = Stuck-at Fault D = Detect
TF = Transition Fault LS = Locate Some
CF(in) = (inversion) Coupling Fault DS = Detect Some

as a charge on a capacitor the RAM is called a *dynamic* RAM (DRAM). This charge dissipates in time due to leakage currents, and has to be dynamically refreshed; hence the name. In fact DRAMs are largely analog devices. On the other hand, data can also be stored in a latch - a digital device. In this case, as long as power is supplied to the chip, the data is retained indefinitely. For this reason such a RAM is called a *static* RAM (SRAM).

Defects in memory circuits can be partitioned into two categories: global and local. Global defects are often related to the manufacturing process. They include incorrect thickness of polysilicon or gate oxide that may result from variations of oven temperature, and affect a large area of a wafer. These defects are readily detectable by various parametric tests. Local or *spot* defects, such as dust particles on the chip or gate oxide pinholes, manifest themselves mostly as *functional* faults. They are interpreted at the layout level as broken wires, shorts between wires, missing contacts, extra contacts and newly created transistors. Spot defects are mapped into the following functional classes:

- A stuck-at fault in a cell,
- A stuck-open fault in a cell, i.e. the cell cannot be accessed by read or write operations,
- A multiple cell access fault,
- A data retention fault, where a cell changes its state spontaneously,
- A coupling fault,
  - state coupling (SRAMs only), where the coupled cell is driven to some state only when the coupling cell is in a particular state,
  - idempotent coupling, where the coupled cell is driven to some state due to a change of state in a coupling cell,
  - dynamic coupling (DRAMs only), where the coupled cell is driven to some state when the coupling cell is being accessed.
- A transition fault (SRAMs only), where a cell's state can change only in one direction (e.g., from 0 to 1, but not from 1 to 0),
- A pattern-sensitivity fault (DRAMs only), where a cell's state is influenced by the state, or change of state of the neighboring cells. The cell's neighborhood may be defined by physical adjacency, a common row or column.

It is clear that due to the fundamental differences in the data storage mechanism, different fault models apply to DRAMs and SRAMs. A comprehensive survey of defect-based fault models for DRAMs and SRAMs is given in [5, 42].

Experimental studies have indicated that traditional and pattern-sensitivity fault tests are ill-suited for testing SRAMs, as the fault coverage of these tests falls in the $50\% - 70\%$ range [20]. An inductive fault analysis method, described in [20, 43], revealed that stuck-at faults constitute about 50% of faults occurring in SRAMs and data retention faults constitute up to 18%.

On the basis of these observations special SRAM tests, such as *IFA-9* and *IFA-13*, have been designed [20]. The IFA-9 test is an extension of the March C- test, allowing it to detect data retention faults. IFA-13 (a further extension of IFA-9) has been designed to deal with the sequential behavior of sense amplifiers in the presence of stuck-open faults. These tests are more efficient and yield higher fault coverage than general functional tests that detect coupling faults, such as MARCH A, B, or C.

3.2 Design-Oriented Fault Modeling

Defect-oriented fault modeling and inductive fault analysis techniques allow for the generation of tests that are more efficient than traditional tests. However, with the development of special-purpose memories, such as multi-port-RAMs, CAMs and FLASH memories, the number of fault models is growing at an alarming rate [4, 20, 21, 33], making them increasingly difficult to track. Unfortunately, many fault models also fail to specify the design details and the underlying technology (Bipolar, NMOS, CMOS, BiCMOS) of the circuit for which they were designed. As a result, tests designed to detect faults for a given fault model may inadvertently be used to test functionally similar memories for which that fault model may not apply. This confusing situation often forces industrial test engineers to use the old, well known fault models and tests, regardless of how inefficient or unreliable they may be.

In order to remedy this situation, we postulate the following guidelines for *design-*

oriented fault modeling:

- The circuit and the underlying technology should be specified to avoid confusion.
- Faults should be labeled in terms of defective circuit components, not in terms of functional behavior (e.g. *transistor stuck-on fault*, not *transition fault*).
- Particular attention should be paid to application-specific aspects of the design.

These guidelines are in agreement with the concept of logical testing, described in the previous chapter. However, in this thesis we will show that by following these guidelines simpler and more accurate fault models and logical tests can be developed.

3.3 Summary

An overview of memory testing topics has been presented. Traditional and modern tests have been described and compared in terms of their lengths and fault coverage. Faults indigenous to static and dynamic memories have been described. This characterization is a result of defect-based fault modeling techniques. We have also listed tests designed exclusively for SRAMs. These tests are more efficient and yield higher fault coverage than general functional tests. A design-oriented approach to fault modeling has been proposed. Its purpose is to categorize faults in terms of defects in the cell circuit and not in the cell's functional behavior. This approach should result in more accurate fault models that take into account the cell design, implementation technology, and design-specific functionality of a memory.

Chapter 4

Formal Models

In order to reason about memory faults a proper formalism is required. We adopt a formalism developed by Brzozowski and Jürgensen [11] for sequential circuit testing and diagnosis.

Finite-state automata are used to describe possible behaviors of a sequential circuit. The correct behavior of the circuit is modeled by a Mealy automaton $A^0 = (Q^0, X, Y^0, \delta^0, \lambda^0)$ called the *good machine type*. Here, Q^0 is the set of states, X is the input alphabet, Y^0 is the output alphabet, $\delta^0 : Q^0 \times X \to Q^0$ is the transition function, and $\lambda^0 : Q^0 \times X \to Y^0$ is the output function. A good machine type initialized to some state $q^0 \in Q^0$, is referred to as a *good machine* (A^0, q^0). Similarly, any incorrect behavior, referred to as a *fault type* is also a Mealy automaton $A^i = (Q^i, X, Y^i, \delta^i, \lambda^i)$. Fault types have the same input alphabet as A^0, but differ from it in the set of states, the output alphabet, transition and output functions. By the same token, a fault type initialized to some state is called a *fault*. The good machine type A^0, together with a finite family of fault types $A^1, \ldots, A^m$ and a set of *potential initial states* $P^i \subseteq Q^i$ for $i = 0, \ldots, m$ constitute a *fault model* for A^0 denoted as $\mathcal{F}_{A^0}$.

Since random-access memories are in fact sequential circuits (i.e., the output depends on the input and the internal state of the circuit), we shall use a sample RAM to illustrate a summary of the above formalism.

4.1 Memory Models

We define a model of a bit-addressable RAM of size n. A *fault-free n-cell RAM type* is a Mealy automaton [11]

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

where $Q = \{0,1\}^n$, $X = \bigcup_{i=1}^{n} X_i$ with $X_i = \{r^i, w_0^i, w_1^i\}$, $Y = \{0, 1, \$\}$, and the transition function δ and the output function λ are defined by

$$\begin{aligned}
\delta((q_1, \ldots, q_i, \ldots, q_n), r^i) &= (q_1, \ldots, q_i, \ldots, q_n), \\
\delta((q_1, \ldots, q_i, \ldots, q_n), w_0^i) &= (q_1, \ldots, 0, \ldots, q_n), \\
\delta((q_1, \ldots, q_i, \ldots, q_n), w_1^i) &= (q_1, \ldots, 1, \ldots, q_n), \\
\lambda((q_1, \ldots, q_i, \ldots, q_n), r^i) &= q_i, \\
\lambda((q_1, \ldots, q_i, \ldots, q_n), w_0^i) &= \$, \\
\lambda((q_1, \ldots, q_i, \ldots, q_n), w_1^i) &= \$.
\end{aligned}$$

Inputs r^i, w_0^i, w_1^i denote a 'read', 'write 0' and 'write 1' operations on cell i, respectively. Output \$ is merely a formal symbol denoting lack of output during 'write' operations.

Similarly, a model for an n-word by l-bit memory has been developed [15]. It is specified as an n-tuple automaton

$$M_{n,l} = (Q_{n,l}, X_{n,l}, Y_l, \delta_{n,l}, \lambda_{n,l}),$$

where

$$Q_{n,l} = \{0, 1, \ldots, 2^l - 1\}^n,$$

$$X_{n,l} = \{r^1, w_0^1, w_1^1, \ldots, w_{2^l-1}^1, \ldots, r^n, w_0^n, w_1^n, \ldots, w_{2^l-1}^n\},$$

and

$$Y_l = \{0, 1, \ldots, 2^l - 1, \$\}.$$

The transition function $\delta_{n,l}$ and the output function $\lambda_{n,l}$ are defined by

$$\delta_{n,l}((q^1, \ldots, q^i, \ldots, q^n), x) = \begin{cases} (q^1, \ldots, q^i, \ldots, q^n), & \text{if } x = r^i, \\ (q^1, \ldots, 0, \ldots, q^n), & \text{if } x = w_0^i, \\ (q^1, \ldots, 1, \ldots, q^n), & \text{if } x = w_1^i, \\ (q^1, \ldots, 2, \ldots, q^n), & \text{if } x = w_2^i, \\ (q^1, \ldots, 3, \ldots, q^n), & \text{if } x = w_3^i, \\ \vdots & \vdots \\ (q^1, \ldots, 2^l - 1, \ldots, q^n), & \text{if } x = w_{2^l-1}^i, \end{cases}$$

and

$$\lambda_{n,l}((q^1, \ldots, q^i, \ldots, q^n), x) = \begin{cases} q^i, & \text{if } x = r^i, \\ \$, & \text{otherwise.} \end{cases}$$

Inputs r^i, and w_j^i denote a 'read' and 'write j' operations on word i, respectively, where $j \in \{0, 1, 2, 3, \ldots, 2^l - 1\}$. Symbol \$ is used as before to denote lack of output.

4.2 Classic Fault Models

Classic memory faults, described in the previous chapter, can be formalized as follows [11]:

- A *stuck-at-a fault type* is denoted as:

$$M^{i=a} = (Q^{i=a}, X, Y, \delta^{i=a}, \lambda^{i=a}),$$

where the set of states is

$$Q^{i=a} = \{q \mid q = (q_1, \ldots, q_{i-1}, a, q_{i+1}, \ldots, q_n)\},$$

the input alphabet X and the output alphabet Y remain unchanged, and the transition function $\delta^{i=a}$ is a restriction of δ to $Q^{i=a} \times X$, except for the transitions under input $w^i_{\bar{a}}$ where $\bar{a}$ is the complement of a. In this case the next state function takes the form:

$$\delta^{i=a}((q_1, \ldots, q_{i-1}, a, q_{i+1}, \ldots, q_n), w^i_{\bar{a}}) = (q_1, \ldots, q_{i-1}, a, q_{i+1}, \ldots, q_n).$$

- A *transition fault type* is denoted as:

$$M^{i\neg a} = (Q, X, Y, \delta^{i\neg a}, \lambda).$$

In this Mealy automaton $\delta^{i\neg a}$ is identical to δ with the exception of the transition resulting from the input w^i_a, when cell i is in the state $\bar{a}$:

$$\delta^{i=a}((q_1, \ldots, q_{i-1}, \bar{a}, q_{i+1}, \ldots, q_n), w^i_a) = (q_1, \ldots, q_{i-1}, \bar{a}, q_{i+1}, \ldots, q_n).$$

- A *coupling fault type* involves two separate cells i and j, respectively. Cell i is referred to as the *coupling* cell, and cell j is the *coupled* cell. This fault is also defined as a Mealy automaton:

$$M^{ia\Rightarrow jb} = (Q, X, Y, \delta^{ia\Rightarrow jb}, \lambda),$$

where $\delta^{ia\Rightarrow jb}$ differs from δ in the transition resulting from the input w_a^i, when cell i is in the state $\bar{a}$ and cell j is in the state $\bar{b}$:

$$\begin{aligned} &\delta^{i=a}((q_1, \ldots, q_{i-1}, \bar{a}, q_{i+1}, \ldots, q_{j-1}, \bar{b}, q_{j+1}, \ldots, q_n), w_a^i) \\ &= \quad (q_1, \ldots, q_{i-1}, a, q_{i+1}, \ldots, q_{j-1}, b, q_{j+1}, \ldots, q_n). \end{aligned}$$

4.3 Observer

The goal of testing is to determine whether a circuit A' under test can be classified as a good machine type A^0 or some other fault type A^i ($1 \leq i \leq m$), on the basis of its input and output sequences. Initially A' is in some unknown state p. This is denoted by a machine (A', p). One step in the process of this classification consists of applying a test input to A', obtaining its response, and then comparing both input and output to possible behaviors of A^0 through A^m. This comparison will divide the set of machine types into two sets: one with outputs identical to that of A', and one with outputs different from that of A'. Machine types belonging to the latter set may be discarded as A' does not belong to them. Successive and successful repetition of this step will result in the first set to become a singleton ($A' = A^i$), or an empty set, which means that A' belongs to some other, unspecified fault type. This process of fault detection can be accomplished by means of a *(deterministic) observer*, which is an initialized deterministic semi-automaton $\Delta = \Delta(A^0, \mathcal{F}) = (D, X \times Y, \delta, d_0)$, defined as follows [10, 11].

For $i = 0, 1, \ldots, m$, let $A^i = (Q^i, X, Y^i, \delta^i, \lambda^i)$ and let $\widetilde{Q}^i = Q^i \cup \{\omega\}$ where ω is a new state symbol denoting the discarded machine types. Furthermore, for $q \in Q^i$ and $(x, y) \in X \times Y^i$, let the transition function be defined as

$$\tilde{\delta}^i(q,(x,y)) = \begin{cases} \delta^i(q,x), & \text{if } \lambda^i(q,x) = y, \\ \omega, & \text{if } \lambda^i(q,x) \neq y. \end{cases}$$

and let $\tilde{\delta}^i(\omega,(x,y)) = \omega$. Each state of the observer's state set $d \in D$ is a tuple with components d_q^i for $i = 0, 1, \ldots, m$ and $q \in P^i$ (set of potential initial states). Therefore the set D is defined as

$$D = \{d \mid d_q^i \in \widetilde{Q}^i,\ i = 0, 1, \ldots, m\ ,\ q \in P^i\}.$$

The observer's initial state d_0 is denoted by $[d_0]_q^i = q$ which includes all possible initial states of all fault models and the good machine. Its transition function $\delta(d,(x,y))$ equals e if and only if each component of e is $e_q^i = \tilde{\delta}^i(d_q^i,(x,y))$ for all i and q.

The maximum amount of information about an arbitrary sequential circuit A' obtainable from a test is: the initial state q, the current state q', and the value of the index i indicating the machine type. All possible outcomes of such identification can be described by the set

$$K = \{(i, q, q') \mid i = 0, 1, \ldots, m,\ q \in P^i,\ q' \in Q^i\}.$$

A complete deterministic observer can easily become unmanageably large as demonstrated in [10]. However, since we are only interested in establishing whether A' is a good machine or not, the observer can be substantially reduced. This is accomplished by extracting, out of the set of observer's states D, a partition B_F where the affiliation to either the good machine type or to the faulty machine types, but not both, has been established.

The partition $B_F = (B_0,\ B_{\neq 0})$ called a *fault partition* consists of two disjoint sets of states $B_0 = \{(0, q, q') \mid q \in P^0,\ q' \in Q^0\}$ and $B_{\neq 0} = \{(i, q, q') \mid i = 1, \ldots, m,\ q \in P^i,\ q' \in Q^i\}$ which uniquely determine whether the circuit under test is modeled by the good machine type or not. In other words, these states are *B_F-decided.*

The fault partition B_F is one of a number of other useful partitions, referred to in general as *B-partitions.* An input word which results in a transition to a B-partition, consisting of sets of B-decided states, is said to *B-diagnose* the circuit A' [10, 11].

4.4 Diagnosing Languages

An observer Δ can be modified in the following manner:

1. The output part Y from the input alphabet $X \times Y$ is removed, thereby introducing nondeterminism.

2. A set F of B-decided states is introduced as a set of final or accepting states.

As a result a *nondeterministic B-acceptor* $\overline{\Delta}\,(A_0, \mathcal{F}_0, B) = \left(D, X, \overline{\delta}, \{d_0\}, F\right)$ is obtained, where

$$\overline{\delta}(d, x) = \{d' \mid \exists y \in Y \ :\ \delta(d(x, y)) = d'\}.$$

Any word w is said to *B-diagnose* if and only if it always results in $\overline{\Delta}$ ending up in an accepting state. The set of all B-diagnosing words for the fault model $\mathcal{F}$ constitute a regular language $L_B(\mathcal{F})$ [11]. This observation has led to several useful properties:

- If $L_B(\mathcal{F}_1) \subseteq L_B(\mathcal{F}_2)$ then a fault model $\mathcal{F}_1$ is said to *B-cover* a fault model $\mathcal{F}_2$ for a diagnosis goal B. This is particularly useful in determining whether tests for one type of fault are applicable to testing for another type of fault, i.e. establishing the *fault coverage* of a particular test [55].

- Given several sets of B-diagnosing words for different fault models that constitute a family $\mathbf{F}$, a single set of B-diagnosing words can be obtained for the entire set of these fault models by the following formula [55]:

$$L_B\left(\bigcup_{\mathcal{F}\in\mathbf{F}} \mathcal{F}\right) = \bigcap_{\mathcal{F}\in\mathbf{F}} L_B(\mathcal{F}).$$

4.5 Lower Bounds

One of the early papers on lower bounds for the detection of coupling faults in RAMs was presented by Brzozowski and Cockburn [9]. Due to the recent rapid increase in memory sizes, efficient detection of faults is the subject of extensive research. Cockburn and Brzozowski [14] have presented a summary of known lower bounds and test sequence lengths for detecting various faults in RAMs. These bounds have been established using language theoretic tools and are shown in Table 4.1.

Table 4.1: Lower bounds and known test lengths for faults in n-bit RAMs.

Fault Model	Best Lower Bound	Best Known Test Length	Comments
general toggling	$2n^2 + n$	$2n^2 + n$	optimal for $n \geq 2$.
2-limited toggling	$n\lfloor\log_2(n-1)\rfloor + 5n$	$4n\lceil\log_2 n\rceil + n$	optimal for $n = 2$; shortest known for $n \geq 6$.
single toggling	$5n - 2$	$5n - 2$	optimal for $n \geq 2$.
general coupling	16, for $n = 2$; $2n^2 + 3n$, for $n > 2$	$2n^2 + 4n$	optimal for $n = 2$: conjectured optimal for $n \geq 3$.
4-limited coupling	$n\lfloor\log_2(n-1)\rfloor + 7n$	$4n\lceil\log_2 n\rceil + 17n$	shortest known for $n \geq 15$.
3-limited coupling	$9n - 2$	$14n - 12$	optimal for $n = 2$; shortest known for $n \geq 7$.
2-limited coupling	$9n - 2$	$12n - 8$	optimal for $n = 2$; conjectured optimal for $n \geq 3$.
single coupling	$9n - 2$	$10n - 4$	optimal for $n = 2$; conjectured optimal for $n \geq 3$.
toggle-free coupling	$9n - 2$	$15n - 14$	optimal for $n = 2$; shortest known for $n \geq 4$.

4.6 Summary

We have presented an overview of a formalism developed by Brzozowski and Jürgensen for sequential circuit testing and diagnosis [11]. We utilize this formalism in our behavioral analysis of memory cells. It provides the means to distinguish faulty behaviors from the fault free ones and to derive shortest input sequences with which that distinction can be made.

Chapter 5

CAM Fundamentals

In a typical random-access memory (RAM) stored data is accessed by specifying the *address* of the location in memory where data resides. On the other hand, a content-addressable memory (CAM), as the name suggests, allows access to stored data on the basis of a simultaneous comparison of the content of all memory locations with a particular datum, called the *search key.* This ability lends itself naturally to applications where simultaneous comparison is a principal operation. Chisvin and Duckworth [13] provide an extensive list of such applications: "file maintenance, pattern recognition, symbolic representation of information, data retrieval, parallel arithmetic algorithms, data correlation, speech recognition, radar analysis, connectivity testing, spelling checking, list and string processing, air traffic control, relaxation problems, language translations and intelligent network routing systems."

Some authors [7, 13, 26, 41] use the terms *content-addressable memory* and *associative memory* interchangeably. However, as pointed out in [30], the proper meaning of *associativity* applies to the more abstract concept of links between the meanings of data items — for example, associating the word "fire" with the word "hot" — as opposed to a particular organization of storage elements that facilitate a simultaneous bit-pattern matching operation; consequently, we will use only the term *content-addressable*.

The potential utility of CAMs has been recognized by the computer industry at least since the late sixties [30, 40]; however, the implementation of such memories has been hindered by the following factors reported by Chisvin and Duckworth in 1989 [13]:

- functional and design complexity,
- relatively high cost of storage capacity,
- poor storage density compared to conventional memory,
- slow access time,
- lack of software to properly utilize CAM's capabilities.

Since then, implementation methods have improved dramatically, but because of the inherent higher complexity of a CAM cell (exceeding that of static RAM), CAMs are unlikely to replace RAMs as general purpose memories, although general purpose CAM designs have been investigated [2].

Most CAMs manufactured today can be found in custom designs, and are often embedded in larger circuits. Many application-specific CAM configurations have been reported [3, 7, 18, 24, 26, 28, 31, 32, 36, 41, 45, 50]. The most significant functional differences between these configurations include the accessibility by address as well as content, match-and-update, resolution of multiple hits, synchronous or asynchronous operation, etc., resulting in diverse implementations of CAM's peripheral circuitry. However, the design of the storage element of a core cell, in most cases, is similar and consists of a cross-coupled inverter circuit, such as those found in static RAMs [2, 3, 7, 18, 26, 31, 40, 41, 45]. Different designs of the core cell's comparison circuitry represent attempts to address various electrical pitfalls such as data-dependent bit line loads, or charge sharing problems [44].

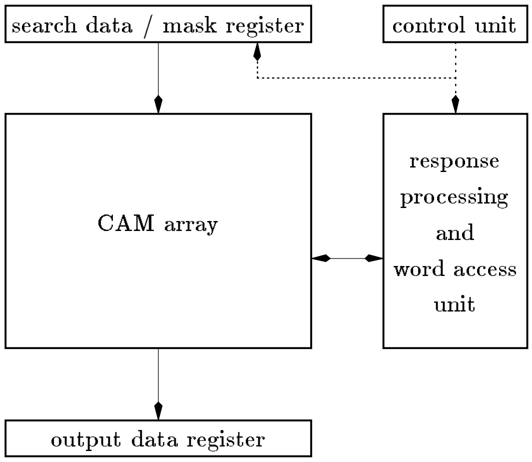


Figure 5.1: CAM: a block diagram.

Notable exceptions to the static CAM schema are dynamic implementations of the core cell [35, 38, 53, 56]. They are more complex than those of a typical high-density dynamic RAM, due to the necessity of charge retention during the simultaneous data comparison process. Like all dynamic memory implementations, these CAMs have to incorporate additional refresh circuitry.

Embedded memories of ASICs are usually static, as they are considered to be less troublesome [54], easier to build, and are generally faster than dynamic configurations. Hence, the analysis in this thesis is limited to static memories.

5.1 Overview of CAM Architecture

As mentioned earlier, most of the CAMs manufactured today are designed with a specific application in mind. Before analyzing some of these various implementations, we take a brief look at general components of a CAM. A generic architecture of a CAM and its basic cell are depicted in Fig. 5.1.

All CAM implementations consist of the following functional blocks:

- **data and mask register,** which stores a particular word according to which memory is to be accessed. Since it is possible that a part of data word may be irrelevant to the search criteria, the ability to mask irrelevant parts is usually provided. The search key is, therefore, composed of the parts of the word stored in the data register that have not been masked out.

- **response processing and word access unit,** which determines the functionality of the CAM. The implementation of this functional block is the main source of differences between various CAM designs. This block handles resolution of multiple hits, keeps track of unused locations for use during write operations, and with other application specific operations. The unit also often contains an address register for reading and writing in RAM mode.

- **output data register,** which incorporates sensing circuitry that retrieves and holds words picked up from bit lines during a read operation.

- **CAM array,** which stores the data. It consists of a homogeneous matrix of core cells. A number of existing designs will be considered in the next section.

- **Control unit,** which is used to coordinate the operation of the aforementioned blocks.

5.2 Static CAM Cell Circuits

A common CMOS implementation of a CAM cell is depicted in Fig. 5.2 (a). It is composed of a typical six-transistor SRAM cell (*storage section*) and a three-transistor matching circuitry (*comparison section*), which brings the transistor count of this CAM cell to nine. Various special-purpose CAM circuits have been developed using this design [4, 18, 22, 26].

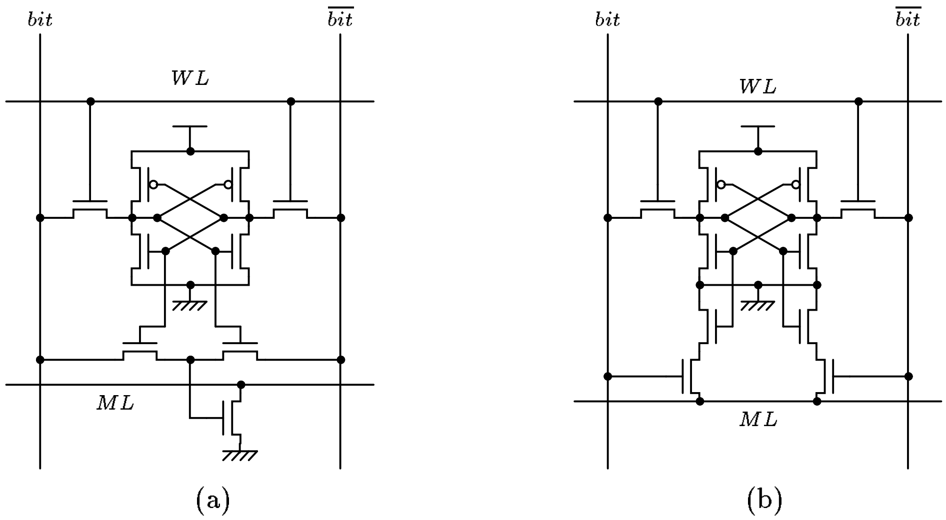


Figure 5.2: CMOS implementations of a single-port static CAM cell.

'Read' and 'write' operations are identical to to those in a SRAM cell. During a 'write' operation, the *bit* and $\overline{bit}$ lines are driven to complementary logic values representing the bit to be stored. Next, by raising and then lowering the word line *WL* the bit is stored by forcing the inverter outputs to coincide with the values on the *bit* and $\overline{bit}$ lines. Unlike SRAMs, CAMs have one more 'write' operation - 'write $\times$', which preserves the previous state of the cell. It is used for bit-masking purposes, where only a part of the word is to be overwritten. During a 'write $\times$' both *bit* and $\overline{bit}$ are driven to high voltage.

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

The process of searching is unique to the CAM, and is performed in the comparison section of the cell. In Fig. 5.2 (a), this section consists of the bottom three transistors of the cell's circuit. The match line *ML* together with the bottom-most transistor constitute a wired-AND for all the cells in the memory word. This means that the *ML* line will remain isolated from ground as long as all the transistors connecting it to ground are not conducting. The remaining two transistors implement an XOR function between the *bit* and $\overline{bit}$ lines and values stored on the outputs of the inverters. Note that only one of these transistors conducts at any given time. The compare operation is a two-step process. Initially, both bit lines are grounded and the *ML* line in the CAM cell is pre-charged. Next, bit lines are driven according to the search key. If a mismatch occurs, the bottom-most transistor will be forced to conduct, thus discharging the *ML* line. Note that if a particular bit in the search word is masked out, both corresponding *bit* and $\overline{bit}$ lines remain grounded, thus preventing any influence on the *ML* line. Such masking is called a 'compare $\times$' operation.

This design has a few shortcomings. In order to discharge the *ML* line, a logical 1 has to appear on the gate of the bottom-most transistor. This value is supplied by one of the $bit/\overline{bit}$ lines via a conducting n-MOS transistor, which causes a voltage drop of approximately $0.7V$. A 'weak' 1 appears on the gate of the bottom-most transistor, effectively excluding this design from low power applications. Moreover, these n-MOS pass transistors constitute variable loads for the differential *bit* lines. This results in unpredictable delays during 'read' and 'write' operations and, in effect, slower worst-case operation [44].

Another problem with this implementation is the fact that the state of the match line is affected by non-compare operations. Therefore, both $bit/\overline{bit}$ lines have to be discharged, and the match line has to be pre-charged before an actual compare operation, causing additional delays.

A ten-transistor CAM cell design has been reported in [28, 34]. It differs slightly from the previous cell in its implementation of the matching section, as shown in Fig. 5.2 (b). As before, it contains two transistors controlled by the storage section; however, here, each of the $bit/\overline{bit}$ lines drives the gate of a dedicated transistor. Since the loads on $bit/\overline{bit}$ lines

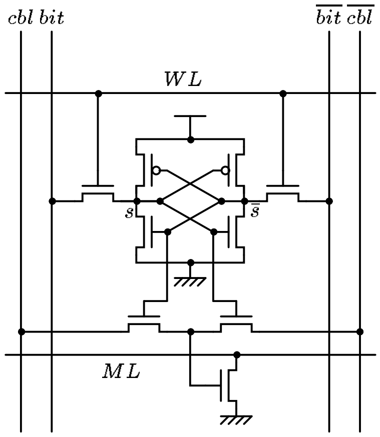
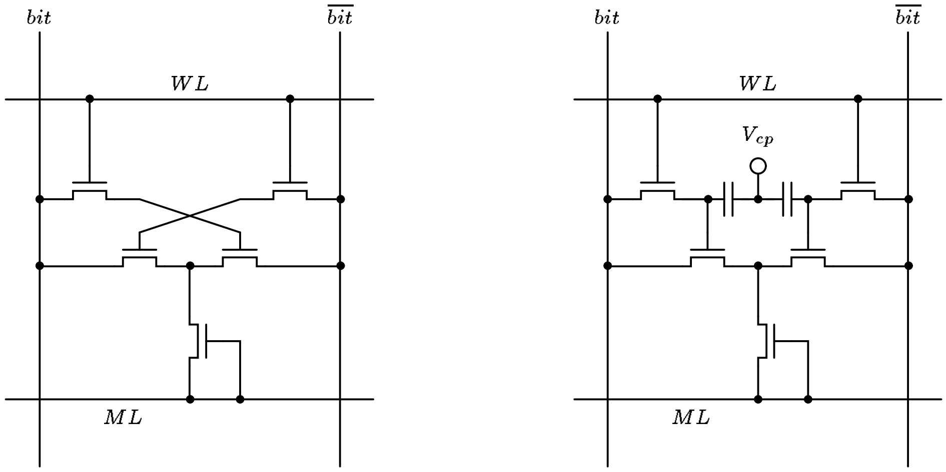


Figure 5.3: Dual-port static CAM cell.

are no longer data-dependent and the discharge path is gated directly by the $bit/\overline{bit}$ lines, this design could yield itself to low power, high speed applications. Unfortunately, this design, as well as the one proposed by McAuley and Cotton [36], are prone to accidental match-line discharge due to charge-sharing across stray capacitances on the dedicated transistors as described in [44]. This problem has been addressed by reversing the order of the transistors between the match line and ground [44, 52].

Although also ill-suited for low power, a dual-port implementation of a CAM cell presented in Fig. 5.3, is free from the shortcomings of the previous cells [7, 31, 44, 45]. Separate pairs of differential lines are used: $bit/\overline{bit}$ lines for reading and writing data, and compare bit lines ($cbl/\overline{cbl}$) for performing matching operations. Constant loads on the $bit/\overline{bit}$ lines precisely define delays during reads and writes. The match line is not affected by non-compare operations and, therefore, can be kept pre-charged by default, resulting in faster 'compare' operations. The separation of differential lines also facilitates efficient match-and-update operations required, for example, in real-time image processing applications [45].



(a) (b)

Figure 5.4: Dynamic CAM cell designs. (a) Wade, (b) Yamagata.

5.3 Dynamic CAM Cell Circuits

The CAM cell implementations presented so far were all static. For applications where speed and robustness are not as critical as high storage capacity, a dynamic CAM cell implementation has been considered [35, 38, 53, 56]. The increase of bit density has been accomplished by reduction of the number of transistors in each cell. Two such designs: by Wade and Sodini [53], and Yamagata et al. [56], are shown in Fig. 5.4 (a) and (b), respectively.

The ability to compare the entire cell array contents against a search key requires that DCAM cells retain their charge even in an event of a mismatch. This constraint precludes the use of destructive 'read' techniques, dominant in DRAM designs.

In the circuit presented by in Fig. 5.4 (a), data is stored as an electric charge on the gate capacitance of a storage transistor. Cross-coupling of the bit lines facilitates higher storage charges due to the presence of an inversion layer caused by a substantial gate-to-source voltage V_{gs}, thus overcoming problems encountered in earlier designs [38].

This cross-coupling scheme requires inverted values on the *bit* and $\overline{bit}$ lines during a write operation.

A 'write $\times$' operation involves driving both bit lines to 0 during a write operation; this action results in no charge being stored on either transistor. In effect, such a cell stores no value and the *ML* line is truly isolated at that location. The ability of storing no value is indigenous solely to dynamic cells, in contrast to static ones which always hold some value.

Compare operations are performed by first pre-charging both $bit/\overline{bit}$ lines then the match line *ML*, and finally driving the $bit/\overline{bit}$ lines to values reflecting the search key in a non-inverted manner. A mismatch will discharge *ML* through the bit line set to 0. Unlike in their static counterparts, 'compare $\times$' operations in dynamic CAM cells entail keeping the bit lines at 1, thus preventing *ML* from discharging.

The above-mentioned design is limited by the physical dimensions of the storage transistors in the circuit, since charge stored on the gate of a transistor is directly proportional to its size. A minimum storage capacitance of $30fF$ is required in order to preserve data integrity from alpha-particle-induced soft errors. Capacitances of this magnitude are difficult to obtain in transistors developed using sub-micron processes. In the circuit presented in Fig. 5.4 (b) this issue has been addressed by augmenting the gate capacitances of the storage transistors with stacked capacitors, thus facilitating a successful utilization of a $0.8\mu m$ process. The capacitors are configured differentially to a reference voltage V_{cp} equal to half of the supply voltage. The utilization of stacked capacitors as a means of increasing storage charge is superior to the cross-coupling schema presented above. Moreover, the absence of cross-coupling makes this design closely related to that presented by Mundy [38]. The 'write' and 'compare' operations of this dynamic CAM cell are performed identically to those presented above for Wade's CAM design, Fig. 5.4 (a), except that the bit line values need not be inverted during the write operation. The approach taken by Yamagata, Fig. 5.4 (b), has the potential to achieve the highest bit packing density, akin to that of DRAMs, though at the expense of performance.

5.4 Summary

In this chapter we have given an overview of a CAM architecture. We have presented several different CAM cell designs, both static and dynamic, and commented on their strengths and weaknesses. It is important to note that although all CAMs will perform the same operations, the way these operations are implemented varies among cell designs. For this reason it is reasonable to expect that these cells will exhibit different faulty behaviors in the presence of similar faults (e.g., *bit*-line stuck-at 0).

Chapter 6

Survey of CAM Tests

All memory circuits require extensive testing and CAMs are no exception. Unfortunately, the literature concerning testing of CAMs is very sparse. Giles and Hunter [19] present a simple methodology for testing for (cell) stuck-at-faults. A scheme for the detection of pattern-sensitivity faults along with stuck-at faults has been presented by Mazumder et al. [34, 35]. A built-in self test (BIST) algorithm called SMARCH [39], originally designed for testing embedded RAMs, has been extended by Kornachuk et al. [31] to testing embedded CAMs. Al-Assadi et al. [4] have proposed a functional fault model for a CAM architecture; this model is based on a fault model developed for a single cell. This fault model has been expanded and an appropriate test has been developed by Lin and Wu [33].

Algorithms reported in [19, 31, 33], are applicable to CAM designs which incorporate an explicit word addressing scheme of the type found in conventional RAMs. They are not applicable to designs with implicit entry addressing, such as the design presented by McAuley and Cotton [36]. The algorithm in [34] uses a special addressing mode which requires additional *design for testability* circuitry. The following sections describe these algorithms.

6.1 Giles and Hunter (1985)

One of the first tests for CAMs was presented by Giles and Hunter [19]; we refer to this test as $T_{G\&H}$. The following eight steps of this test are quoted from [19]; we have relabeled the array dimension symbols to be consistent with our notation, where n is the number of words (rows) and l is the number of bits per word (columns):

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

To illustrate this algorithm we consider a CAM array of four 2-bit words; thus $n = 4$, $l = 2$. CAM array's contents and test sequences for each steps is presented in Table 6.1.

The authors claim that this test detects all single stuck-at-faults in the CAM array found in the Motorola MC68851 Paged Memory Management Unit, but give no proof that this indeed is the case. The transistor circuit for this CAM cell was not provided.

Table 6.1: Example of test algorithm by Giles and Hunter.

Step 1	Step 2	Step 3		Step 4	Step 5
Array	Sequence	Array	Sequence	Array	Sequence
↓ 00	11,10,01,00	↓ 11	00,01,10,11	↑ 00	00,01,10,11
↓ 01		↓ 10		↑ 01	
↓ 10		↓ 01		↑ 10	
↓ 11		↓ 00		↑ 11	

Step 6	Step 7	Step 8	
Array	Sequence	Array	Sequence
00	10,01	11	01,10
00		11	
00		11	
00		11	

The CAM array described in [19] has a single primary output called HIT/$\overline{\text{MISS}}$, which is a distributed OR of all the match lines. The complexity of this algorithm is $8n + 2l$.

Certain hardware enhancements aimed at decreasing this complexity have also been proposed [19]:

- Augmenting the address decoder hardware with a "bulk load" ability can reduce the complexity of this algorithm to $6n + 2l + 2$.
- An "all-hit" detector (an AND operation on all match lines) will allow for algorithm modifications resulting in reduction of its complexity to $2l + 6$.

A detailed analysis of this test is presented in Chapter 9.

6.2 Mazumder et al. (1987)

Mazumder et al. [34] propose a test that can detect pattern-sensitivity faults as well as stuck-at faults. In general, faults are considered to be 'pattern-sensitive' if a faulty behavior of a cell is manifested only when the remaining cells hold specific values. In their

n \ l	0	1	2	3	4	5	6
0	0	1	2	3	4	0	1
1	2	3	4	0	1	2	3
2	4	0	1	2	3	4	0
3	1	2	3	4	0	1	2
4	3	4	0	1	2	3	4
5	0	1	2	3	4	0	1

Figure 6.1: Cell number assignment [34].

analysis, the authors make assumptions about the effect of certain operations on the state of the accessed memory cells, and define a restricted pattern-sensitive fault model (PSF):

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

The testing scheme presented in [34] requires incorporation of special circuitry in the match sensing periphery that allows simultaneous comparisons of every fifth match line. This requirement is imposed by partitioning the CAM array into a "mosaic" of adjacent von Neumann neighborhoods, which repeats itself every fifth row as shown in Fig. 6.1. Binary sequences are repeatedly written to the CAM array, in a manner that only one cell per neighborhood changes state. These binary patterns are established on the basis of a graph-theoretic analysis and are presented in Table 6.2.

Table 6.2: Test patterns used in the algorithm by Mazumder et al. [34].

Op. #	$s_4 s_3 s_2 s_1 s_0$	Op. #	$s_4 s_3 s_2 s_1 s_0$	Op. #	$s_4 s_3 s_2 s_1 s_0$	Op. #	$s_4 s_3 s_2 s_1 s_0$
0	00000	8	00011	16	00110	24	00101
1	01000	9	01011	17	01110	25	01101
2	11000	10	11011	18	11110	26	11101
3	10000	11	10011	19	10110	27	10101
4	10001	12	10010	20	10111	28	10100
5	11001	13	11010	21	11111	29	11100
6	01001	14	01010	22	01111	30	01100
7	00001	15	00010	23	00111	31	00100

The algorithm presented by Mazumder et al. [34] is described below; we have relabeled the array dimension variables to be consistent with our notation, where n is the number of words (rows) and l is the number of bits per word (columns):

1. Write 0 into every cell of CAM.
2. Do l compares walking a 1 through a field of 0's. (This is identical to steps 6 and 7 in the Giles and Hunter algorithm.)
3. Load the first of 32 patterns into the array[1].
4. A search for the loaded pattern should result in n mod 5 matches.
5. If the 21st pattern has been loaded (all 1's), do l compares walking a 0 through a field of 1's.
6. Load the next pattern. Repeat the previous two steps until all 32 patterns have been used.

This algorithm has a computational complexity of $33n + 2l + 160$. As stated in [34], this complexity of this algorithm can be reduced to $2l + 325$ by implementing changes

[1]This is actually done five times, every fifth line. See [34] for details.

proposed by Giles and Hunter [19] to the decoder circuitry.

This test imposes extra requirements on the addressing scheme which are not available in most CAMs. Also, the test's correctness is based on the assumption that 'read' operations do not affect the state of a cell; in the next chapter we show that this assumption is violated by some possible faults in our circuit-based fault model. For these reasons we exclude this test from further consideration.

6.3 Kornachuk et al. (1994)

The test presented by Kornachuk et al. [31] is a built-in self test (BIST) for embedded CAMs that use the cell of Fig. 5.3. We refer to this test as $T_{C\text{-}SM}$. It is an adaptation of the SMarch test, of length $24nl$, originally developed for testing embedded SRAMs [39]. SMarch has been shown to detect stuck-at and coupling faults. In fact, SMarch is a serialized version of the 'March C-' test [20]. The serialization was done in order to utilize a scan-path circuit of the embedded SRAM.

The SMarch test consists of six steps (elements) [39]. We have re-labeled the array dimension symbols to be consistent with the notation used in this thesis, where i and j are row and column indices, respectively. The memory array consists of n rows and l columns. An $r_0^{(i,j)}$ denotes a 'read' of the cell (i,j) expecting a 0 as output. An $w_0^{(i,j)}$ denotes a 'write 0' to the cell (i,j). Operations $r_1^{(i,j)}$ and $w_1^{(i,j)}$ are similarly defined. The superscript $(\;)^{(l-1)\to 0}$ indicates l column-wise repetitions, starting from left to right. The reader should be familiar with [31, 39].

The 'compare' operations are implied, and performed simultaneously with every 'read' and 'write' operation. Since multiple matches can be expected, a priority encoder is used to produce the highest address where a match was obtained.

1. FOR address $i = 1$ to n: $(r_-^{(i,j)} w_0^{(i,j)})^{(l-1)\to 0}\ (r_0^{(i,j)} w_0^{(i,j)})^{(l-1)\to 0}$

{this step initializes the CAM with 0's }

2. FOR address $i = 1$ to n: $(r_0^{(i,j)} w_1^{(i,j)})^{(l-1)\to 0}$ $(r_1^{(i,j)} w_1^{(i,j)})^{(l-1)\to 0}$

   { read 0's and replace with 1's}

3. FOR address $i = 1$ to n: $(r_1^{(i,j)} w_0^{(i,j)})^{(l-1)\to 0}$ $(r_0^{(i,j)} w_0^{(i,j)})^{(l-1)\to 0}$

   { read 1's and replace with 0's}

4. FOR address $i = n$ to 1: $(r_0^{(i,j)} w_1^{(i,j)})^{(l-1)\to 0}$ $(r_1^{(i,j)} w_1^{(i,j)})^{(l-1)\to 0}$

   { read 0's and replace with 1's}

5. FOR address $i = n$ to 1:

   $(r_1^{(i,j)} w_0^{(i,j)})^{(l-1)\to 0}$ $(r_0^{(i,j)} w_0^{(i,j)})^{(l-1)\to 0}$

   { read 1's and replace with 0's}

6. FOR address $i = n$ to 1: $(r_0^{(i,j)} w_0^{(i,j)})^{(l-1)\to 0}$ $(r_0^{(i,j)} w_0^{(i,j)})^{(l-1)\to 0}$

   {only the first read is important. Final state is selectable.}

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

The adaptation of the SMARCH test to the CAM circuit was possible due to the dual-port nature of this particular CAM design, where one set of differential lines ($bit/\overline{bit}$) is used only for the 'read' and 'write' operations, and a second set of differential lines ($cbl/\overline{cbl}$) is dedicated solely to the 'compare' operations. Table 6.3 illustrates the execution of the second march element for the CAM BIST for a 3×3 sample CAM; after the first march element, all cells are 0. During $T_{C\text{-}SM}$, 'compare' operations occur in the same clock cycle as the 'read' and 'write' operations and thus are perceived to be executed

Table 6.3: CAM BIST March Element [31].

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

in parallel. To be precise, ‘compare’ operations are indeed performed concurrently with the ‘read’ operations, but they occur immediately after the completion of the ‘write’ operations; thus the newly stored value is the subject of the comparison. Since these operations are performed in parallel, the length of $T_{C\text{-}SM}$ remains $24nl$, although twice as many operations are actually executed. A detailed analysis of this test is presented in Chapter 9.

6.4 Al-Assadi et al. (1994)

So far, the fault coverage of testing algorithms has been established on the basis of fault models originally created for random-access memories. Al-Assadi et al. [4] produced a fault model of a CAM cell by performing a study of possible faults that may occur in the static, nine-transistor CMOS CAM cell circuit depicted in Fig. 5.2(a).

The behavior of a faulty CAM cell, under a single fault assumption, has been defined as follows [4]:

1. The cell's state is determined by the logic value of the node S.
2. The cell is *always matched* if the match operation always yields $\overline{BM} = 0$ regardless of the cell's state.
3. The cell is *always mismatched* if the match operation always yields $\overline{BM} = 1$ regardless of the cell's state.
4. The cell is *partly matched* if the match operation always yields $\overline{BM} = 1$ when $S = x$, and $\overline{BM} = 0$ when $S = \overline{x}$ regardless of the search key.
5. The cell is *conditionally matched* if the match operation yields proper result when $S = x$, but not for $S = \overline{x}$.
6. Some faults may cause spontaneous changes of the cell's state, resulting in improper match results. Such behavior is referred to as *complex/indeterminate.*

A summary of faults and corresponding behaviors is listed in Table 6.4 and Table 6.5.

The impact of all possible faults has been mapped onto the state of the match line resulting in a description of a functional fault model of a CAM cell. This fault model is presented in Table 6.6. It was also established that many faults in the peripheral circuitry

Table 6.4: Behavior of the CAM cell under storage cell faults [4].

Fault in storage cell	Write		Match		Effect on CAM
	0	1	0	1	(during a match operation)
Short:$V_{ss} - WL$	x	x	x	x	CAM sa-x
Short:$V_{ss} - bit$	$\surd$	?	?	?	cell always matched
Short:$V_{ss} - \overline{bit}$	?	$\surd$	?	?	cell always matched
Short:$V_{dd} - WL$	$\surd$	$\surd$	?	?	cell always matched
Short:$V_{dd} - bit$	?	$\surd$	?	$\surd$	conditional match-1
Short:$V_{dd} - \overline{bit}$	$\surd$	?	$\surd$	?	conditional match-0
Short:$WL - bit$	?	$\surd$	?	$\surd$	conditional match-1
Short:$WL - \overline{bit}$	$\surd$	?	$\surd$	?	conditional match-0
Short:$bit - \overline{bit}$	?	?	?	?	cell always mismatched
Short:$V_{dd} - S$	?	$\surd$	?	$\surd$	conditional match-1
Short:$V_{ss} - S$	$\surd$	?	$\surd$	?	conditional match-0
Short:$S - \overline{S}$	?	?	?	?	cell always matched
Short:$S - WL$	$\surd$	?	$\surd$	?	conditional match-0
Short:$S - bit$	$\surd$	$\surd$	?	?	cell always matched
Short:$V_{dd} - \overline{S}$	$\surd$	?	$\surd$	?	conditional match-0
Short:$V_{ss} - \overline{S}$	?	$\surd$	?	$\surd$	conditional match-1
Short:$\overline{S} - WL$	?	$\surd$	?	$\surd$	conditional match-1
Short:$\overline{S} - \overline{bit}$	$\surd$	$\surd$	?	?	cell always matched
T1 stuck-on	$\surd$	$\surd$	$\surd$	?	cell always matched
T1 stuck-open	?	$\surd$	?	$\surd$	conditional match-1
T2 stuck-on	$\surd$	$\surd$	$\surd$	$\surd$	fault-free
T2 stuck-open	$\surd$	$\surd$	$\surd$	$\surd$	fault-free
T3 stuck-on	?	$\surd$	?	$\surd$	conditional match-1
T3 stuck-open	$\surd$	$\surd$	$\surd$	?	conditional match-0
T4 stuck-on	$\surd$	$\surd$	$\surd$	$\surd$	fault-free
T4 stuck-open	$\surd$	$\surd$	$\surd$	$\surd$	fault-free
T5 stuck-on	$\surd$	?	$\surd$	?	conditional match-0
T5 stuck-open	$\surd$	$\surd$	?	$\surd$	conditional match-1
T6 stuck-on	$\surd$	$\surd$	?	?	cell always matched
T6 stuck-open	$\surd$	?	$\surd$	?	conditional match-1

$\surd$:operation is proper

?: operation is improper

Table 6.5: Faults in the comparison logic of a CAM cell [4].

Fault in storage cell	Write		Match		Effect on CAM
	0	1	0	1	(during a match operation)
Short:$bit - \overline{BM}$	$\surd$	$\surd$	?	?	conditional match-1
Short:$\overline{bit} - \overline{BM}$	$\surd$	$\surd$	?	?	conditional match-0
Short:$S - \overline{bit}$	?	?	?	?	complex/indeterminate
Short:$\overline{S} - bit$	?	?	?	?	complex/indeterminate
Short:$\overline{S} - \overline{BM}$	$\surd$	$\surd$	s	s	partial match-1
Short:$S - \overline{BM}$	$\surd$	$\surd$	s	s	partial match-0
Short:$bit - ML$	$\surd$	$\surd$	?	?	complex/indeterminate
Short: $V_{ss} - ML$	?	$\surd$	?	?	cell always mismatched
Short:$V_{ss} - \overline{BM}$	?	$\surd$	$\surd$	?	cell always matched
T7 stuck-on	$\surd$	$\surd$	?	?	conditional match-1
T7 stuck-open	$\surd$	$\surd$	s	s	conditional match-0
T8 stuck-on	$\surd$	$\surd$	?	?	conditional match-0
T8 stuck-open	$\surd$	$\surd$	s	s	conditional match-1
T9 stuck-on	$\surd$	$\surd$	?	?	cell always mismatched
T9 stuck-open	$\surd$	$\surd$	?	?	cell always matched

s: CAM state dependent

can be mapped to equivalent faults in the CAM array.

The study presented by Al-Assadi et al. [4] is not without flaws. In Table 6.4, for example, a 'match 0' operation in the presence of the 'T1 stuck-on' fault is reported as proper. We will show that this is not the case.

Suppose that the initial state of the cell is $S = 1$. When transistor $T1$ is 'stuck-on' the cell is always affected by the state of the *bit* line. At the beginning of a 'match 0' operation both bit lines must be grounded to allow for a match line pre-charge. As a result of a grounded *bit* line, S becomes 0 and transistor $T8$ stops conducting. Although $\overline{bit}$ line is raised high to represent a 0 during the comparison process, the $\overline{BM}$ line remains low and, consequently, ML is not discharged. In effect, the operation produces an erroneous result. To further support this argument, the listing for the symmetric 'T6 stuck-on' fault in Table 6.4 correctly states that both 'match 0' and 'match 1' are improper.

Table 6.6: Functional fault model for the CAM cell [4].

Functional fault in storage cell	Effect on CAM
coupled *bit* & coupled $\overline{bit}$	cell always matched
short between V_{ss} and *bit* or $\overline{bit}$	cell always matched
coupling involving word line with *bit* line or with node S	conditional match-1
coupling involving word line with $\overline{bit}$ line or with node $\overline{S}$	conditional match-0
word unaccessible	CAM sa-x
word always accessible	cell always matched
coupling involving *bit* & $\overline{bit}$	cell always mismatched
sa-1	conditional match-1
sa-0	conditional match-0
stuck-on/open in XOR logic	conditional match
bit-match transistor stuck-open	cell always matched
bit-match transistor stuck-on	cell always mismatched
shorts between *bit*, $\overline{bit}$ and $\overline{BM}$	conditional match
shorts between nodes S & $\overline{S}$ and $\overline{BM}$	partial match

More flaws can be found in Table 6.5. For example, in the presence of the fault 'Short:$V_{ss} - \overline{BM}$' a 'match 0' operation is reported as proper. However, when the state of the cell is $S = 1$ this operation will result in an erroneous match. This fact is confirmed by a comment in [4].

The results presented in [4] are the first systematic fault analysis of a single-port CAM. They indicate that the effects of defects found in a CAM cell can be detected by improper 'compare' operations. Unfortunately, this work is plagued with typographical errors, hence the detailed results require careful re-evaluation.

6.5 Lin and Wu (1998)

Lin and Wu [33] continue the work of Al-Assadi et al. [4] described in the previous section. They expand the fault model for the cell in Fig. 5.2(a) to include faults that affect *valid* bits, set by the 'write' operations and reset by special 'erase' operations. These bits indicate which words should be included in 'compare' operations, and are used in many CAM designs. The additional CAM-specific faults are:

1. The *stuck-valid fault*, caused when a word's valid bit is always set.
2. The *stuck-invalid fault*, caused when a word's valid bit can never be set.
3. The *cross-match fault*, caused by a short between *bit* and $\overline{bit}$ which belong to two neighboring cells of the same word.

The *always matched* and *always mismatched* faults have been renamed as *stuck matched* and *stuck mismatched* faults, respectively.

Lin and Wu also introduce the T_{CAM} test, of length $9n + 5l$, that detects all faults in their model. The T_{CAM} test consists of five steps (march elements). We have re-labeled the array dimension symbols to be consistent with the notation used in this thesis.

$$\begin{aligned} T_{CAM} \;\; = \;\; & (w^i_{0\ldots0})^{\updownarrow}[c_{\times^{j-1}1\times^{l-j}}]^{1\rightarrow l}[c_{0^{j-1}\times 0^{l-j}}]^{1\rightarrow l}[c_{0^{j-1}10^{l-j}}]^{1\rightarrow l} \\ & (w^i_{1\ldots1}c_{1\ldots1}w^i_{0\ldots0})^{\updownarrow} \\ & (w^i_{1\ldots1})^{\updownarrow}[c_{\times^{j-1}0\times^{l-j}}]^{1\rightarrow l} \\ & (e^i)^{\updownarrow}[c_{\times^{j-1}1\times^{l-j}}]^{1\rightarrow l} \\ & (w^i_{0\ldots0}c_{0\ldots0}w^i_{1\ldots1})^{\updownarrow}, \end{aligned}$$

where $(w^i_{0\ldots0})^{\updownarrow}$ denotes the writing of all-0 words in order either from n to 1 or from 1 to n, $\times^{j-1}0\times^{l-j}$ is the l-bit word with 0 in position j and $\times$ in every other position, the symbol $[c_{\times^{j-1}0\times^{l-j}}]^{1\rightarrow l}$ represents a sequence of l 'compare' operations to words of the form $\times^{j-1}0\times^{l-j}$, where j varies from 1 to l, and e is the 'erase' operation. The remaining symbols are similarly defined.

This work has several shortcomings.

- Faults associated with the transistors that comprise the cross-coupled inverters in Fig. 5.2(a) are not considered.

- Shorts between *bit* and GND or between *s* and *bit* do not constitute a *stuck-matched* fault. By definition of a *stuck-matched* fault, the result of a 'compare' operation is an unconditional match regardless of the state of the cell.

  When a short between *bit* and GND exists, a 6-transistor CMOS storage element can be in either of the two stable states at power-up. A 'write 0' works correctly. However, a 'write 1' operation causes the cell to become metastable; hence the output is not reliable. Since the resolution of the metastable state is non-deterministic, there is no guarantee that the cell will not end up in state 1 again. When the faulty cell is in state 1, a 'compare 0' results in a mismatch, and a 'compare 1' results in a match. Only in state 0, all 'compare' operations result in a match. Analogous argument applies to the $\overline{bit}$-*GND* fault and the 'write 0' operation.

  If a short between *s* and *bit* exists and the cell is in state 0, then $\bar{s} = 1$ and T_3 is on. During a 'compare 1', after the match line is floated-high, the *bit* line becomes high, causing T_5 to conduct. This event will also cause the cell to change its state to 1. However, this change of state will be slower than that during a 'write 1' operation. Since not until $\bar{s}$ becomes 0, will T_3 stop conducting, there is no guarantee that the match line will not discharge sufficiently to be interpreted as a mismatch.

In view of these inconsistencies, the validity of the fault model presented requires re-examination.

6.6 Summary

We have described three tests developed explicitly for CAMs. The fault models used for these tests are often not sufficient to describe various faults indigenous to CAMs. Al-Assadi et al. [4] were first to develop a fault model specifically for a CAM. Lin and Wu [33] expanded on that work, defined additional faults and designed a test for their fault model. Yet their fault model is still inadequate, and some of the fault analysis is flawed. Moreover, this fault model pertains only to single-port CAMs. Due to the variety of CAM cell designs a more general approach is necessary, one that can establish fault models that reflect defects that occur in a given CAM cell design.

Chapter 7

CAM Model

In this chapter we present behavioral models for a fault-free CAM cell, input stuck-at, state stuck-at, transistor stuck-(on/open) and bridging faults. These fault models have been developed under the single fault assumption. The first four fault models have been previously reported [47, 48]. We have developed these cell-specific fault models based on a transistor circuit of a static CMOS CAM cell, shown in Fig. 7.1, that is utilized by Nortel Corporation in their telecommunication ASICs and previously described in Chapter 5.

The circuit of this cell can be divided on the basis of its functionality into a *storage section* and a *comparison section.*

The storage section has two cross-coupled CMOS inverters (T_1 - T_4), differential bit lines ($bit/\overline{bit}$) used for reading and writing data into a column of cells, and a word select line (WL) that enables these operations in a row of cells via transistors T_5 and T_6. In a quiescent state WL is driven low, the $bit/\overline{bit}$ lines are driven high and the cell stores either 0 or 1. During a 'write 1' or a 'write 0' operation, the $bit/\overline{bit}$ lines are driven to a true/complementary representation of the desired bit value. Raising and then lowering WL stores the bit in the cell. Changes of the cell's state (e.g. during a 'write 1' when $s = 0$) are a result of the dominant influence of stronger pull-down transistors. Weaker

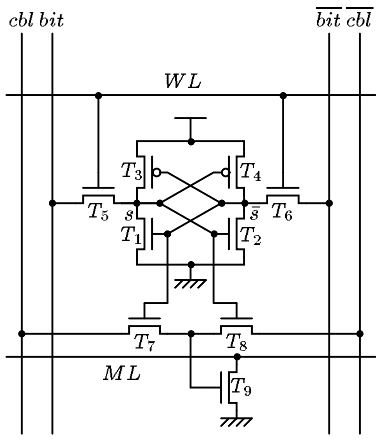


Figure 7.1: Dual-port static CAM cell with dedicated compare lines.

pull-up transistors maintain quiescent states of the cell. The 'write $\times$' operation preserves the state of the cell. This operation can be implemented in two ways. One implementation requires both of the $bit/\overline{bit}$ lines to be driven high; proper margining of transistors T_1 - T_4 assures the preservation of the cell's state. The other implementation is exactly like a 'read' operation described below, except that any output is disregarded. We have analyzed the former implementation, and drawn conclusions on the latter by studying the 'read' operation itself.

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

The comparison section consists of transistors T_7, T_8 and T_9, the differential compare bit lines ($cbl/\overline{cbl}$) for matching operations, and the match line (ML). In a quiescent state $cbl/\overline{cbl}$ are driven low, ML is driven high and either T_7 or T_8 conducts, depending on the state of the cell. During a 'compare' operation ML is first isolated. Next, $cbl/\overline{cbl}$ are driven according to the desired search key. If a mismatch occurs, transistor T_9 is forced to conduct, thus discharging ML. In the case of a 'compare $\times$' operation, both of $cbl/\overline{cbl}$

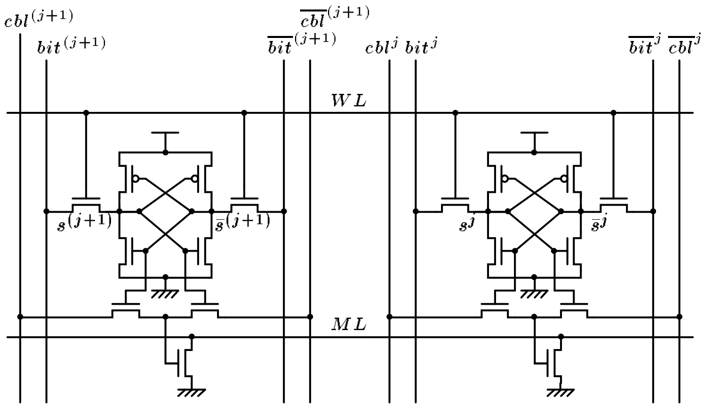


Figure 7.2: Two neighboring CAM cells.

remain low, resulting in an unconditional match.

Altogether seven operations can be performed on this cell: 'read', 'write 0', 'write 1', 'write ×', 'compare 0', 'compare 1' and 'compare ×'.

In this chapter we also consider two adjacent CAM cells, that comprise a part of a single memory word, as depicted in Fig. 7.2. Since the two neighboring cells share the *WL* and *ML* lines, only the same type of operation can be performed on both cells at any time, i.e. 'read', 'write', or 'compare'. Hence nineteen operations are possible on two cells: a 'read' operation, which returns a 00, 01, 10 or 11, 'write 00', 'write 01', 'write 0×', 'write 10', 'write 11', 'write 1×', 'write ×0', 'write ×1', 'write ××', 'compare 00', 'compare 01', 'compare 0×', 'compare 10', 'compare 11', 'compare 1×', 'compare ×0', 'compare ×1', 'compare ××'. The output generated by 'compare' operations, i.e., the *ML*, will indicate a match only if both cells store values that are compatible with the search key.

7.1 Fault-free Cell

Since no clock signal is supplied to individual cells, a single CAM cell can be viewed as an asynchronous sequential circuit, whose behavior can be modeled by the block diagram of Fig. 7.3.

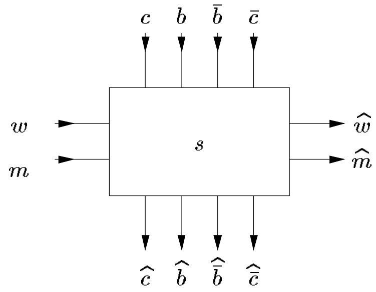


Figure 7.3: Model of the CAM cell of Fig. 7.1.

7.1.1 Event-Sequence Model

Since $bit/\overline{bit}$ lines and ML are used for both input and output in the circuit, they are represented by separate variables in the model. For brevity we use b, $\bar{b}$, w, etc., for $bit/\overline{bit}$, WL, etc. Now, b, $\bar{b}$ and m are inputs, and $\widehat{b}$, $\widehat{\bar{b}}$, and $\widehat{m}$ are outputs. Although WL and $cbl/\overline{cbl}$ are not usually meant to provide any output, monitoring the state of these lines, if possible, might improve the CAM's testability. For this reason, we generalize our model to include these lines; w, c, $\bar{c}$ for input, and $\widehat{w}$, $\widehat{c}$, $\widehat{\bar{c}}$ for output.

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

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

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

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

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

The domain of each variable in state C is the set $Y = \{0, 1, \tilde{0}, \tilde{1}\}$. The values 0 and 1 represent lines driven to the logic values 0 and 1 respectively, while $\tilde{0}$ and $\tilde{1}$ denote lines that were first discharged and then isolated (*floated low*) or pre-charged and then isolated (*floated high*). The CAM cell has two possible initial states: $C = 0\ 11\ 00\ 1 \cdot 0$ and $C = 0\ 11\ 00\ 1 \cdot 1$.

The behavior of a cell is represented by sequences of events that take place during the seven operations. Table 7.1 lists events that occur during these operations. By events we mean changes in the value of the total state C. The occurrences of these events are ordered from top to bottom. Note that, for every operation, the top and bottom entries in each column are initial states.

Example: 'Write 0' operation from state 1.

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

It should be noted that the analyses presented in Table 7.1 are done under the assumption that operations occur one at a time. However, 'compare' operations utilize a separate set of differential lines and thus some concurrent executions are feasible. For

Table 7.1: Read, write and compare operations in a fault-free CAM cell.

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

Write operations						
	$(s = 0)$			$(s = 1)$		
	w_0	w_1	$w_\times$	w_0	w_1	$w_\times$
Description	$w\ b\bar{b}\ c\bar{c}\ m{\cdot}s$	$w\ b\bar{b}\ c\bar{c}\ m{\cdot}s$	$w\ b\bar{b}\ c\bar{c}\ m{\cdot}s$	$w\ b\bar{b}\ c\bar{c}\ m{\cdot}s$	$w\ b\bar{b}\ c\bar{c}\ m{\cdot}s$	$w\ b\bar{b}\ c\bar{c}\ m{\cdot}s$
initial state	0 11 00 1·0	0 11 00 1·0	0 11 00 1·0	0 11 00 1·1	0 11 00 1·1	0 11 00 1·1
set $b/\bar{b}$	0 01 00 1·0	0 10 00 1·0	0 11 00 1·0	0 01 00 1·1	0 10 00 1·1	0 11 00 1·1
raise w	1 01 00 1·0	1 10 00 1·0	1 11 00 1·0	1 01 00 1·1	1 10 00 1·1	1 11 00 1·1
new state	1 01 00 1·0	1 10 00 1·1	1 11 00 1·0	1 01 00 1·0	1 10 00 1·1	1 11 00 1·1
lower w	0 01 00 1·0	0 10 00 1·1	0 11 00 1·0	0 01 00 1·0	0 10 00 1·1	0 11 00 1·1
raise $b/\bar{b}$	0 11 00 1·0	0 11 00 1·1	0 11 00 1·0	0 11 00 1·0	0 11 00 1·1	0 11 00 1·1

Compare operations						
	$(s = 0)$			$(s = 1)$		
	c_0	c_1	$c_\times$	c_0	c_1	$c_\times$
Description	$w\ b\bar{b}\ c\bar{c}\ m{\cdot}s$	$w\ b\bar{b}\ c\bar{c}\ m{\cdot}s$	$w\ b\bar{b}\ c\bar{c}\ m{\cdot}s$	$w\ b\bar{b}\ c\bar{c}\ m{\cdot}s$	$w\ b\bar{b}\ c\bar{c}\ m{\cdot}s$	$w\ b\bar{b}\ c\bar{c}\ m{\cdot}s$
initial state	0 11 00 1·0	0 11 00 1·0	0 11 00 1·0	0 11 00 1·1	0 11 00 1·1	0 11 00 1·1
float m	$0\ 11\ 00\ \tilde{1}{\cdot}0$	$0\ 11\ 00\ \tilde{1}{\cdot}0$	$0\ 11\ 00\ \tilde{1}{\cdot}0$	$0\ 11\ 00\ \tilde{1}{\cdot}1$	$0\ 11\ 00\ \tilde{1}{\cdot}1$	$0\ 11\ 00\ \tilde{1}{\cdot}1$
set $c/\bar{c}$	$0\ 11\ 01\ \tilde{1}{\cdot}0$	$0\ 11\ 10\ \tilde{1}{\cdot}0$	$0\ 11\ 00\ \tilde{1}{\cdot}0$	$0\ 11\ 01\ \tilde{1}{\cdot}1$	$0\ 11\ 10\ \tilde{1}{\cdot}1$	$0\ 11\ 00\ \tilde{1}{\cdot}1$
m discharges	$0\ 11\ 01\ \tilde{1}{\cdot}0$	0 11 10 0·0	$0\ 11\ 00\ \tilde{1}{\cdot}0$	0 11 01 0·1	$0\ 11\ 10\ \tilde{1}{\cdot}1$	$0\ 11\ 00\ \tilde{1}{\cdot}1$
read m & ground $c/\bar{c}$	$0\ 11\ 00\ \tilde{1}{\cdot}0$	$0\ 11\ 00\ \tilde{0}{\cdot}0$	$0\ 11\ 00\ \tilde{1}{\cdot}0$	$0\ 11\ 00\ \tilde{0}{\cdot}1$	$0\ 11\ 00\ \tilde{1}{\cdot}1$	$0\ 11\ 00\ \tilde{1}{\cdot}1$
raise m	0 11 00 1·0	0 11 00 1·0	0 11 00 1·0	0 11 00 1·1	0 11 00 1·1	0 11 00 1·1

example, a 'compare' operation can be performed in parallel with a 'read' operation, but it cannot be performed until a 'write' operation is completed. Modeling concurrent operations is an open research topic.

For bridging faults, we create a model for the two neighboring cells. We represent two neighboring cells of Fig. 7.2 as the block diagram of Fig. 7.4, where, for notational simplicity, input lines b', $\bar{b}'$, c', $\bar{c}'$ denote input lines of cell $j+1$, and b, $\bar{b}$, c, $\bar{c}$ denote those of cell j. Output lines are similarly represented. The total state of the two cells is

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

formally represented by the 22-tuple:

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

but for notational simplicity, we represent it symbolically as

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

where the input variables have been separated by spaces for readability, and the $\cdot$ has been inserted to separate the input variables from the internal state $S = \langle s', s \rangle$.

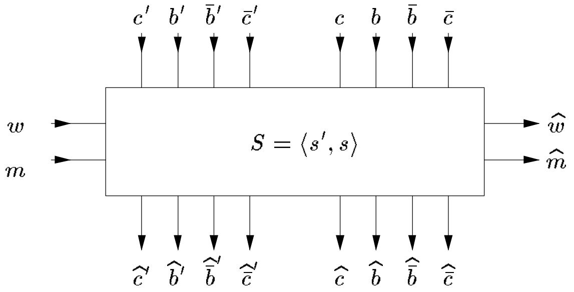


Figure 7.4: Model of two neighboring CAM cells.

A fault-free pair of cells has four possible initial states: $0\ 11\ 00\ \ 11\ 00\ 1 \cdot 00$, $0\ 11\ 00\ \ 11\ 00\ 1 \cdot 01$, $0\ 11\ 00\ \ 11\ 00\ 1 \cdot 10$, $0\ 11\ 00\ \ 11\ 00\ 1 \cdot 11$. The behavior of these cells is also represented by sequences of events that take place during the nineteen operations.

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

7.1.2 FSM Model

Most testing algorithms utilize sequences of 'read', 'write' and 'compare' operations as input, and observe the resulting output. Accordingly, we introduce an FSM model of a CAM cell. This model is derived from the event-sequence model of Section 7.1.1.

We represent the behavior of a fault-free CAM cell as a simplified block diagram, which is shown in Figure 7.5(a).

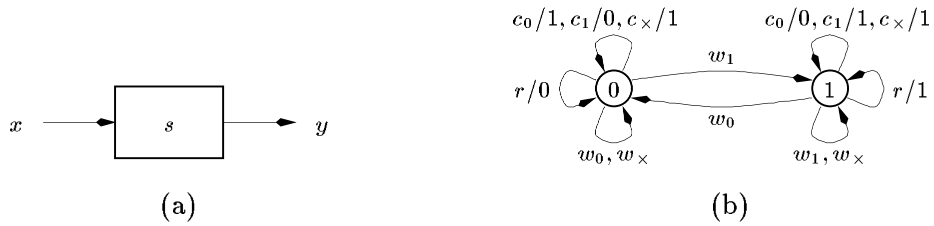


Figure 7.5: (a) Simplified behavioral model (b) Behavior of a fault-free cell.

The input x of this FSM comprises all seven operations of a CAM cell, and the output y comprises the responses to these operations without regard to the output's origin, i.e., the *bit*/$\overline{bit}$ lines or *ML*. State s represents the value stored by the cell. Formally, a *fault-free CAM cell* is a Mealy automaton

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

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

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

and

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

This automaton is depicted in Figure 7.5(b), where the symbol \$ has been omitted for clarity.

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

In a *faulty CAM cell* FSM, the faulty set of states Q' is $Q \cup \{I\}$ and the faulty set of output symbols Y' is $Y \cup \{I\}$. State I can be interpreted in two ways. From an "asynchronous" point of view it represents a temporary, metastable state, where the cell holds some indeterminate logic value. This interpretation is important due to the possibility of simultaneous operations in this type of CAM. From the "synchronous" perspective it represents the loss of information regarding the current state of the cell due to the non-deterministic resolution of a metastable state. In either case, state I is not considered as an initial state. Output symbol I stands for an intermediate logic value, which is caused in a faulty CMOS circuit when both pull-up and pull-down transistors simultaneously conduct.

7.2 Input Stuck-at Faults

Input stuck-at faults are all stuck-at-0 and stuck-at-1 faults that appear on input lines of a memory cell. Accordingly, we define the input stuck-at fault model to comprise these faults, under the single fault assumption. In order to model input stuck-at faults, their effect on the internal operation of the cell had to be determined. An event-sequence behavioral analysis has been performed for every input stuck-at fault; these analyses have been reported in [48] are presented in Appendix A.1.1. Subsequently, FSM models for each of these faults have been developed. The resulting *faulty CAM cells* are presented in Figs. 7.6 and 7.7, where incorrect operations and outputs are in boldface.

Table 7.2: A 'write 0' operation in a correct and faulty CAM cell.

Write 0 operation ($s = 1$)		
	Correct	$\bar{b}$-*sa-0*
Description	$w\ b\bar{b}\ c\bar{c}\ m{\cdot}s$	$w\ b\bar{b}\ c\bar{c}\ m{\cdot}s$
initial state	0 11 00 1·1	0 1**0** 00 1·1
set $b/\bar{b}$	0 01 00 1·1	0 0**0** 00 1·1
raise w	1 01 00 1·1	1 0**0** 00 1·1
new state	1 01 00 1·0	1 0**0** 00 1·**I**
lower w	0 01 00 1·0	0 0**0** 00 1·**I**
raise $b/\bar{b}$	0 11 00 1·0	0 1**0** 00 1·**0**|**1**

Example: 'Write 0' from state 1 in a correct cell and in the presence of the $\bar{b}$-*sa-0* fault. Initial state: 0 11 00 $1 \cdot 1$ (correct), 0 1**0** 00 $1 \cdot 1$ (faulty). First, b is lowered: 0 01 00 $1 \cdot 1$ (correct), 0 0**0** 00 $1 \cdot 1$ (faulty). Then w is raised: 1 01 00 $1 \cdot 1$ (correct), 1 0**0** 00 $1 \cdot 1$ (faulty). In the correct cell s changes: 1 01 00 $1 \cdot 0$. Since in the faulty cell both b and $\bar{b}$ are low, s becomes indeterminate: 1 0**0** 00 $1 \cdot \mathbf{I}$. Next, w is lowered: 0 01 00 $1 \cdot 0$ (correct), 0 0**0** 00 $1 \cdot \mathbf{I}$ (faulty). Finally, both b and $\bar{b}$ are raised: 0 11 00 $1 \cdot 0$ (correct), 0 1**0** 00 $1 \cdot \mathbf{0}|\mathbf{1}$ (faulty), and in the faulty cell s eventually becomes either 0 or 1. This sequence is summarized in Table 7.2.

From the event-sequence model we construct an FSM. In the presence of this fault, only operations w_0, $w_\times$ and r are incorrect, all others are correct; hence we get Fig. 7.6(c).

As the example indicates the b-*sa-0* and $\bar{b}$-*sa-0* faults increase the state set Q by the "indeterminate" state I.

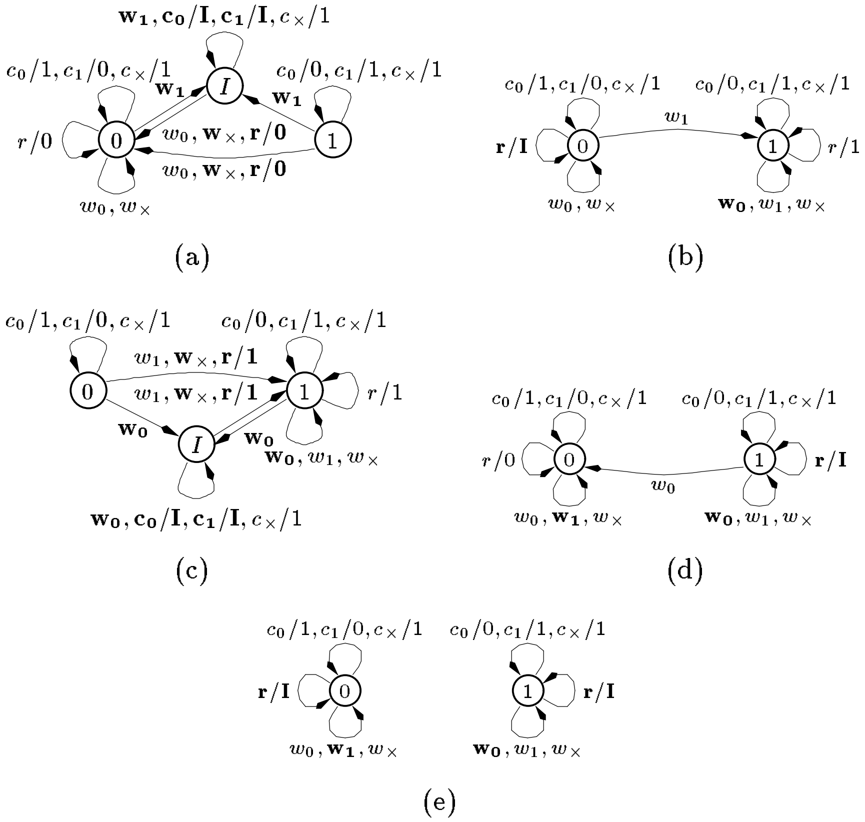


Figure 7.6: FSM models for faults: (a) *b-sa-0*, (b) *b-sa-1*, (c) $\bar{b}$*-sa-0*, (d) $\bar{b}$*-sa-1*, (e) *w-sa-0*.

For the two implementations of the 'write $\times$' operation, described at the beginning of this chapter, the reader may verify that in the presence of input stuck-at faults both 'read' and 'write $\times$' operations affect the cell in a similar manner (when the output is ignored). The FSM model presented here is, therefore, appropriate regardless of how the 'write $\times$' operation has been implemented.

The comparison of each of the faulty machines to the fault-free CAM cell has led to the derivation of simple tests for each fault. We refer to the shortest tests that detect a particular fault in a single cell as *elementary tests*; these tests are essential to the detection of the associated faults. Table 7.3 lists all the shortest tests that end in either a 'compare'

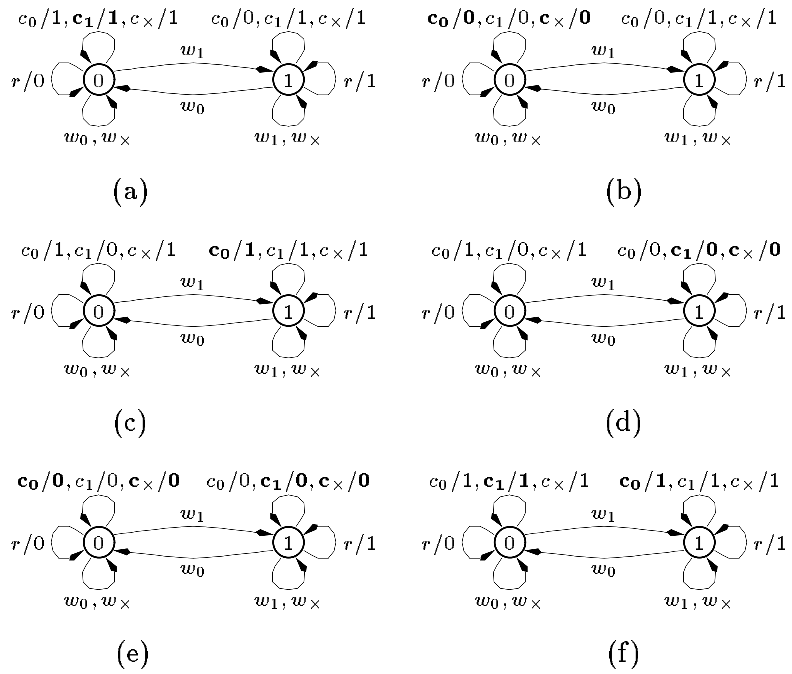


Figure 7.7: FSM models for faults: (a) *c-sa-0*, (b) *c-sa-1*, (c) $\bar{c}$*-sa-0*, (d) $\bar{c}$*-sa-1*, (e) *m-sa-0*, (f) *m-sa-1*.

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

The *w-sa-1* fault automaton is identical to that of the fault-free CAM cell; therefore,

[2] OBSERVER is a program for diagnosing and testing sequential machines. It is based on the theory developed in [10] and was written at the University of Waterloo by C.-J. Shi; additional features were later added by P. Kwiatkowski and P. R. Sidorowicz.

Table 7.3: Summary of input stuck-at faults.

Fault	Test Sequence	Faulty Response	Fault-Free Response
b-sa-0	$\mathbf{w_1 w_\times c_1}$	0	1
	$w_1 w_\times c_0$	1	0
	$w_1 r$	0	1
b-sa-1	$\mathbf{w_1 w_0 c_0}$	0	1
	$w_1 w_0 c_1$	1	0
	$w_1 w_0 r$	1	0
b-sa-0	$\mathbf{w_0 w_\times c_0}$	0	1
	$w_0 w_\times c_1$	1	0
	$w_0 r$	1	0
$\bar{b}$*-sa-1*	$\mathbf{w_0 w_1 c_1}$	0	1
	$w_0 w_1 c_0$	1	0
	$w_0 w_1 r$	0	1
w-sa-0	$\mathbf{w_1 c_1 w_0 c_0}$	1 0 or 0 1	1 1
	$w_0 c_0 w_1 c_1$	"	"
	$w_1 c_0 w_0 c_1$	"	0 0
	$w_0 c_1 w_1 c_0$	"	"
w-sa-1	see Section 8.2.1		
c-sa-0	$\mathbf{w_0 c_1}$	1	0
c-sa-1	$\mathbf{w_0 c_0}$	0	1
	$w_0 c_\times$	0	1
$\bar{c}$*-sa-0*	$\mathbf{w_1 c_0}$	1	0
$\bar{c}$*-sa-1*	$\mathbf{w_1 c_1}$	0	1
	$w_1 c_\times$	0	1
m-sa-0	$\mathbf{c_1 c_0}$	0 0	1 0 or 0 1
	$c_0 c_1$	"	"
	$c_\times$	0	1
m-sa-1	$\mathbf{c_1 c_0}$	1 1	1 0 or 0 1
	$c_0 c_1$	"	"

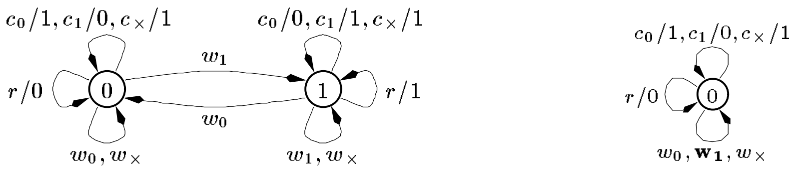
no test exists for a single cell. Consequently, this is the only input stuck-at fault that is not independently testable. However, this fault is detectable in conjunction with other words, and will be considered in Chapter 8.

7.3 State Stuck-at Faults

State stuck-at faults are stuck-at-0 and stuck-at-1 faults of nodes s and $\bar{s}$ in Fig. 7.1. Accordingly, we define the state stuck-at fault model to comprise these faults, under the single fault assumption. State stuck-at faults are often referred to in the literature as cell stuck-at faults. An event-sequence behavioral analysis has been performed for every state stuck-at fault; these analyses have been reported in [48] and are presented in Appendix A.1.2.

Example: 'Write 1' from state 0 in a correct cell and in the presence of the *s-sa-0* fault. Initial state: 0 11 00 1 $\cdot$ 0. First, $\bar{b}$ is lowered: 0 10 00 1 $\cdot$ 0. Then w is raised: 1 10 00 1 $\cdot$ 0. In the correct cell s changes: 1 10 00 1 $\cdot$ 1. In the faulty cell, this transition does not occur: 1 10 00 1 $\cdot$ **0**. Next, w is lowered: 0 10 00 1 $\cdot$ 1 (correct), 0 10 00 1 $\cdot$ **0** (faulty). Finally, both b and $\bar{b}$ are raised: 0 11 00 1 $\cdot$ 1 (correct), 0 11 00 1 $\cdot$ **0** (faulty).

FSM models for all the state stuck-at faults have been constructed, as shown in Fig. 7.9. The model associated with this example is shown in Fig. 7.8.



(a) (b)

Figure 7.8: Cell behavior: (a) correct, (b) *s-sa-0*.

(a) (b)

Figure 7.9: FSM models for faults: (a) *s-sa-0* and *$\bar{s}$-sa-1*, (b) *s-sa-1* and *$\bar{s}$-sa-0*.

One important characteristic of state stuck-at faults is that, although the cell can only be in one state, 'compare' and 'read' operations in that state generate determinate and correct output. We have generated elementary tests for every state stuck-at fault, using the OBSERVER program. We have determined that, in a single cell, every state stuck-at fault can be detected by tests of length 2. Examples of elementary tests for these faults in our fault model are shown in Table 7.4

Table 7.4: Summary of state stuck-at faults.

Fault	Test Sequence	Faulty Response	Fault-Free Response
s-sa-0	w_1c_1	0	1
s-sa-1	w_0c_0	0	1
$\bar{s}$*-sa-0*	w_0c_0	0	1
$\bar{s}$*-sa-1*	w_1c_1	0	1

7.4 Transistor On/Open Faults

Transistor faults constitute an important class of faults that occur in static memory cells. They are a result of defects that render transistors inoperable, in the sense that these transistors either always conduct or never conduct. These two types of transistor faults are known as *stuck-on* and *stuck-open* faults, respectively. Depending on the location of the faulty transistor, such faults may cause spontaneous changes of the cell's state that are slow enough not to be detected by typical testing algorithms and thus are classified as *data retention* faults. Transistor stuck-on faults are more readily testable, as they result in an increase in the quiescent supply current I_{DDQ}; hence they are detectable by parametric tests. This is not the case with transistor stuck-open faults; several testing methods for transistor stuck-open faults in SRAMs have been proposed [20, 37]. The *transistor fault model* comprises all the transistor stuck-on and stuck-open faults of the CAM cell, under the single fault assumption. An event-sequence behavioral analysis has been performed for every transistor fault; these analyses have been reported in [47] and are presented in Appendix A.1.3.

Table 7.5: A 'compare 1' operation in a correct and faulty CAM cell.

Compare 1 operation ($s = 1$)		
Description	Correct $w\ b\bar{b}\ c\bar{c}\ m{\cdot}s$	T_7-*on* $w\ b\bar{b}\ c\bar{c}\ m{\cdot}s$
initial state	0 11 00 1·1	0 11 00 1·1
float m	0 11 00 $\tilde{1}$·1	0 11 00 $\tilde{1}$·1
set $c/\bar{c}$	0 11 10 $\tilde{1}$·1	0 11 10 $\tilde{1}$·1
m discharges	0 11 10 $\tilde{1}$·1	0 11 10 **0**·1
read m & ground $c/\bar{c}$	0 11 00 $\tilde{1}$·1	0 11 00 **0**·1
raise m	0 11 00 1·1	0 11 00 1·1

Example: 'Compare 1' from state 1 in a correct cell (Fig. 7.1) and in the presence of the T_7-*on* fault. Initial state: 0 11 00 $1 \cdot 1$. First, m is floated: 0 11 00 $\tilde{1} \cdot 1$. Note that both transistors T_7 and T_8 in the faulty cell are on. Then c is raised: 0 11 10 $\tilde{1} \cdot 1$. Since in the correct cell T_7 is off, T_9 does not conduct and m remains floating: 0 11 10 $\tilde{1} \cdot 1$. In the faulty cell, the conducting transistors T_7 and T_8 act like a voltage divider, and hence, the voltage on the gate of T_9 will rise high enough to discharge m: 0 11 10 $\cdot 1$. Next, c is lowered: 0 11 00 $\tilde{1} \cdot 1$ (correct), 0 11 00 $\mathbf{0} \cdot 1$ (faulty). Finally, m is raised: 0 11 00 $1 \cdot 1$. These sequences are summarized in Table 7.5.

FSM models for all transistor faults have been constructed, as shown in Figs. 7.11 and 7.12. The model associated with this example is shown in Fig. 7.10.

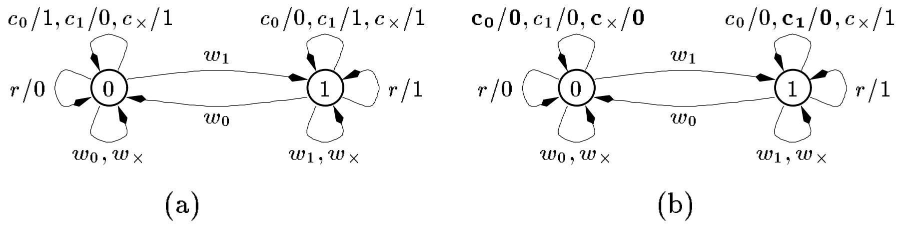


Figure 7.10: Cell behavior: (a) correct, (b) T_7-*on*.

As expected, faults affecting transistors T_1-T_4, cause data-retention failures, modeled but the indeterminate state I. Also, stuck-open faults of pass transistors T_5 and T_6 result in indeterminate output from 'read' operations.

In a single cell, every fault, except T_3-*on* and T_4-*on*, can be detected by tests of length at most 3. Examples of elementary tests for the reliably testable faults in our fault model are shown in Table 7.6. The results presented in this table indicate that transistor-open faults affect a cell's data retention capabilities are easily detectable by functional tests, due to the availability of the 'write $\times$' operation. However, in order to obtain full fault coverage with respect to this fault model, parametric tests are necessary.

Table 7.6: Summary of transistor faults.

Fault	Test Sequence	Faulty Response	Fault-Free Response
T_1-*on*	$w_1 w_\times c_1$	0	1
T_1-*open*	$w_0 w_\times c_0$	0	1
T_2-*on*	$w_0 w_\times c_0$	0	1
T_2-*open*	$w_1 w_\times c_1$	0	1
T_3-*on*	Not testable reliably. Parametric test required.		
T_3-*open*	$w_1 w_\times c_1$	0	1
T_4-*on*	Not testable reliably. Parametric test required.		
T_4-*open*	$w_0 w_\times c_0$	0	1
T_5-*on*	$w_0 c_0$	0	1
T_5-*open*	$w_0 w_1 c_1$	0	1
T_6-*on*	$w_1 c_1$	0	1
T_6-*open*	$w_1 w_0 c_0$	0	1
T_7-*on*	$w_1 c_1$	0	1
T_7-*open*	$w_0 c_1$	1	0
T_8-*on*	$w_0 c_0$	0	1
T_8-*open*	$w_1 c_0$	1	0
T_9-*on*	$w_1 c_1$	0	1
T_9-*open*	$c_1 c_0 / c_0 c_1$	11	10 or 01

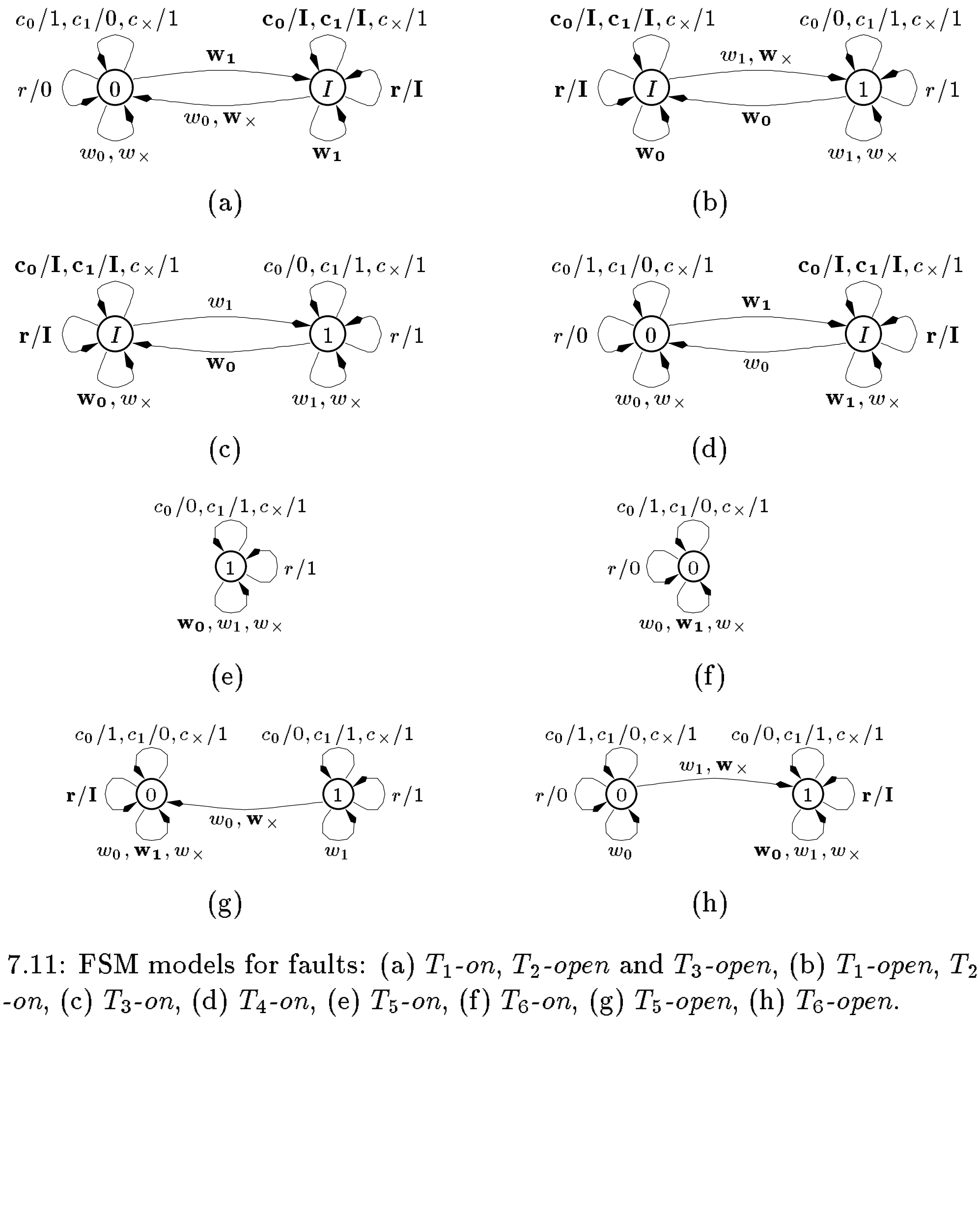


Figure 7.11: FSM models for faults: (a) T_1-*on*, T_2-*open* and T_3-*open*, (b) T_1-*open*, T_2-*on* and T_3-*on*, (c) T_3-*on*, (d) T_4-*on*, (e) T_5-*on*, (f) T_6-*on*, (g) T_5-*open*, (h) T_6-*open*.

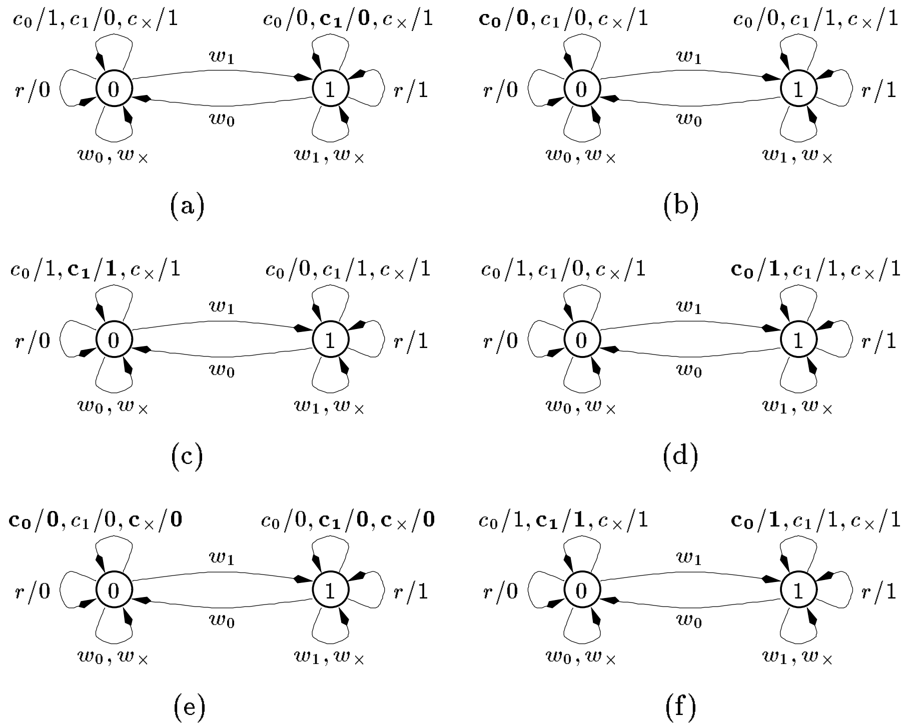


Figure 7.12: FSM models for faults: (a) T_7-*on*, (b) T_8-*on*, (c) T_7-*open*, (d) T_8-*open*, (e) T_9-*on*, (f) T_9-*open*.

7.5 Bridging Faults

Unintentional shorts between lines or nodes within modern VLSI circuits are a common occurrence. Such defects are of particular concern because depending on the resistance of a given short, its presence may or may not manifest itself as a logical fault. We consider two types of bridging faults: those that occur between input lines of a CAM cell, i.e., *intra-cell bridging faults* and those that occur between adjacent input lines of neighboring cells, i.e., *inter-cell bridging faults.* These faults can be further categorized into those that are a result of *non-resistive* (*hard*) shorts, and those due to *resistive* shorts. We define the bridging fault model for the CAM in order to establish which faults within the model are detectable reliably by functional tests. The bridging fault model comprises all the bridging faults between input lines of the CAM cell, as well as between adjacent lines of two neighboring cells, under the single fault assumption. An event-sequence behavioral analysis has been performed for every bridging fault; these analyses are presented in Appendix A.1.4.

7.5.1 Intra-cell Bridging Faults

Here we consider bridging faults that occur within a single cell.

Example: 'Compare 1' from state 0 in a correct cell and in the presence of the c-$\bar{c}$-hrd fault. Initial state: 0 11 00 $1 \cdot 0$. First, m is floated: 0 11 00 $\tilde{1} \cdot 0$. Then c is raised: 0 11 10 $\tilde{1} \cdot 0$ (correct). Since in the faulty cell c and $\bar{c}$ are shorted together, c remains grounded[3]: 0 11 **00** $\tilde{1} \cdot 0$ (faulty). In the correct cell $c = 1$ which switches on T_9 (T_7 conducts). This causes m to discharge: 0 11 10 $0 \cdot 0$ (correct). In the faulty cell $c = 0$, and hence, m will not discharge: 0 11 **00** $\tilde{\mathbf{1}} \cdot 0$ (faulty). Next, m is read and c is lowered: 0 11 00 $0 \cdot 0$ (correct), 0 11 00 $\tilde{\mathbf{1}} \cdot 0$ (faulty). Finally, m is raised: 0 11 00 $1 \cdot 0$. Table 7.7 lists events that occur during a 'compare 1' operation in a correct and a faulty cell when $s = 0$.

Table 7.7: A 'compare 1' operation in a correct and faulty CAM cell.

Compare 1 operation ($s = 0$)		
Description	Correct w $b\bar{b}$ $c\bar{c}$ $m{\cdot}s$	c-$\bar{c}$-hrd w $b\bar{b}$ $c\bar{c}$ $m{\cdot}s$
initial state	0 11 00 1·0	0 11 00 1·0
float m	0 11 00 $\tilde{1}$·0	0 11 00 $\tilde{1}$·0
set $c/\bar{c}$	0 11 10 $\tilde{1}$·0	0 11 **00** $\tilde{1}$·0
m discharges	0 11 10 0·0	0 11 **00** $\tilde{\mathbf{1}}$·0
read m & ground $c/\bar{c}$	0 11 00 0·0	0 11 00 $\tilde{\mathbf{1}}$·0
raise m	0 11 00 1·0	0 11 00 1·0

The FSM model associated with this example is shown in Fig. 7.13.

[3]Pull-down circuitry dominates pull-up circuitry.

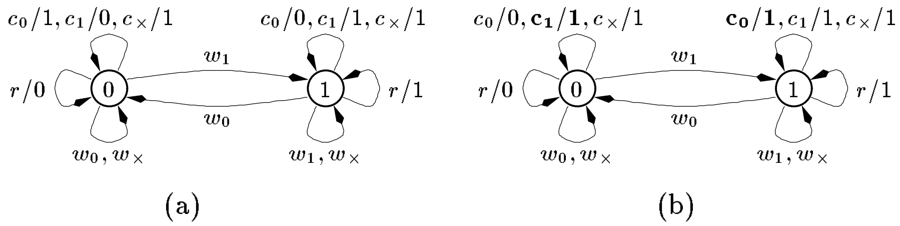


Figure 7.13: Cell behavior: (a) correct, (b) *c-$\bar{c}$-hrd*.

7.5.2 Inter-cell Bridging Faults

We now consider bridging faults between adjacent lines of two neighboring cells, i.e., the *$\bar{b}'$-b-hrd*, *$\bar{b}'$-b-res*, *$\bar{c}'$-c-hrd* and *$\bar{c}'$-c-res* faults. Since, in this CAM circuit layout, $b/\bar{b}$ are implemented on a different layer of metal than $c/\bar{c}$ lines, faults *$\bar{c}'$-b-hrd*, *$\bar{c}'$-b-res* *$\bar{b}'$-c-hrd* and *$\bar{b}'$-c-res* are unlikely to occur, and hence these faults are not being considered here.

Example: 'Compare $\times$1' from state 00 in a correct cell pair and in the presence of the *$\bar{c}'$-c-hrd* fault. Initial state of the cell pair: 0 11 00 11 00 $1 \cdot 00$. First, m is floated: 0 11 00 11 00 $\tilde{1} \cdot 00$. Then c is raised: 0 11 00 11 10 $\tilde{1} \cdot 00$ (correct). In a faulty cell pair c remains at 0 as it is grounded by $\bar{c}'$: 0 11 00 11 **00** $\tilde{1} \cdot 00$ (faulty). In the correct cell pair m to discharges: 0 11 00 11 10 $0 \cdot 00$ (correct), and in the faulty one it does not: 0 11 00 11 **00** $\tilde{\mathbf{1}} \cdot 00$ (faulty). Next, m is read and c is lowered: 0 11 00 11 00 $0 \cdot 00$ (correct), 0 11 00 11 00 $\tilde{\mathbf{1}} \cdot 00$ (faulty). Finally, m is raised: 0 11 00 11 00 $1 \cdot 00$. Table 7.8 lists events that occur during a 'compare 1' operation in a correct and a faulty cell when $s = 0$.

The FSM model associated with this example is shown in Fig. 7.14.

FSM models for the remaining intra- and inter-cell bridging faults are shown in Figs. 7.15 and 7.16, respectively. In Fig. 7.16 all compare operations have been omitted for clarity, as their output from states 00, 01, 10 and 11 is correct (see Fig. 7.14(a)).

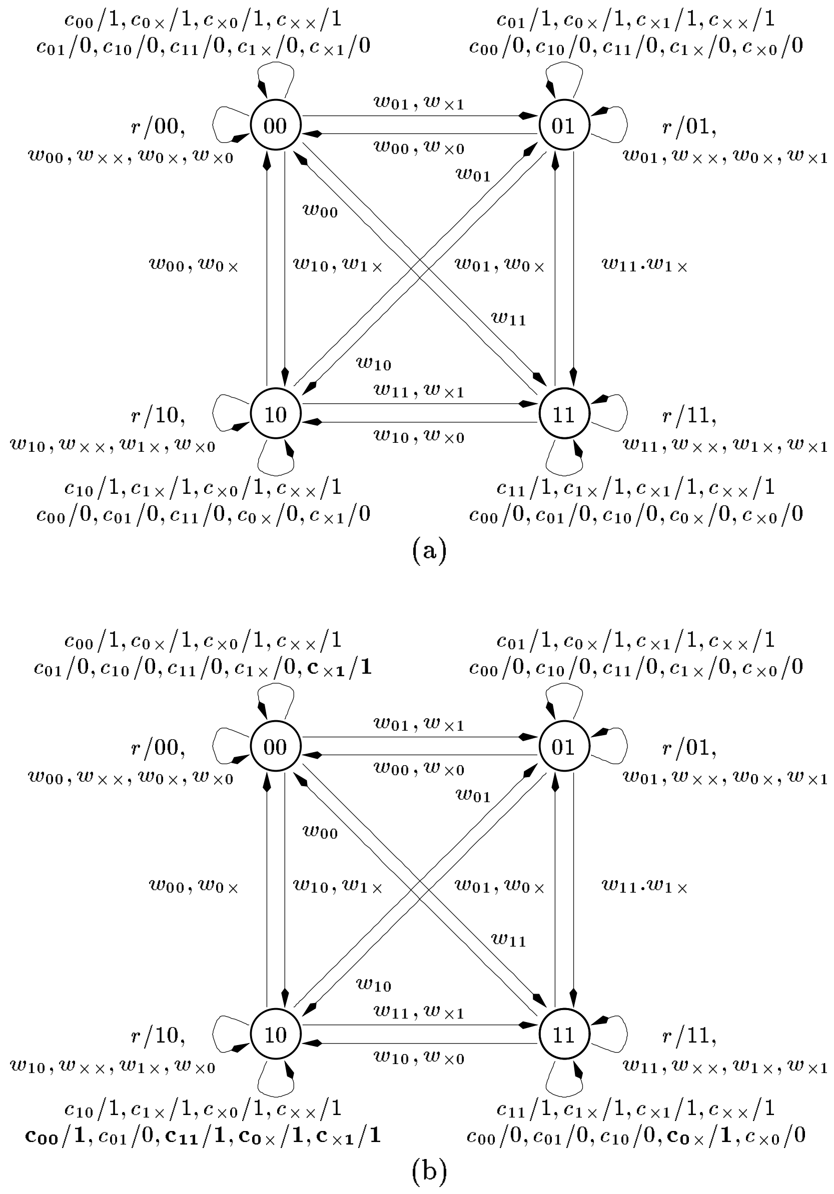


Figure 7.14: Behavior of two adjacent CAM cells: (a) correct, (b) $\vec{c}'$-*c-hrd*.

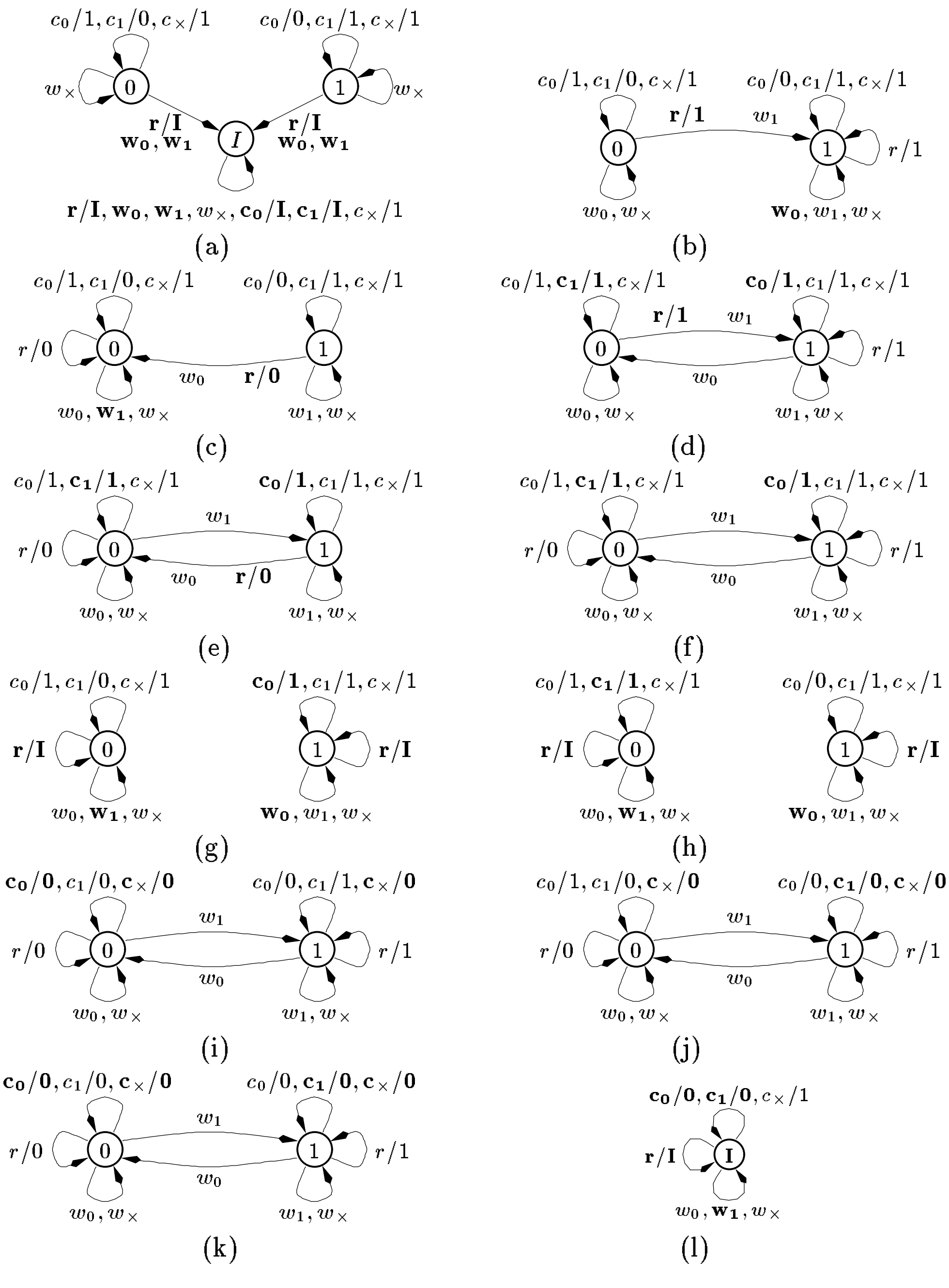


Figure 7.15: FSM models for faults: (a) *b-$\bar{b}$-hrd*, (b) *b-w-hrd*, (c) *$\bar{b}$-hrd*, (d) *b-m-hrd*, (e) *$\bar{b}$-m-hrd*, (f) *c-$\bar{c}$-hrd*, (g) *c-w-hrd*, (h) *$\bar{c}$-w-hrd*, (i) *c-m-hrd* and *c-m-res*, (j) *$\bar{c}$-m-hrd* and *$\bar{c}$-m-res*, (k) *m-w-hrd* and *m-w-res*, (l) *s-$\bar{s}$-hrd*.

Table 7.8: A 'compare $\times 1$' operation in a correct and faulty CAM cell pair.

Compare $\times 1$ operation ($S = 00$)												
	Correct						$\bar{c}'$-c-*hrd*					
Description	w	$b'\bar{b}'$	$c'\bar{c}'$	$b\bar{b}$	$c\bar{c}$	$m{\cdot}S$	w	$b'\bar{b}'$	$c'\bar{c}'$	$b\bar{b}$	$c\bar{c}$	$m{\cdot}S$
initial state	0	11	00	11	00	1·00	0	11	00	11	00	1·00
float m	0	11	00	11	00	$\tilde{1}$·00	0	11	00	11	00	$\tilde{1}$·00
set $c/\bar{c}$	0	11	00	11	10	$\tilde{1}$·00	0	11	00	11	$\mathbf{0}$0	$\tilde{1}$·00
m discharges	0	11	00	11	10	0·00	0	11	00	11	$\mathbf{0}$0	$\tilde{\mathbf{1}}$·00
read m & ground $c/\bar{c}$	0	11	00	11	00	0·00	0	11	00	11	00	$\tilde{\mathbf{1}}$·00
raise m	0	11	00	11	00	1·00	0	11	00	11	00	1·00

The output of 'compare' and 'read' operations from the indeterminate states $I0$, $0I$, $1I$ and $I1$ is indeterminate.

In a single cell, every hard bridging fault, except b-$\bar{b}$-*hrd*, $\bar{c}$-w-*hrd* and c-w-*hrd* can be detected by tests of length at most 3. In an adjacent cell pair, both inter-cell hard bridging faults can also be detected by tests of length at most 3.

Resistive bridging faults, on the other hand, are not as easily testable. Only c-m-*res*, $\bar{c}$-m-*res* and m-w-*res* are detectable by functional tests. The remaining faults do not alter the functional behavior of the cell, however they do increase I_{DD} or I_{DDQ} and, hence, are readily detectable by parametric tests.

Examples of elementary tests for the reliably testable faults in our fault model are shown in Table 7.9.

7.6 Summary

Using a particular static CMOS CAM as an example, we have developed a fault modeling methodology. The modeling steps include circuit analysis, asynchronous behavior analysis, and FSM representation of the fault-free and faulty circuits. Four fault models for an n-word by l-bit static CMOS CAM have been defined: input stuck-at, state stuck-at, transistor stuck-(on/open), and bridging. It has been found that some of the faults

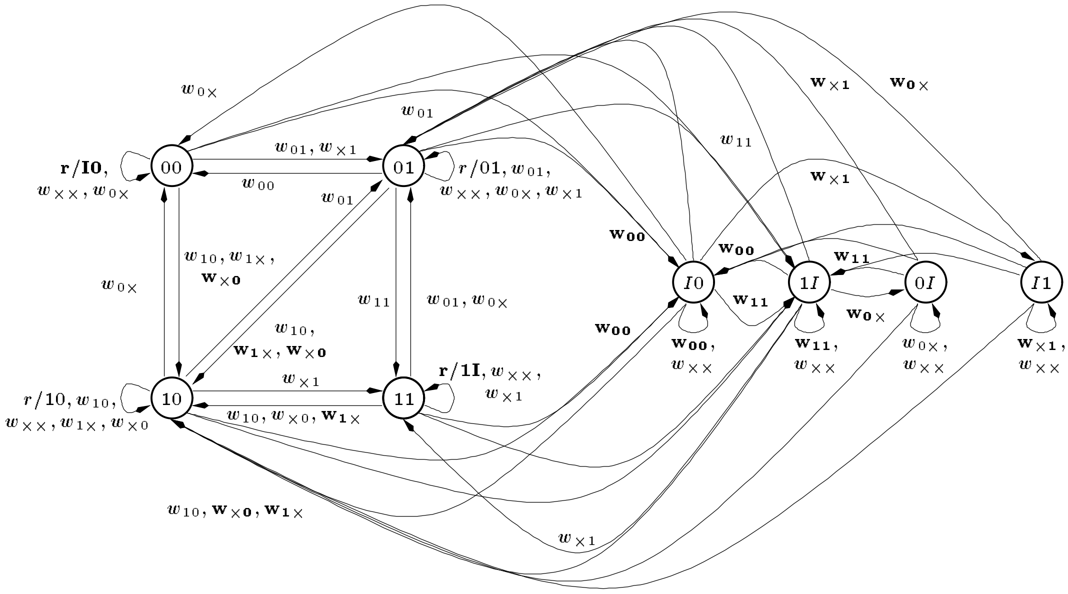


Figure 7.16: FSM model: inter-cell $\bar{b}'$-b-hrd bridging fault.

described resemble well known fault categories: cell stuck-at, transition, coupling, etc. However, many of the faults presented here have a distinct behavior and do not cleanly fit these categories. A systematic behavioral analysis produced the following results:

- All input and state stuck-at faults are reliably detectable by functional tests, although some faults are not independently testable and have to be considered in conjunction with other cells, as shown in Chapter 8.

- $\frac{2nl}{18} \times 100\%$ (approx. 11%) of transistor faults are not detectable by functional CAM tests; however all transistor-open faults that compromise data retention in static memory cells can be reliably detected, due to the enhanced functionality of this type of memory.

- $\frac{9nl+7l-2}{18nl+8l+2n-4} \times 100\%$ (approx. 50%) of faults in the bridging model are not reliably detectable by functional tests.

Table 7.9: Summary of intra- and inter-cell bridging faults.

Fault	Test Sequence	Faulty Response	Fault-Free Response
b-barb-hrd	Not testable reliably. Parametric test required.		
b-w-hrd	$w_1 w_0 c_0$	0	1
$\bar{b}$*-w-hrd*	$w_0 w_1 c_1$	0	1
b-m-hrd	$w_0 c_1$	1	0
$\bar{b}$*-m-hrd*	$w_0 c_1$	1	0
c-$\bar{c}$*-hrd*	$w_0 c_1$	1	0
c-w-hrd	Not testable reliably. Parametric test required.		
$\bar{c}$*-w-hrd*	Not testable reliably. Parametric test required.		
c-m-hrd	$w_0 c_0$	0	1
$\bar{c}$*-m-hrd*	$w_1 c_1$	0	1
m-w-hrd	$w_1 c_1$	0	1
s-$\bar{s}$*-hrd*	$c_1 c_0$	00	01 or 10
b-barb-res	Not testable reliably. Parametric test required.		
b-w-res	Not testable reliably. Parametric test required.		
$\bar{b}$*-w-res*	Not testable reliably. Parametric test required.		
b-m-res	Not testable reliably. Parametric test required.		
$\bar{b}$*-m-res*	Not testable reliably. Parametric test required.		
c-$\bar{c}$*-res*	Not testable reliably. Parametric test required.		
c-w-res	Not testable reliably. Parametric test required.		
$\bar{c}$*-w-res*	Not testable reliably. Parametric test required.		
c-m-res	$w_0 c_0$	0	1
$\bar{c}$*-m-res*	$w_0 w_1 c_1$	0	1
m-w-res	$w_1 c_1$	0	1
s-$\bar{s}$*-res*	Not testable reliably. Parametric test required.		
$\bar{b}'$*-b-hrd*	$w_{00} w_{\times 0} c_{00}$	0	1
$\bar{c}'$*-c-hrd*	$w_{11} c_{0\times}$	1	0
$\bar{b}'$*-b-res*	Not testable reliably. Parametric test required.		
$\bar{c}'$*-c-res*	Not testable reliably. Parametric test required.		

- Faults that are not detectable by functional tests cause an increase of I_{DD} or I_{DDQ}; hence they are detectable through parametric testing.
- 'Read' operations are inadequate sources of output for testing static CMOS CAMs; 'compare' operations are a more reliable choice.
- Elementary tests, of length at most 3, have been determined; these tests are necessary to detect reliably testable faults in an arbitrary CAM cell.

Chapter 8

Development of a CAM Test

In the previous chapter we have developed four fault models under the single fault assumption. We have found that not all faults are reliably detectable by functional tests; however, for those that are, elementary tests have been generated. We have also found faults that can only be detected in conjunction with tests on other cells. Table 8.1 lists all the necessary elementary tests to detect the reliably testable faults. In this chapter we develop an overall test that will detect these faults in an n-word by l-bit CAM array. This process consists of four steps:

1. A test for a single cell is developed,
2. This test is extended to an n-word by 1-bit CAM (bit-oriented memory),
3. It then is extended to a 1-word by l-bit CAM (single l-bit word),
4. On the basis of the two extensions, a test for an n-word by l-bit CAM is generated.

Table 8.1: Summary of elementary tests for a CAM.

Elementary Test	Faults Detected
$w_1 w_\times c_1$	*b-sa-0*, T_1*-on*, T_2*-open*, T_3*-open*
$w_1 w_0 c_0$	*b-sa-1*, T_6*-open*, *b-w-hrd*
$w_0 w_\times c_0$	$\bar{b}$*-sa-0*, T_1*-open*, T_2*-on*, T_4*-open*
$w_0 w_1 c_1$	$\bar{b}$*-sa-1*, T_5*-open*, $\bar{b}$*-w-hrd*, $\bar{c}$*-m-res*
$w_1 c_1 w_0 c_0$	*w-sa-0*
$w_0 c_1$	*c-sa-0*, T_7*-open*, *b-m-hrd* $\bar{b}$*-m-hrd*, *c-*$\bar{c}$*-hrd*
$w_0 c_0$	*c-sa-1*, *s-sa-1*, $\bar{s}$*-sa-0*, T_5*-on*, T_8*-on*, *c-m-hrd*, *c-m-res*
$w_1 c_0$	$\bar{c}$*-sa-0*, T_8*-open*,
$w_1 c_1$	$\bar{c}$*-sa-1*, *s-sa-0*, $\bar{s}$*-sa-1*, T_6*-on*, T_7*-on*, T_9*-on*, $\bar{c}$*-m-hrd*, *m-w-hrd*, *m-w-res*
$c_1 c_0$	*m-sa-0*, *m-sa-1*, T_9*-open*, *s-*$\bar{s}$*-hrd*
$w_{00} w_{\times 0} c_{00}$	$\bar{b}'$*-b-hrd*
$w_{11} c_{0\times}$	$\bar{c}'$*-c-hrd*

8.1 Test for a Single Cell

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

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

and

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

of length 9 that detect all independently testable faults. It has been shown that for every independently testable fault, there exists a test that is included in T_{cell} (T'_{cell}). The details are presented in Appendix A.2. Moreover, these two tests are *irredundant*, since the removal of any of the input symbols in the test will result in some fault becoming undetectable. Tests for individual faults that are included in T_{cell} are indicated in Table 8.1. With the exception of tests $w_1 c_0$ and $w_0 c_1$, these tests have been chosen because their response in a fault-free cell is a match, and in a faulty cell, a mismatch. Thus, the CAM's implicit-AND match property, i.e., that a match line discharges in the presence of even a single bit mismatch, can be efficiently utilized to test cells in parallel.

8.2 Extension to n-word by 1-bit CAM

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

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

2. Match detection is done through a *hit* line (high when at least one match is detected), a *multi-hit* line (high when multiple matches are detected), and an encoder that returns the address k of the highest priority match line[1].

We model the correct behavior of an n-word by 1-bit word CAM as an FSM

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

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

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

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

and

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

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

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

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

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

8.2.1 Testing the *w-sa-1* fault

As indicated in Section 7.2, *w-sa-1* is not independently testable and affects all the cells along the faulty w. However, this fault is detectable in the following manner:

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

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

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

8.2.2 Complete Test for n-word by 1-bit CAM

To achieve 100% fault coverage under our fault model, we combine $T_{n\text{-}bit}$ with $T_{w\text{-}sa\text{-}1}$:

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

We remind the reader that we are using the single-fault assumption. Note that the initial $(w_0^i)^{\updownarrow}$ of the $T_{w\text{-}sa\text{-}1}$ test has been dropped, as the CAM is expected to hold only

0s at the end of $T_{n\text{-}bit}$. Note also that $w_\times$ is not needed in the last part of the test, because the faults which require $w_\times$ would have been detected by the first part of the test.

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

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

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

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

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

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

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

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

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

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

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

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

and

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

All faults, except for c^j*-sa-0*, $\bar{c}^j$*-sa-0*, T_7^j*-open*, T_8^j*-open*, b^j*-m-hrd*, $\bar{b}^j$*-m-hrd*, c^j-$\bar{c}^j$-*hrd* and $\bar{c}^{(j+1)}$-c^j*-hrd* are detectable by performing the respective tests on all cells in parallel since these faults manifest themselves with a mismatch. Detection of each of the remaining faults, where $1 \leq j \leq l$, is more complex. Their respective tests have a mismatch as a fault-free response and a match as a faulty one; thus, they cannot be applied to all the cells in parallel, as no faulty response would ever be propagated along m. To propagate a faulty response along m, a mismatch must be attempted on each cell j individually, while simultaneously applying a $c_\times$ to the remaining $l-1$ cells. Let $\times^{j-1}0\times^{l-j}$ be the l-bit word with 0 in position j and $\times$ in every other position. The word $\times^{j-1}1\times^{l-j}$ is similarly defined. The symbol $[c_{\times^{j-1}0\times^{l-j}}]^{\leftrightarrow}$ represents a sequence of l 'compare' operations to words of the form $\times^{j-1}0\times^{l-j}$, where j varies from 1 to l or from l to 1. In this manner one c_0 and $l-1$ $c_\times$ are performed on each cell. Also, each pair of cells undergoes a $c_{0\times}$, necessary for the detection of the $\bar{c}^{(j+1)}$-c^j*-hrd* faults.

The inter-cell fault $\bar{c}^{(j+1)}$-c^j*-hrd* requires a special pattern to be used during a 'write' operation. We address this issue by inserting a $w_{\times 0\ldots\times 0}c_{0\ldots 0}$ followed by a $w_{0\times\ldots 0\times}c_{0\ldots 0}$

to detect all possible $\bar{c}^{(j+1)}$-c^j-*hrd* faults.

The extension of the T_{cell} test to the 1-word by l-bit CAM takes the form

$$\begin{aligned} T_{word} \;=\;& w_{0\ldots0}w_{1\ldots1}w_{\times\ldots\times}c_{1\ldots1}[c_{\times^{j-1}0\times^{l-j}}]^{\leftrightarrow} \\ & w_{0\ldots0}w_{\times\ldots\times}c_{0\ldots0}[c_{\times^{j-1}1\times^{l-j}}]^{\leftrightarrow} \\ & (w_{\times0\ldots\times0}c_{0\ldots0}w_{0\times\ldots0\times}c_{0\ldots0}), \end{aligned}$$

where $w_{0\ldots0}$ denotes the writing of an all-0 word, $w_{\times0\ldots\times0}$ denotes writing 0s to odd bits within a word and masking the even bits, etc.

The T_{word} test consists of 7 'write' operations and $2l+4$ 'compare' operations and has an overall length of $2l+11$.

8.4 Extension to n-word by l-bit CAM

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

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

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

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

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

and

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

The following test detects all reliably detectable faults under the input stuck-at, state stuck-at, transistor stuck-(on/open) and bridging fault models:

$$\begin{aligned} T_{S\&B} \;=\; & (w^i_{0\ldots0})^{\updownarrow} \\ & (w^i_{1\ldots1} w^i_{\times\ldots\times} c_{1\ldots1})^{n\downarrow1} [c_{\times^{j-1}0\times^{l-j}}]^{\leftrightarrow} \\ & (w^i_{0\ldots0} w^i_{\times\ldots\times} c_{0\ldots0})^{n\downarrow1} [c_{\times^{j-1}1\times^{l-j}}]^{\leftrightarrow} \\ & (w^n_{\times0\ldots\times0} c_{0\ldots0} w^n_{0\times\ldots0\times} c_{0\ldots0}) \\ & (w^n_{1\ldots1} c_{1\ldots1} w^n_{0\ldots0} w^u_{1\ldots1} c_{1\ldots1}). \end{aligned}$$

This test consists of two parts, analogous to those of the $T_{n\text{-}bit\text{-}compl}$ test. The first part consists of an initialization which sets all words to $0\ldots0$, and two march elements. For a fault-free CAM, each of the march elements should result in n hits with $k = i$, followed by l mismatches at $k = 0$. Any other response indicates a fault. In the second part (last two lines), two hits are expected with $k = 1$, after which multiple hits are monitored, since they indicate faults.

The $T_{S\&B}$ test consists of $5n+5$ 'write' operations and $2n+2l+4$ 'compare' operations. Therefore, the length of our test for all the reliably detectable faults in an n-word by l-bit CAM is $7n + 2l + 9$.

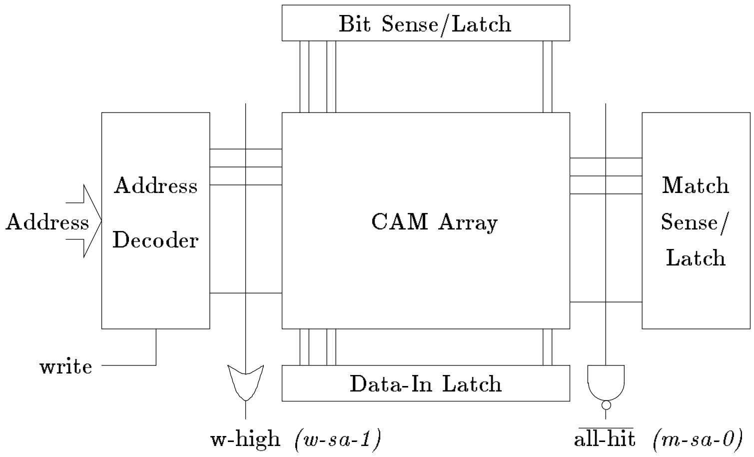


Figure 8.1: DFT suggestions.

8.5 DFT Suggestions

The length of the $T_{S\&B}$ test is linear in the number of bits in the CAM. Certain DFT hardware enhancements, if implementable[2], can substantially reduce this test length. Depicted as logic gates in Figure 8.1 they target faults that contribute most to the length of the test; these enhancements are described below.

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

$$
\begin{aligned}
T'_{S\&B} \;=\; & (w^i_{0\ldots0})^{\updownarrow} \\
& (w^i_{1\ldots1} w^i_{\times\ldots\times})^{n\downarrow1} c_{1\ldots1} [c_{\times^{j-1}0\times^{l-j}}]^{\leftrightarrow} \\
& (w^i_{0\ldots0} w^i_{\times\ldots\times})^{n\downarrow1} c_{0\ldots0} [c_{\times^{j-1}1\times^{l-j}}]^{\leftrightarrow} \\
& (w^n_{\times0\ldots\times0} c_{0\ldots0} w^n_{0\times\ldots0\times} c_{0\ldots0}) \\
& (w^n_{1\ldots1} c_{1\ldots1} w^n_{0\ldots0} w^u_{1\ldots1} c_{1\ldots1}),
\end{aligned}
$$

[2]The cost of DFT must be acceptable, i.e., additional area, complexity, performance degradation, etc.

the length of which is reduced to $5n + 2l + 11$.

- A *w-high* output, indicating that at least one write line is active. This output can directly detect any *w-sa-1* fault, as well as *w-sa-0* faults during 'write' operations; hence, the suffix of the test $T'_{S\&B}$, $(w^n_{1\ldots1}c_{1\ldots1}w^n_{0\ldots0}w^u_{1\ldots1}c_{1\ldots1})$, can be omitted. If, however, the *w-high* output is unavailable, the *w-sa-1* fault can still be detected by reducing the above-mentioned suffix to $([\bar{b}^l = 0\ldots0]c_{1\ldots1})$, where $[\bar{b}^l = 0\ldots0]$ represents setting all the $\overline{bit}$ lines to ground — like the first step in the 'write 0' operation — without actually raising any of the *WL*s. This will result in the word affected by the *w-sa-1* fault having $1\ldots1$ written into it. A match resulting from a subsequent $c_{1\ldots1}$ operation indicates a presence of this fault.

- The ability to perform a *bulk-write* operation to all words in the memory simultaneously, together with the $\overline{\textit{all-hit}}$ output, will result in a test length that is independent of the number of words in the CAM. For example, the length of $T'_{S\&B}$ would be reduced to $2l + 16$. If used with the *w-high* output, the length of the test would be further reduced to $2l + 11$

8.6 Summary

We have presented the construction of the $T_{S\&B}$ test which detects all reliably testable faults under the input stuck-at, state stuck-at, transistor stuck-(on/open) and bridging fault models. The construction process consists of the development of a test for a single cell, extension of that test first to an n-word by 1-bit CAM (bit-oriented memory), then to a 1-word by l-bit CAM (single l-bit word), and by combining the two, the generation of a linear-time test for an n-word by l-bit CAM. In addition, some DFT enhancements that reduce the test length have been suggested.

Chapter 9

Evaluation of CAM Tests

Several algorithms for testing CAMs have been reported [19, 31, 33, 34, 48]. The tests in [19, 31, 33, 48] are applicable to CAM designs which incorporate an explicit word addressing scheme of the type found in conventional RAMs. The test of [34] imposes extra requirements on the addressing scheme which are not available in most CAMs. It also assumes that 'read' operations do not affect the state of a cell; this assumption is violated by some faults in our fault model. For these reasons we exclude this test from further consideration.

In this chapter we show how our framework can be used to determine fault coverage with respect to a particular fault model. We will illustrate this by establishing the fault coverage of the tests reported in [19] and [31] with respect to the *input stuck-at* fault model of the CAM circuit of Fig. 7.1, under the single-fault assumption. The process of establishing the fault coverage, previously reported in [49], consists of three steps:

1. representation of the test from the perspective of an arbitrary single cell in the array,
2. verification of the existence of elementary tests in the above representation,

3. determination of detection of non-independently testable faults.

9.1 Giles and Hunter Test

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

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

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

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

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

$$
\begin{aligned}
T^{i,j}_{G\&H} \quad = \quad & w^{i,j}_{((i-1)\,\mathrm{div}\,2^j)\bmod 2}\,[c^j_{(p\,\mathrm{div}\,2^j)\bmod 2}]^0_{p=n-1} && (9.1)\\
& w^{i,j}_{1-(((i-1)\,\mathrm{div}\,2^j)\bmod 2)}\,[c^j_{1-((p\,\mathrm{div}\,2^j)\bmod 2)}]^0_{p=n-1} && (9.2)\\
& w^{i,j}_{((i-1)\,\mathrm{div}\,2^j)\bmod 2}\,[c^j_{(p\,\mathrm{div}\,2^j)\bmod 2}]^{n-1}_{p=0} && (9.3)\\
& w^{i,j}_0\,[c^j_{I[p,j]}]^{l-1}_{p=0} && (9.4)\\
& w^{i,j}_1\,[c^j_{1-I[p,j]}]^{l-1}_{p=0} && (9.5)
\end{aligned}
$$

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

We have shown in Section 7.2 that 'compare' operations, even when faulty, do not affect the state of the cell, so interleaving 'write' operations with arbitrary number of 'compare' operations will not influence any state transitions resulting from 'write' operations. Each cell (i,j), therefore, is subject to one of the following two sequences of write operations: $w_0w_1w_0w_0w_1$ or $w_1w_0w_1w_0w_1$. In the case of $n = 2^l$, each of the initial three 'write' operations are followed by $n/2$ 'compare 0' operations and $n/2$ 'compare 1' operations. The order of 'compare' operations depends on the column j in which the given cell is located, but can be treated as arbitrary. If $n < 2^l$, in the worst case, each of the initial three 'write' operations will only be followed by a sequence of "matching" 'compare' operations, i.e., a $w^{i,j}_0$ would precede a sequence of c^j_0's or a $w^{i,j}_1$ would precede a sequence of c^j_1's. The last two 'write' operations are always followed by $l-1$ "mismatching" 'compare' operations, where a w_0 precedes a c_1 and vice-versa, and a single "matching" 'compare' operation, for all possible values of n and l. The order of occurrence of the single "matching" 'compare' operation depends on the column j in which the given cell is located.

We now verify the presence of elementary tests for input stuck-at faults:

b-sa-0 According to Table 7.3, a test that detects this fault is $w_1 w_\times c_1$. The sequence $w_1^{i,j} w_\times^{i,j}$ does not occur in $T_{G\&H}^{i,j}$. Since after the initial $w_1^{i,j}$ the cell finds itself in an indeterminate state, none of the subsequent 'compare' operations can result in a dependable output.

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

b-sa-1 According to Table 7.3, a test that detects this fault is $w_1 w_0 c_0$. The sequence $w_1^{i,j} w_0^{i,j}$ occurs during the execution of (9.1) and (9.2), or (9.2) and (9.3) in $T_{G\&H}^{i,j}$. Each $w_0^{i,j}$ is guaranteed to be followed by at least one c_0^j.

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

w-sa-0 According to Table 7.3, a test that detects this fault is $w_1 c_1 w_0 c_0$. The sequence $w_1^{i,j} w_0^{i,j}$ occurs during the execution of (9.2) and (9.3), or (9.3) and (9.4) in $T_{G\&H}^{i,j}$. Each $w_1^{i,j}$ is guaranteed to be followed by at least one c_1^j; each $w_0^{i,j}$ is guaranteed to be followed by at least one c_0^j.

c-sa-0 According to Table 7.3, a test that detects this fault is $w_0 c_1$. This test occurs in (9.4) of $T_{G\&H}^{i,j}$, as the $w_0^{i,j}$ is guaranteed to be followed by a c_1^j.

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

c-sa-1 According to Table 7.3, a test that detects this fault is $w_0 c_0$. A $w_0^{i,j}$ occurs in either (9.1) or (9.2) of $T_{G\&H}^{i,j}$. Each $w_0^{i,j}$ is guaranteed to be followed by at least one c_0^j.

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

m-sa-0* and *m-sa-1 Assume that there are no duplicate words in the array. According to Table 7.3, tests that detect these faults are $c_1 c_0$ or $c_0 c_1$. Either $c_1^j c_0^j$ or $c_0^j c_1^j$ occurs in (9.4) or (9.5) of $T_{G\&H}^{i,j}$ after $w_0^{i,j}$ or $w_1^{i,j}$, respectively. In the presence of a duplicate word, however, any *m-sa-0* fault would be masked by a match on the fault-free duplicate.

We now focus on the non-independently testable faults:

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

The above analysis of the $T_{G\&H}$ test shows that, in the cell of Fig. 7.1, $T_{G\&H}$ does not reliably detect the *b-sa-0* and *$\bar{b}$-sa-0* faults. There are $2l$ *b-sa-0* and *$\bar{b}$-sa-0* faults in an n-word by l-bit CAM. Since there are $4n + 8l$ possible single faults in our fault model, $T_{G\&H}$ detects only $(1 - \frac{l}{2n+4l}) \cdot 100\%$ of faults in the input stuck-at fault model. For example [31], for $n = 32$, $l = 29$ only 83.89% of faults are detected.

We can modify the $T_{G\&H}$ test to achieve 100% fault coverage, under the assumption that $n \leq 2^l$. Since in the presence of the *b-sa-0* and *$\bar{b}$-sa-0* faults, a $w_\times$ forces the cell to a determinate but erroneous state, judicious insertion of $w_\times$ into $T_{G\&H}$ will assure detection of these faults. This augmented test, presented below, has length $11n + 2l$.

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

9.2 Kornachuk et al. Test

$T_{C\text{-}SM}$ presented in [31] can be represented using our notation. In this case, 'read', 'write' and 'compare' operations are performed on entire words. The subscripts of 'write' and 'compare' operations represent the decimal equivalent of an l-bit binary number. The subscripts of 'read' operations represent the expected output of the operation. In this case, the subscript '$-$' indicates an arbitrary decimal value read or used as a comparison key during the initialization step of the test. The concurrent 'compare' operations are listed directly below the 'read' and 'write' operations.

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

Since multiple matches can be expected, a priority encoder is used to produce the highest address where a match was obtained. We can represent $T_{C\text{-}SM}$ from the perspective of an arbitrary cell located at coordinates (i, j) in the array where $1 \leq i \leq n$ and $0 \leq j \leq l-1$. The subscripts of 'write' and 'compare' operations represent single bit values, where '$-$' stands for an arbitrary value used during initialization. The subscripts of 'read' operations denote the expected output of the operation.

$$T^{i,j}_{C\text{-}SM} = \begin{bmatrix} r^{i,j}_{-} w^{i,j}_{0} \\ c^{j}_{-}\ c^{j}_{-} \end{bmatrix}^{((l-1)-j)} \begin{bmatrix} r^{i,j}_{-} w^{i,j}_{0} \\ c^{j}_{-}\ c^{j}_{-} \end{bmatrix} \begin{bmatrix} r^{i,j}_{0} w^{i,j}_{0} \\ c^{j}_{0}\ c^{j}_{0} \end{bmatrix}^{(j+1)} \tag{9.6}$$

$$\begin{bmatrix} r^{i,j}_{0} w^{i,j}_{0} \\ c^{j}_{0}\ c^{j}_{0} \end{bmatrix}^{((l-1)-j)} \begin{bmatrix} r^{i,j}_{0} w^{i,j}_{1} \\ c^{j}_{1}\ c^{j}_{1} \end{bmatrix} \begin{bmatrix} r^{i,j}_{1} w^{i,j}_{1} \\ c^{j}_{1}\ c^{j}_{1} \end{bmatrix}^{(j+1)} \tag{9.7}$$

$$\begin{bmatrix} r^{i,j}_{1} w^{i,j}_{1} \\ c^{j}_{1}\ c^{j}_{1} \end{bmatrix}^{((l-1)-j)} \begin{bmatrix} r^{i,j}_{1} w^{i,j}_{0} \\ c^{j}_{0}\ c^{j}_{0} \end{bmatrix} \begin{bmatrix} r^{i,j}_{0} w^{i,j}_{0} \\ c^{j}_{0}\ c^{j}_{0} \end{bmatrix}^{(j+1)} \tag{9.8}$$

$$\begin{bmatrix} r^{i,j}_{0} w^{i,j}_{0} \\ c^{j}_{0}\ c^{j}_{0} \end{bmatrix}^{((l-1)-j)} \begin{bmatrix} r^{i,j}_{0} w^{i,j}_{1} \\ c^{j}_{1}\ c^{j}_{1} \end{bmatrix} \begin{bmatrix} r^{i,j}_{1} w^{i,j}_{1} \\ c^{j}_{1}\ c^{j}_{1} \end{bmatrix}^{(j+1)} \tag{9.9}$$

$$\begin{bmatrix} r^{i,j}_{1} w^{i,j}_{1} \\ c^{j}_{1}\ c^{j}_{1} \end{bmatrix}^{((l-1)-j)} \begin{bmatrix} r^{i,j}_{1} w^{i,j}_{0} \\ c^{j}_{0}\ c^{j}_{0} \end{bmatrix} \begin{bmatrix} r^{i,j}_{0} w^{i,j}_{0} \\ c^{j}_{0}\ c^{j}_{0} \end{bmatrix}^{(j+1)} \tag{9.10}$$

$$\begin{bmatrix} r^{i,j}_{0} w^{i,j}_{0} \\ c^{j}_{0}\ c^{j}_{0} \end{bmatrix}^{((l-1)-j)} \begin{bmatrix} r^{i,j}_{0} w^{i,j}_{0} \\ c^{j}_{0}\ c^{j}_{0} \end{bmatrix} \begin{bmatrix} r^{i,j}_{0} w^{i,j}_{0} \\ c^{j}_{0}\ c^{j}_{0} \end{bmatrix}^{(j+1)} \tag{9.11}$$

We now verify the presence of elementary tests for input stuck-at faults:

b-sa-0 According to Table 7.3, a test that detects this fault is $w_1 r$. The $w^{i,j}_{1} r^{i,j}_{1}$ sequence occurs in (9.7) of $T^{i,j}_{C\text{-}SM}$. Although after the initial $w^{i,j}_{1}$ the cell's state is indeterminate, the subsequent $r^{i,j}_{1}$ forces the cell to a determinate, erroneous state 0 and returns this erroneous value.

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

b-sa-1 According to Table 7.3, a test that detects this fault is $w_1 w_0 r$. In the presence of this fault, 'read' operations do not affect the cell's state. The sequence $w_1^{i,j} w_0^{i,j}$ occurs during the execution of (9.7) and (9.8) in $T^{i,j}_{C\text{-}SM}$. Each $w_0^{i,j}$ is guaranteed to be followed by at least one $r_0^{i,j}$.

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

w-sa-0 In the presence of this fault 'read' operations do not provide a reliable output; they do not, however, affect the cell's state. According to Table 7.3, a test that detects this fault is $w_1 c_1 w_0 c_0$. The sequence $w_1^{i,j} w_0^{i,j}$ occurs during the execution of (9.7) and (9.8) in $T^{i,j}_{C\text{-}SM}$. Although $c_1^j c_0^j$ is performed "in parallel", these 'compare' operations actually occur after the 'write' is completed, and thus they are applied to the newly written value; hence the required test occurs in $T^{i,j}_{C\text{-}SM}$.

c-sa-0 According to Table 7.3, a test that detects this fault is $w_0 c_1$. This test occurs during the execution of (9.6) and (9.7) in $T^{i,j}_{C\text{-}SM}$, before the first $w_1^{i,j}$. During (9.6) the cell is subject to a $w_0^{i,j}$. During (9.7), a c_1^j occurs concurrently with the $r_0^{i,j}$, that precedes the first $w_1^{i,j}$. Since in the presence of this fault 'read' operations do not affect the state of the cell, the required test occurs in $T^{i,j}_{C\text{-}SM}$.

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

c-sa-1 According to Table 7.3, a test that detects this fault is $w_0 c_0$. A $w_0^{i,j}$ is certain to occur in (9.6), (9.8), (9.10) and (9.11) of $T^{i,j}_{C\text{-}SM}$. Each $w_0^{i,j}$ is guaranteed to be followed by one c_0^j during the same clock cycle.

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

m-sa-0* and *m-sa-1 Since the array is "filled" serially, each of the rows in the array holds a unique value some point during the execution of a march element. A comparison with an identical key-word that results a single match is performed (See Table 6.3). According to Table 7.3, tests that detect these faults are c_1c_0 or c_0c_1 while the state of the cell remains unchanged. The sequence $c_0^jc_1^j$ occurs during the execution of (9.6) and (9.7), and during the execution of (9.8) and (9.9) in $T^{i,j}_{C\text{-}SM}$ concurrently with $w_0^{i,j}r_0^{i,j}$ that immediately precedes a $w_1^{i,j}$. Symmetrically, the sequence $c_1^jc_0^j$ occurs during the execution of (9.7) and (9.8) and during the execution of (9) and (10) in $T_{C\text{-}SM}$ concurrently with $w_1^{i,j}r_1^{i,j}$ that immediately precedes a $w_0^{i,j}$.

We now focus on the non-independently testable faults:

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

The analysis of the $T_{C\text{-}SM}$ test in our FSM model of input stuck-at faults shows that $T_{C\text{-}SM}$ detects all the faults in the model.

9.3 Summary

We have shown how evaluation or verification of tests can be conducted within our framework, by evaluating two CAM tests with respect to the input stuck-at fault model of a CAM cell of Fig. 7.1, as examples. We have demonstrated that the $T_{G\&H}$ test originally developed for a different CAM cell [19], does not reliably detect the *b-sa-0* and $\bar{b}$*-sa-0* faults and requires a modest restriction on the size of the CAM. This test can be modified to achieve 100% fault coverage at the cost of increased length. We have also shown that the $T_{C\text{-}SM}$ test [31] provides 100% fault coverage with no restrictions on the size of the CAM; however, it is significantly longer. This length is dictated by the requirements of BIST.

Although not all fault models were considered, this exercise clearly indicates the necessity of creating custom fault models that reflect the idiosyncrasies of a particular cell design.

Chapter 10

SRAM Testing

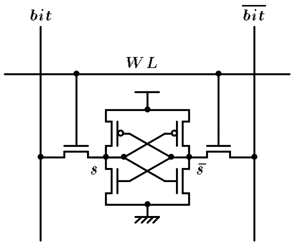
Word-oriented static random-access memories are well-known storage devices. Though their bit densities are not nearly as great as those of DRAMs, their speed and reliability makes them currently the most common choice for embedding in larger ASICs. One application of this type of memory is the implementation of the data field in caches.

In this chapter we investigate the testability properties of a word-oriented SRAMs based on the cell shown in Figure 10.1(a). In Chapter 7 we introduced the *input stuck-at fault model* which consists of any stuck-at fault which affects the input lines of a memory cell; here this model has been developed for the SRAM cell. The focus of this chapter is the evaluation of well-known tests, MATS+, MATS++, MARCH Y and MARCH C-, with respect to the input stuck-at fault model of an n-word by l-bit SRAM. Here, we demonstrate that any test that uses 'read' and 'write' operations can reliably detect at most 50% of the faults in our fault model. We also show that MATS+ has even worse fault coverage. These results have been previously reported in [46]

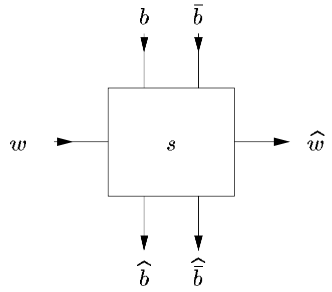
10.1 Analysis of an SRAM Cell

The following behavioral analysis has been applied to a CAM in Chapter 7. Since an SRAM cell constitutes only the storage section of a CAM cell, a simplified analysis is presented here.

10.1.1 SRAM Cell Circuit



(a)



(b)

Figure 10.1: (a) SRAM cell, (b) its model.

The circuit of the cell is shown in Figure 10.1(a). It consists of two cross-coupled CMOS inverters, differential bit lines ($bit/\overline{bit}$) used for reading and writing data into a column of cells, and a word select line (WL) that enables these operations in a row of cells. In a quiescent state WL is driven low, the $bit/\overline{bit}$ lines are driven high and the cell stores either 0 or 1. During a 'write 1' or a 'write 0' operation, the $bit/\overline{bit}$ lines are driven to a true/complementary representation of the desired bit value. Raising and then lowering WL stores the bit in the cell. Changes of the cell's state, when writing a 1 to a cell containing a 0, for example, are a result of the dominant influence of stronger pull-down transistors. (Weaker pull-up transistors maintain quiescent states of the cell.) A 'read' operation is performed by isolating both $bit/\overline{bit}$ lines, and then raising WL, thus causing one of the $bit/\overline{bit}$ lines to discharge. The resultant voltage differential between the $bit/\overline{bit}$

lines is detected by sense amplifiers that re-create the contents of the accessed word.

Altogether, three operations can be performed on a SRAM cell: 'write 0', 'write 1' and 'read'.

10.1.2 Event-Sequence Model

To model input stuck-at faults, we have to understand how they affect the internal operation of the cell. A single SRAM cell is an asynchronous sequential circuit whose behavior can be modeled by the block diagram of Figure 10.1(b). Since $bit/\overline{bit}$ lines are used for both input and output in the circuit, they are represented by separate variables in the model. For brevity we use b, $\bar{b}$, w for $bit/\overline{bit}$, WL. Now, b and $\bar{b}$ are inputs and $\widehat{b}$ and $\widehat{\bar{b}}$ are outputs. Although WL is not usually meant to provide any output, monitoring the state of this line, if possible, might improve the SRAM's testability (potential source of DFT suggestions). For this reason, we generalize our model to include this line; w for input, and $\widehat{w}$ for output.

The total state of the cell is defined by the values present on the input and output lines of the cell, and by its internal state s. It is represented by the 7-tuple:

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

However, it turns out that in the correct cell, as well as in the presence of input stuck-at faults, the output values are identical to those of the inputs; hence, we omit them for simplicity. The simplified total state is symbolically represented by $C = w\ b\bar{b}\ \cdot s$, where the input variables have been separated by a space for readability and the symbol $\cdot$ has been inserted to separate the input variables from the internal state.

The domain of each variable in state C is the set $Y = \{0, 1, \tilde{0}, \tilde{1}\}$. The values 0 and 1 represent lines driven to the logic values 0 and 1 respectively, while $\tilde{0}$ and $\tilde{1}$ denote lines that were first discharged and then isolated (*floated low*) or pre-charged and then isolated (*floated high*). The SRAM cell has two possible initial states: $C = 0\ 11 \cdot 0$ and $C = 0\ 11 \cdot 1$.

Table 10.1: Read and write operations in a fault-free SRAM cell.

Read operations			Write operations				
	$(s=0)$	$(s=1)$		$(s=0)$		$(s=1)$	
				w_0	w_1	w_0	w_1
Description	$w\ b\bar{b}{\cdot}s$	$w\ b\bar{b}{\cdot}s$	Description	$w\ b\bar{b}{\cdot}s$	$w\ b\bar{b}{\cdot}s$	$w\ b\bar{b}{\cdot}s$	$w\ b\bar{b}{\cdot}s$
initial state	$0\ 11{\cdot}0$	$0\ 11{\cdot}1$	initial state	$0\ 11{\cdot}0$	$0\ 11{\cdot}0$	$0\ 11{\cdot}1$	$0\ 11{\cdot}1$
float $b/\bar{b}$	$0\ \tilde{1}\tilde{1}{\cdot}0$	$0\ \tilde{1}\tilde{1}{\cdot}1$	set $b/\bar{b}$	$0\ 01{\cdot}0$	$0\ 10{\cdot}0$	$0\ 01{\cdot}1$	$0\ 10{\cdot}1$
raise w	$1\ \tilde{1}\tilde{1}{\cdot}0$	$1\ \tilde{1}\tilde{1}{\cdot}1$	raise w	$1\ 01{\cdot}0$	$1\ 10{\cdot}0$	$1\ 01{\cdot}1$	$1\ 10{\cdot}1$
b or $\bar{b}$ discharges[1]	$1\ 0\tilde{1}{\cdot}0$	$1\ \tilde{1}0{\cdot}1$	new state	$1\ 01{\cdot}0$	$1\ 10{\cdot}1$	$1\ 01{\cdot}0$	$1\ 10{\cdot}1$
read $b/\bar{b}$ & lower w	$0\ \tilde{0}\tilde{1}{\cdot}0$	$0\ \tilde{1}\tilde{0}{\cdot}1$	lower w	$0\ 01{\cdot}0$	$0\ 10{\cdot}1$	$0\ 01{\cdot}0$	$0\ 10{\cdot}1$
raise $b/\bar{b}$	$0\ 11{\cdot}0$	$0\ 11{\cdot}1$	raise $b/\bar{b}$	$0\ 11{\cdot}0$	$0\ 11{\cdot}1$	$0\ 11{\cdot}0$	$0\ 11{\cdot}1$

The behavior of a cell is represented by sequences of events that take place during the three operations. Table 10.1 lists events that occur during these operations. By events we mean changes in the value of the total state C. The occurrences of these events are ordered from top to bottom. Note that each operation starts in (top entry) and returns to (bottom entry) an initial state.

Example: 'Write 0' operation from state 1.

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

If it were possible to control any input and observe any output, testing would be a trivial process: if a simple comparison of the state of each line to its expected value were made, detection of any disagreement would indicate a fault. Unfortunately, it is

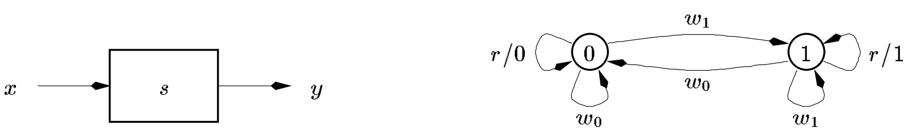
[1]There is a connection from bit to V_{dd} when $s = 0$ and from $\overline{bit}$ to V_{dd} when $s = 1$. But this connection is through a weak p-transistor and an n-transistor, and drivers for the $bit/\overline{bit}$ are not used. Hence, we still represent these cases as floating values.

not possible to monitor any real circuit at this level of detail and, thus, a more abstract model of an SRAM cell is needed.

10.1.3 FSM Model

Most testing algorithms utilize sequences of 'read' and 'write' operations as input, and observe the resulting output. Accordingly, we introduce an FSM model of an SRAM cell. This model is derived from the event-sequence model of Section 10.1.2.

We represent the behavior of a fault-free SRAM cell as a simplified block diagram, which is shown in Figure 10.2(a). The input x of this FSM comprises all three operations



(a) (b)

Figure 10.2: (a) Simplified behavioral model (b) Behavior of a fault-free SRAM cell.

of a SRAM cell, and the output y comprises the responses to these operations. State s represents the value stored by the cell. Formally, a *fault-free SRAM cell* is a Mealy automaton $M = (Q, X, Y, \delta, \lambda)$, where $Q = \{0, 1\}$ is the set of states, $X = \{r, w_0, w_1\}$ is the set of input symbols, $Y = \{0, 1, \$\}$ is the set of output symbols, where \$ is a formal symbol denoting lack of output during 'write' operations, and the transition function δ and the output function λ are defined by

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

This FSM is depicted in Figure 10.2(b), where the symbol \$ has been omitted for clarity.

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

In an *faulty SRAM cell* FSM, the faulty set of states Q' is $Q \cup \{I\}$ and the faulty set of output symbols Y' is $Y \cup \{I\}$, where I represents an "indeterminate" logic value, as defined in Chapter 7.

10.2 Input Stuck-at Faults in an SRAM Cell

A event-sequence analysis has been performed for all six input stuck-at faults under a single-fault assumption; these analyses are presented in Appendix B.1. Subsequently, FSM models for each of these faults have been developed. The resulting *faulty SRAM cells* are presented in Figure 10.3, where incorrect operations and outputs are in bold type.

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

From the event-sequence model we construct an FSM. In this example, operations w_0 and r are incorrect, but w_1 is correct; hence we get Figure 10.3(c).

Just as for the CAM cell of Chapter 7, the *b-sa-0* and $\bar{b}$*-sa-0* faults increase the state set Q by the "indeterminate" state I.

The comparison of each of the faulty machines to the fault-free SRAM cell has led to the derivation of simple tests for four out of six possible faults. We refer to the shortest test that detects a particular fault in a single cell as an *elementary test*; these tests are essential to the detection of the associated faults. Table 10.2 lists all the elementary tests for an SRAM cell.

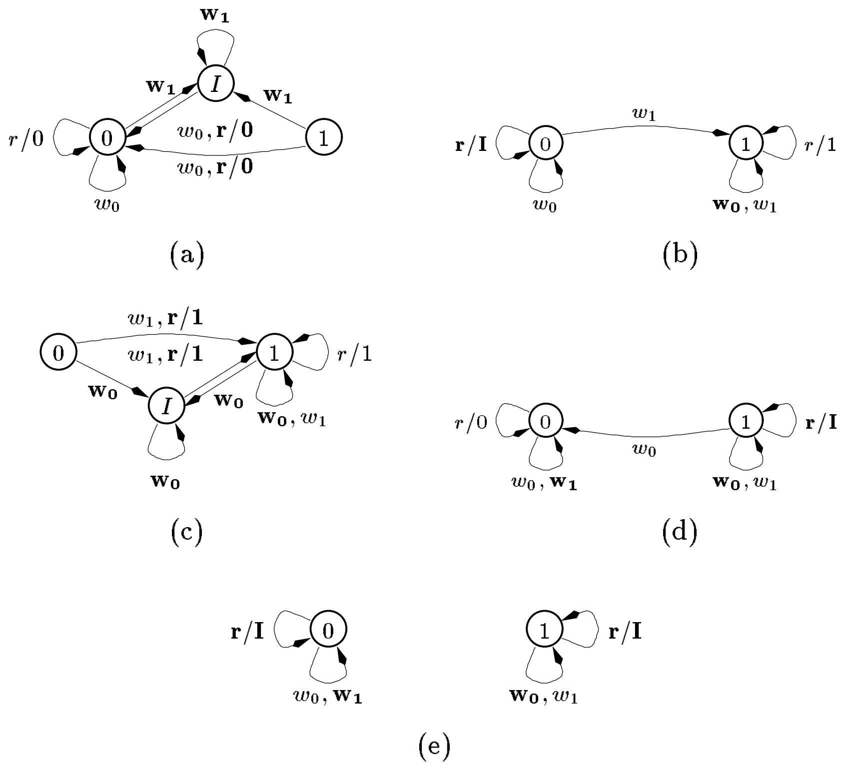


Figure 10.3: Faulty behaviors of an SRAM cell: (a) *b-sa-0*, (b) *b-sa-1*, (c) $\bar{b}$*-sa-0*, (d) $\bar{b}$*-sa-1*, (e) *w-sa-0*.

These tests have been generated by the Observer program for all input stuck-at faults except the *w-sa-1* and the *w-sa-0* faults. We refer to the faults that are detectable in a single cell as *independently testable*.

The reader should note that in the presence of *b-sa-1*, $\bar{b}$*-sa-1* and *w-sa-0* faults the 'read' operation produces an unreliable output, and also that in the presence of the *b-sa-0* and $\bar{b}$*-sa-0* faults this operation changes the state of the cell. For instance, in the last example a faulty w_0 forces the cell to an indeterminate state I, whereas a good cell would remain in state 0. This means that a faulty w_0 does not necessarily sensitize this fault. Fortunately, the faulty r forces the cell to state 1 and provides a reliable faulty output 1.

Table 10.2: Summary of input stuck-at faults and elementary tests.

Fault	Elementary Test	Faulty Response	Fault-Free Response
b-sa-0	w_1r	0	1
b-sa-1	w_1w_0r	1	0
$\bar{b}$*-sa-0*	w_0r	1	0
$\bar{b}$*-sa-1*	w_0w_1r	0	1
w-sa-0	not testable reliably		
w-sa-1	see Section 10.2.1		

The *w-sa-0* fault is an instance of an *address decoder fault A* [20] or a *stuck-open fault* [16]. Also, when started in state 0 (state 1) the *w-sa-0* fault is superficially similar to a cell stuck-at-0 (stuck-at-1) fault, however, the key difference between these faults is that in the case of the *w-sa-0* fault the 'read' operations produce an indeterminate faulty output I. Memory designers often claim that this indeterminate output is resolved preferentially due to inherently unequal bias of sense amplifiers, and that this fault should manifest itself as one of the cell stuck-at faults; we, nevertheless, regard this input stuck-at fault as generally not testable. Detectability of this fault by parametric tests depends on the associated physical defect and how it affects the peripheral circuitry, which is outside the scope of this thesis.

The *w-sa-1* fault FSM, is identical to that of the fault-free SRAM cell; therefore, no test exists for a single cell. This fault, however, is a generalization of an *address decoder fault D* [20] (a multiple coupling fault), where operations on any other word in the memory will affect the faulty word. Therefore, this fault is detectable in conjunction with other words [20]; this will be considered later. Consequently, the *w-sa-1* fault is the only input stuck-at fault that is not independently testable.

The *b-sa-1* (*$\bar{b}$-sa-1*) fault is similar to a cell unable to undergo the $1 \to 0$ ($0 \to 1$) transition. This fault is detected by $w_1 w_0 r$ ($w_0 w_1 r$).

10.2.1 Extension to n-word by 1-bit SRAM

We now consider an n-bit SRAM where each word consists of a single cell (a bit-oriented memory). At this point, an assumption must be made about the functionality of the peripheral circuitry in a fault-free SRAM.

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

We model the correct behavior of an n-word by 1-bit SRAM as a Mealy sequential machine $M = (Q, X, Y, \delta, \lambda)$, where $Q = \{0,1\}^n$, $X = (\bigcup_{i=1}^{n} A_i)$ with $A_i = \{r^i, w_0^i, w_1^i\}$, $Y = \{0, 1, \$\}$ and the transition function δ and the output function λ are defined by

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

and

$$\lambda((q^1, \ldots, q^i, \ldots, q^n), x) = \begin{cases} q^i, & \text{if } x = r^i, \\ \$, & \text{otherwise.} \end{cases}$$

As before, the inputs r^i, w_0^i, w_1^i denote the 'read', 'write 0' and 'write 1' operations on word i, respectively, and the output \$ is merely a formal symbol denoting lack of output.

The w^i-*sa-0* Fault

As stated before, in the presence of a w^i-*sa-0* fault none of the cells along the faulty word line can be written to, or reliably read. Moreover, no operation on any of the remaining

$n-1$ fault-free words can sensitize the cells along the faulty word line such that this fault could be detected. Therefore, in an n-word by 1-bit SRAM, there are n possible w^i-*sa-0* faults that cannot be reliably detected by any combination of 'read' and 'write' operations.

Testing the w^i-*sa-1* fault

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

Assume w^i-*sa-1*. Since word i is always accessed, a 'read' or 'write' operation may be applied to two words simultaneously. In this case, a 'write' operation is always successful for both words. On the other hand, the result of a 'read' operation will be unreliable if the two accessed words contain opposing values; this will not, however, disrupt the contents of either word.

A possible test is:

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

where $1 \leq i \leq n$. The symbol $(\ )^{\updownarrow}$ denotes n operations. These can be done in order either from n to 1 or from 1 to n; hence the $\updownarrow$. The symbol $(\ )^{(n-1)\downarrow 1}$ denotes the direction of the march element: from $n-1$ to 1. Initially, 0s are written into every word. If word u, where $1 \leq u < n$ is the faulty word, then w_1^n will write 1s to both u and n. Thus r^i, when $i = u$, will generate a 1. Now, let $u = 1$. If word n is the faulty word, then word 1 is not, so w_0^1 will write 0 to words 1 and n, and the subsequent r^n will generate a 0. Thus, the occurrence of a 1 during any of the first $n-1$ 'read' operations and an occurrence of a 0 due to the last 'read', indicates the *w-sa-1* fault.

10.2.2 Extension to a 1-word by l-bit SRAM

We now consider a single row of cells comprising an SRAM word.

Let $\mathcal{B} = \{0,1\}$, and to extend this set to represent l-bit words, let $\mathcal{V} = \mathcal{B}^l$. The behavior of a 1-word by l-bit SRAM can be specified by a Mealy sequential machine $M = (Q, X, Y, \delta, \lambda)$ where $Q = \mathcal{V}$, $X = \{r\} \cup (\bigcup_{p \in \mathcal{V}} w_p)$, and $Y = \mathcal{V} \cup \{\$\}$. For $q \in \mathcal{V}$, $x \in X$ the transition function δ and the output function λ are defined by

$$\delta(q,x) = \begin{cases} q, & \text{if } x = r, \\ p, & \text{if } x = w_p,\ p \in \mathcal{V}, \end{cases} \quad \text{and} \quad \lambda(q,x) = \begin{cases} q, & \text{if } x = r, \\ \$, & \text{otherwise.} \end{cases}$$

Faults affecting the $bit/\overline{bit}$ lines of each of the l cells are detectable by performing the respective tests on all cells in parallel since these faults manifest themselves with an erroneous l-bit output of the 'read' operation. Detection of the *w-sa-0* fault is not possible, as this fault affects the entire word the same manner.

10.2.3 Extension to n-word by l-bit SRAM

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

Let $\mathcal{V}$ be defined as before. The correct behavior of a n-word by l-bit SRAM is denoted by a Mealy sequential machine $M = (Q, X, Y, \delta, \lambda)$, where $Q = \mathcal{V}^n$, $X = (\bigcup_{i=1}^{n} A_i)$ with $A_i = \{r^i\} \cup (\bigcup_{p \in \mathcal{V}} w_p^i)$, $Y = \mathcal{V} \cup \{\$\}$. For $1 \leq i \leq n$ and for $q^i \in \mathcal{V}$, $x \in X$, $p \in \mathcal{V}$, the transition function δ and the output function λ are defined by

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

and

$$\lambda((q^1,\ldots,q^i,\ldots,q^n),x) = \begin{cases} q^i, & \text{if } x = r^i,\ 1 \le i \le n, \\ \$, & \text{otherwise.} \end{cases}$$

From the combined fault analysis we conclude that, for the input stuck-at fault model, only the w^i-*sa-0* faults cannot be tested reliably. Since in a n-word by l-bit SRAM, n such faults may exist, the best possible fault coverage of any test under our fault model is $\frac{n+4l}{2n+4l} \cdot 100\%$, which for an 8k-word by 8-bit SRAM [16] is 50.1%, but for a 32-word by 73-bit SRAM [31] is 91.01%.

10.3 Evaluation of Tests

In this section we analyze well-known tests with respect to the input stuck-at fault model. A complete evaluation of the MATS+ test is given here. The same approach has been applied to MATS++, MARCH Y and MARCH C- and the summaries of these evaluations are given. The complete analyses of the latter three tests can be found in Appendix B.2.

10.3.1 Evaluation of the MATS+ test

The MATS+ test [20] for a n-word by 1-bit SRAM, is of length $5n$, and is presented below:

$$\textrm{MATS+} = (w_0^i)^{\updownarrow}(r^i w_1^i)^{n\downarrow 1}(r^i w_0^i)^{1\uparrow n}.$$

First, all the words are initialized to 0. Then, in the fault-free SRAM, each w_1^i in the second march element is preceded by an r^i which should produce a 0. This march element is performed in the direction from n to 1. The rest of the test sequence is similar, with 0 and 1 interchanged and the march element direction reversed.

Evaluation of the MATS+ test for a single cell

First, we restrict the above test to a single cell. Each cell in this test is subject to the following sequence of operations: $w_0 r w_1 r w_0$. Table 10.3 provides the comparison of the fault-free response with the faulty responses of all independently testable faults. The deviations from the fault-free response are indicated in boldface. Tests for individual faults that are included in MATS+ are also given. From this table it is clear that the *b-sa-1* fault will not be detected reliably, as the elementary test for this fault is not present in the MATS+ sequence for a single cell. The reader can verify that for an inverted MATS+ test $w_1 r w_0 r w_1$, where each cell undergoes the complementary sequence of operations, it is the $\bar{b}$-*sa-1* fault that will not be reliably detected.

Table 10.3: Evaluation of the MATS+ test for a single cell.

Fault	MATS+ $w_0 r w_1 r w_0$	Elementary tests for input stuck-at faults within MATS+
fault-free cell	$-\,0\,-\,1\,-$	
b-sa-0	$-\,0\,-\,\mathbf{0}\,-$	$w_0 r \mathbf{w_1 r} w_0$
b-sa-1	$-\,\mathbf{1}\,-\,1\,-$	missing test: $w_1 w_0 r$
$\bar{b}$-*sa-0*	$-\,\mathbf{1}\,-\,1\,-$	$\mathbf{w_0 r} w_1 r w_0$
$\bar{b}$-*sa-1*	$-\,0\,-\,\mathbf{0}\,-$	$\mathbf{w_0 w_1 r} w_0$

Evaluation of the MATS+ test for n-word by 1-bit SRAM

We verify that MATS+ detects w^i-*sa-1* faults by showing that it contains the $T_{w\text{-}sa\text{-}1}$ test. Given

$$\text{MATS+} = (w_0^i)^{\updownarrow}\ (r^i w_1^i)^{n\downarrow 1}\ (r^i w_0^i)^{1\uparrow n},$$

We expand the latter two march elements and get

$$(w_0^i)^{\updownarrow}\ r^n w_1^n (r^i w_1^i)^{(n-1)\downarrow 1}\ r^1 w_0^1 (r^i w_0^i)^{2\uparrow (n-1)} r^n w_0^n.$$

We can disregard the $(w_1^i)^{(n-1)\downarrow 1}$, $(w_0^i)^{2\uparrow(n-1)}$ and w_0^n operations as, for any given word, they occur after a 'read', and they are not required to sensitize the faulty word. The sensitization is accomplished first by the w_1^n and then by the w_0^1. We can also disregard the r^n, r^1 and $(r^i)^{2\uparrow(n-1)}$ operations, as they do not corrupt the contents of the cells[2] and thus we get

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

which completes our proof.

We have shown that Mats+ is unable to detect *b-sa-1* and w^i*-sa-0* [3] faults. We thus conclude that in a n-word by 1-bit SRAM, under the single-fault assumption, this test can reliably detect $\frac{n+3}{2n+4} \cdot 100\%$ possible input stuck-at faults, which is roughly 50% of faults (for large n) in the input stuck-at model.

Evaluation of the Mats+ test for n-word by l-bit SRAM

The word-oriented extension of the Mats+ test for a n-word by l-bit SRAM takes the form:

$$\text{Mats+} = (w_{0\ldots0}^i)^{\updownarrow}(r^i w_{1\ldots1}^i)^{n\downarrow 1}(r^i w_{0\ldots0}^i)^{1\uparrow n},$$

where $w_{0\ldots0}$ denotes the writing of an all-0 word, etc. This is often referred to as a *data background* [16]. The reader should note that, for input stuck-at faults, this extended test detects the same faults per cell as the test for a single cell. Changing the data background does not affect the relative number of faults that may be undetected; it does, however, affect the type of the undetectable faults. There are l possible b^j*-sa-1* faults, where $1 \leq j \leq l$ in a 1-word by l-bit SRAM, and therefore, under the single-fault assumption, this word-oriented extension of the Mats+ test will reliably detect

[2] Single fault assumption.

[3] See Section 10.2.1

$\frac{n+3l}{2n+4l} \cdot 100\%$ of faults under the input stuck-at fault model. Thus, for the previously mentioned (8k $\times$ 8)-bit SRAM, the coverage is 50.05% - a decrease of 0.05% from the optimal coverage of Section 10.2.3. However, for memories with long words, such as the (32 $\times$ 73)-bit SRAM the coverage is only 70.5% - a decrease of 20.51%.

10.3.2 Evaluation of the MATS++ test

The MATS++ test [20] is a well-known, bit-oriented test, of length $6n$. One possible word-oriented extension of this test for an n-word by l-bit SRAM is:

$$\text{MATS++} = (w^i_{0\ldots0})^{\updownarrow}(r^i w^i_{1\ldots1})^{n\downarrow1}(r^i w^i_{0\ldots0} r^i)^{1\uparrow n}.$$

As shown in Appendix B.2, this word-oriented extension of the MATS++ test will reliably detect all the detectable input stuck-at faults, which constitute $\frac{n+4l}{2n+4l} \cdot 100\%$ of all the faults in the fault model; hence for large n the fault coverage is roughly 50%.

10.3.3 Evaluation of the MARCH Y test

Another well-known, bit-oriented test is the MARCH Y test [20], of length $8n$. A word-oriented extension of this test is presented below:

$$\text{MARCH Y} = (w^i_{0\ldots0})^{\updownarrow}(r^i w^i_{1\ldots1} r^i)^{n\downarrow1}(r^i w^i_{0\ldots0} r^i)^{1\uparrow n}(r^i)^{\updownarrow}.$$

This extension of the MARCH Y test will also reliably detect all the detectable input stuck-at faults, i.e. $\frac{n+4l}{2n+4l} \cdot 100\%$ of all the faults in the fault model. This coverage is identical to that of the MATS++, yet the MARCH Y is longer, as it contains redundant elements with respect to the detection of input stuck-at faults.

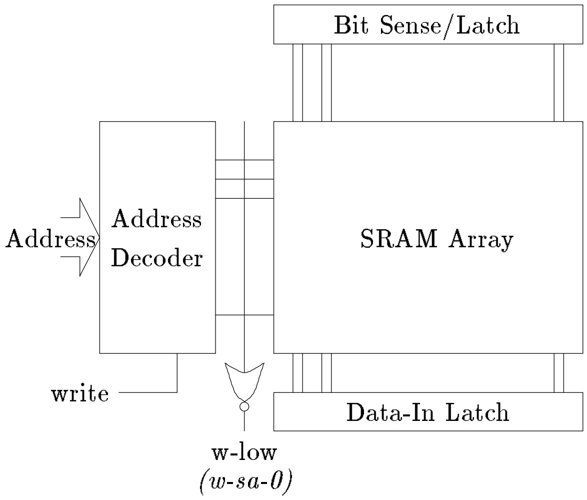


Figure 10.4: DFT suggestions.

10.3.4 Evaluation of the March C- test

The March C- test [20], of length $10n$, is also a well-known, bit-oriented test. A word-oriented extension of this test is presented below:

$$\text{March C-} = (w^i_{0\ldots 0})^{\updownarrow}(r^i w^i_{1\ldots 1})^{n\downarrow 1}(r^i w^i_{0\ldots 0})^{n\downarrow 1}(r^i w^i_{1\ldots 1})^{1\uparrow n}(r^i w^i_{0\ldots 0})^{1\uparrow n}(r^i)^{\updownarrow}.$$

This extension of the March C- test will also reliably detect all the detectable input stuck-at faults. This coverage of input stuck-at faults is identical to that of the Mats++ and March Y, yet the March C- is longer than either of these two tests.

10.4 DFT Suggestions

Earlier we have shown that the w^i-*sa-0* faults cannot be tested reliably by any combination of 'read' and 'write' operations. These faults constitute $\frac{n}{2n+4l} \cdot 100\%$ of faults in our fault model. In Section 10.1.2 we mentioned that considering word select lines as a source of output $\widehat{w}^i$ could improve the SRAM's testability. By providing an additional output

w-$low = NOR(\widehat{w}^1, \ldots, \widehat{w}^n)$, depicted as a logic gate in Figure 10.4, the fault coverage can be improved significantly. This simple DFT enhancement, by indicating that none of the word lines is active, can directly detect any w^i-*sa-0* fault, increasing the fault coverage of the MATS++, MARCH Y and MARCH C- tests to 100% with respect to the input stuck-at fault model.

10.5 Summary

Using a n-word by l-bit static CMOS SRAM as a basis, we have evaluated some commonly known tests with respect to a simple class of faults: the input stuck-at faults. We have demonstrated that these tests may fail to detect close to 50% of faults (for large n) in our fault model. This poor performance is attributed to the fact that 'read' operations are insufficient for the detection of all the input stuck-at faults in this type of static CMOS SRAM. In order to provide 100% fault coverage some DFT enhancement, such as the one that has been suggested, is necessary. We have shown that tests MATS++, MARCH Y and MARCH C- have the same fault coverage despite their different lengths, and that the MATS+ test has an inferior fault coverage in comparison to the previous three. We have also demonstrated that if a column-wise fault (e.g. a bit line fault) is not detectable by a particular bit-oriented test, then complementing the data values in the test will result in a complementary fault not being detected. This property is magnified when such a test is modified to word-oriented memories, because different data backgrounds will have a different combination of complementary types of faults being undetected.

Chapter 11

Conclusions

In this thesis we have presented a formal framework for modeling and testing memories. Our goal was to standardize fault models and to associate them with the specific design, functionality and the underlying technology of memories. This is particularly important in view of the increasing number of application-specific memories that are being developed. We strove to systematize the process of fault modeling, test development, evaluation and verification.

Our framework expands on the concept of defect-based fault modeling and inductive fault analysis. We present it using a CAM cell as an example. A brief review of the obtained results are given below.

11.1 Review

In Chapter 2 fundamental aspects of testing digital circuits have been described using the most common fault model, the stuck-at fault model, as an example. This fault model is insufficient to describe all possible faulty behaviors of a digital circuit, particularly one implemented in CMOS technology. Other fault models are required and appropriate tests have to be devised. Moreover, due to the existence of non-logical faults, logic tests have to be complemented with parametric tests.

Logic testing in itself is a complex issue, particularly when applied to sequential circuits. Exhaustive testing, in general, is an NP-hard problem, so fault models have to be tailored to the circuits under test in order to be accurate, and for the respective tests to be short enough to be useful.

Although circuits should be designed to include features that improve testability, such as scan paths (DFT), test lengths should be minimal while providing optimal fault coverage in the fault model.

In Chapter 3 an overview of memory testing topics has been presented. Traditional and modern tests have been described and compared in terms of their lengths and fault coverage. Faults indigenous to static and dynamic memories have been described. This characterization is a result of defect-based fault modeling techniques. We have also listed tests designed exclusively for SRAMs. These tests are more efficient and yield higher fault coverage than general functional tests. A design-oriented approach to fault modeling has been proposed. Its purpose is to categorize faults in terms of defects in the cell circuit and not in the cell's functional behavior. This approach should result in more accurate fault models that take into account the cell design, implementation technology, and design-specific functionality of a memory.

In Chapter 4 we have presented an overview of a formalism developed by Brzozowski and Jürgensen for sequential circuit testing and diagnosis [11]. We utilize this formalism in our behavioral analysis of memory cells. It provides the means to distinguish faulty behaviors from the fault free ones and to derive shortest input sequences with which that distinction can be made.

In Chapter 5 we have given an overview of a CAM architecture. We have presented several different CAM cell designs, both static and dynamic, and commented on their strengths and weaknesses. It is important to note that although all CAMs will perform the same operations, the way these operations are implemented varies among cell designs. For this reason it is reasonable to expect that these cells will exhibit different faulty behaviors in the presence of similar faults.

In Chapter 6 we have described several tests developed explicitly for CAMs. The fault models used for these tests are often not sufficient to describe various faults indigenous to CAMs. Al-Assadi et al. were first to develop a fault model specifically for a CAM. Lin and Wu expanded on that work, defined additional faults and designed a test for their fault model. Yet their fault model is still inadequate, ad dome of the fault analysis is flawed. Moreover, this fault model pertains only to single-port CAMs. Due to the variety of CAM cell designs a more general approach is necessary; one that can establish fault models that reflect defects that occur in a given CAM cell design.

In Chapter 7 we have developed a fault modeling methodology, using a particular static CMOS CAM as an example. The modeling steps included circuit analysis, asynchronous behavior analysis, and a FSM representation of the fault-free and faulty circuits. Four fault models for an n-word by l-bit static CMOS CAM have been defined: input stuck-at, state stuck-at, transistor stuck-(on/open), and bridging. It has been found that some of the described faults resemble well known fault categories: cell stuck-at, transition, coupling, etc. However, many of the faults presented here have a distinct behavior and do not cleanly fit these categories. A systematic behavioral analysis produced the following results:

- All input and state stuck-at faults are reliably detectable by functional tests, although some faults are not independently testable and have to be considered in conjunction with other cells, as shown in Chapter 8.

- $\frac{2nl}{18} \times 100\%$ (approx. 11%) of transistor faults are not detectable by functional CAM tests; however all transistor-open faults that compromise data retention in static memory cells can be reliably detected, due to the enhanced functionality of this type of memory.

- $\frac{9nl+7l-2}{18nl+8l+2n-4} \times 100\%$ (approx. 50%) of faults in the bridging model are not reliably detectable by functional tests.

- Faults that are not detectable by functional tests cause an increase of I_{DD} or I_{DDQ}; hence they are detectable through parametric testing.

- 'Read' operations are inadequate sources of output for testing static CMOS CAMs; 'compare' operations are a more reliable choice.

- Elementary tests, of length at most 3, have been determined; these tests are necessary to detect reliably testable faults in an arbitrary CAM cell.

In Chapter 8 we have presented the construction of the $T_{S\&B}$ test which detects all reliably testable faults under the input stuck-at, state stuck-at, transistor stuck-(on/open) and bridging fault models. The construction process consists of the development of a test for a single cell, extension of that test first to an n-word by 1-bit CAM (bit-oriented memory), then to a 1-word by l-bit CAM (single l-bit word), and by combining the two, the generation of a linear-time test for an n-word by l-bit CAM. In addition, some DFT enhancements that reduce the test length have been suggested. With a *bulk-write* the test will be linear in the word length.

In Chapter 9 we have shown how evaluation or verification of tests can be conducted within our framework, by evaluating two CAM tests with respect to the input stuck-at fault model of a CAM cell of Fig. 7.1, as examples. We have demonstrated that the $T_{G\&H}$ test originally developed for a different CAM cell [19], does not reliably detect the *b-sa-0* and $\bar{b}$*-sa-0* faults and requires a modest restriction on the size of the CAM. This test can be modified to achieve 100% fault coverage at the cost of increased length. We have also shown that the $T_{C\text{-}SM}$ test [31] provides 100% fault coverage with no restrictions on the size of the CAM; however, it is significantly longer. This length is dictated by the requirements of BIST.

Although not all fault models were considered, this exercise clearly indicates the necessity of creating custom fault models that reflect the idiosyncrasies of a particular cell design.

In Chapter 10 we have evaluated some commonly known tests with respect to the input stuck-at fault model, using a n-word by l-bit static CMOS SRAM as a basis. We have demonstrated that these tests may fail to detect close to 50% of faults (for large n) in our fault model. This poor performance is attributed to the fact that 'read' operations are insufficient for the detection of all the input stuck-at faults in this type of static CMOS SRAMs. In order to provide 100% fault coverage some DFT enhancement, such as the one that has been suggested, is necessary. We have shown that tests MATS++, MARCH Y and MARCH C- have the same fault coverage despite their different lengths, and that the MATS+ test has an inferior fault coverage in comparison to the previous three. We have also demonstrated that if a column-wise fault (e.g. a bit line fault) is not detectable by a particular bit-oriented test, then complementing the data values in the test will result in a complementary fault not being detected. This property is magnified when such a test is modified to word-oriented memories, because different data backgrounds will have a different combination of complementary types of faults being undetected.

11.2 Contributions

The two major contributions of this research are:

- The systematization of the fault modeling, test generation and test evaluation processes. Our approach has led to the identification of cell behaviors that are not covered by the traditional fault models. It has simplified and organized tasks that have previously been done mostly in an *ad hoc* manner.

- The design-oriented fault modeling postulate. Our guidelines help prevent the misapplication of tests to functionally similar, but technologically different circuits. Also, if fault models are representative of the architecture and technology of a circuit, fault coverage will be easier to ascertain, which may result in more efficient tests.

11.3 Future Work

The research presented in this thesis has opened numerous venues for future study. The most obvious ones are listed below:

- CAM tests evaluated with respect to the input stuck-at fault model should also be evaluated in terms of the transistor and bridging fault models.

- Transistor and bridging fault models have to be developed for the SRAM cell.

- Fault models should be developed for other type of memories: single-port static and dynamic CAMs, multi-port SRAMS, DRAMs, and FLASH memories.

- Commonly used tests should be evaluated with respect to these fault models.

- A more efficient version of the OBSERVER program should be developed.

- The feasibility of automatic fault extraction from circuit simulations should be investigated.

Appendix A

CAM Fault Analysis and Test Verification

A.1 CAM Fault Analysis

Here, we discuss operations observably affected by input stuck-at, state stuck-at, transistor stuck-(on/open) and bridging faults. The behaviors of operations not listed here may differ slightly from their fault-free counterparts in the event-sequence model; however in the FSM model these differences cannot be observed. We also assume that any output from state I is unreliable. Such operations are also omitted for clarity, unless they affect the state of the cell. Deviations from the fault-free behavior are indicaated by bold type.

A.1.1 Input Stuck-at Faults

***b-sa-0* fault.** This fault does not affect operations where the bit lines are not used, or those where b is normally driven to 0, i.e., c_0, c_1, $c_\times$, w_0, and r when $s = 0$. Table A.1 describes the faulty behaviors. During an r when $s = 1$, the grounded

Table A.1: Faulty CAM cell behavior due to the *b-sa-0* fault.

Read operations		Write operations			
	$(s = 1)$		w_1 $(s = 0)$	w_1 $(s = 1)$	$w_\times$ $(s = 1)$
Description	w $b\bar{b}$ $c\bar{c}$ $m{\cdot}s$	Description	w $b\bar{b}$ $c\bar{c}$ $m{\cdot}s$	w $b\bar{b}$ $c\bar{c}$ $m{\cdot}s$	w $b\bar{b}$ $c\bar{c}$ $m{\cdot}s$
initial state	0 **01** 00 1·1	initial state	0 **01** 00 1·0	0 **01** 00 1·1	0 **01** 00 1·1
float $b/\bar{b}$	0 **0$\tilde{1}$** 00 1·1	set $b/\bar{b}$	0 **00** 00 1·0	0 **00** 00 1·1	0 **01** 00 1·1
raise w	1 **0$\tilde{1}$** 00 1·1	raise w	1 **00** 00 1·0	1 **00** 00 1·1	1 **01** 00 1·1
b or $\bar{b}$ discharges	1 **0$\tilde{1}$** 00 1·**0**	new state	1 **00** 00 1·**I**	1 **00** 00 1·**I**	1 **01** 00 1·**0**
read $b/\bar{b}$ & lower w	0 **0$\tilde{1}$** 00 1·**0**	lower w	0 **00** 00 1·**I**	0 **00** 00 1·**I**	0 **01** 00 1·**0**
raise $b/\bar{b}$	0 **01** 00 1·**0**	raise $b/\bar{b}$	0 **01** 00 1·**0**|**1**	0 **01** 00 1·**0**|**1**	0 **01** 00 1·**0**

b drives s to ground, causing s to change before $\bar{b}$ has a chance to discharge. As a result, a 0 is always detected. During a w_1, when w is asserted, both $b/\bar{b}$ are 0 causing both T_3 and T_4 to conduct which, results in a metastable state. Once w is de-asserted either T_1 or T_2 dominates over the other and the cell eventually reverts back to one of the two quiescent states. Since the state of the cell cannot be predicted after a w_1, this operation must be followed immediately by a $w_\times$ or an r which forces the cell into a determinate faulty state. The $w_\times$ changes s to 0, thus resembling the behavior of a w_0. Analogous behavior is exhibited in the presence of the $\bar{b}$-*sa-0* fault.

***b-sa-1* fault.** Only r when $s = 0$ and w_0 when $s = 1$ are affected by this fault, as they rely on b being 0 for proper operation. Table A.2 describes these faulty behaviors. An r when $s = 0$ is misinterpreted by the sense amplifiers because both $b/\bar{b}$ remain

Table A.2: Faulty CAM cell behavior due to the *b-sa-1* fault.

Read operation		Write operation	
Description	$(s=0)$ $w\ b\bar{b}\ c\bar{c}\ m{\cdot}s$	Description	$w_0\ (s=0)$ $w\ b\bar{b}\ c\bar{c}\ m{\cdot}s$
initial state	$0\ 11\ 00\ 1{\cdot}0$	initial state	$0\ 11\ 00\ 1{\cdot}1$
float $b/\bar{b}$	$0\ \mathbf{1}\tilde{1}\ 00\ 1{\cdot}0$	set $b/\bar{b}$	$0\ \mathbf{1}1\ 00\ 1{\cdot}1$
raise w	$1\ \mathbf{1}\tilde{1}\ 00\ 1{\cdot}0$	raise w	$1\ \mathbf{1}1\ 00\ 1{\cdot}1$
b or $\bar{b}$ discharges	$1\ \mathbf{1}\tilde{1}\ 00\ 1{\cdot}0$	new state	$1\ \mathbf{1}1\ 00\ 1{\cdot}\mathbf{1}$
read $b/\bar{b}$ & lower w	$0\ \mathbf{1}\tilde{1}\ 00\ 1{\cdot}0$	lower w	$0\ \mathbf{1}1\ 00\ 1{\cdot}\mathbf{1}$
raise $b/\bar{b}$	$0\ 11\ 00\ 1{\cdot}0$	raise $b/\bar{b}$	$0\ 11\ 00\ 1{\cdot}\mathbf{1}$

high; hence, r is not a reliable source of output. (Since c_0, c_1 and $c_\times$ are not affected by this fault, they are more suitable for fault detection.) When $s = 1$, w_0 fails, because the state transition does not occur. Analogous behavior is exhibited in the presence of the $\bar{b}$-*sa-1* fault.

***w-sa-0* fault.** This fault alters the behavior of r, w_0, when $s = 1$, and w_1 when $s = 0$. Table A.3 describes the faulty behaviors. Since w is not asserted, r results in an

Table A.3: Faulty CAM cell behavior due to the *w-sa-0* fault.

Read operations			Write operations		
Description	$(s=0)$ $w\ b\bar{b}\ c\bar{c}\ m{\cdot}s$	$(s=1)$ $w\ b\bar{b}\ c\bar{c}\ m{\cdot}s$	Description	$w_1\ (s=0)$ $w\ b\bar{b}\ c\bar{c}\ m{\cdot}s$	$w_0\ (s=1)$ $w\ b\bar{b}\ c\bar{c}\ m{\cdot}s$
initial state	$0\ 11\ 00\ 1{\cdot}0$	$0\ 11\ 00\ 1{\cdot}1$	initial state	$0\ 11\ 00\ 1{\cdot}0$	$0\ 11\ 00\ 1{\cdot}1$
float $b/\bar{b}$	$0\ \tilde{1}\tilde{1}\ 00\ 1{\cdot}0$	$0\ \tilde{1}\tilde{1}\ 00\ 1{\cdot}1$	set $b/\bar{b}$	$0\ 10\ 00\ 1{\cdot}0$	$0\ 01\ 00\ 1{\cdot}1$
raise w	$\mathbf{0}\ \tilde{1}\tilde{1}\ 00\ 1{\cdot}0$	$\mathbf{0}\ \tilde{1}\tilde{1}\ 00\ 1{\cdot}1$	raise w	$\mathbf{0}\ 10\ 00\ 1{\cdot}0$	$\mathbf{0}\ 01\ 00\ 1{\cdot}1$
b or $\bar{b}$ discharges	$\mathbf{0}\ \tilde{\mathbf{1}}\tilde{1}\ 00\ 1{\cdot}0$	$\mathbf{0}\ \tilde{1}\tilde{\mathbf{1}}\ 00\ 1{\cdot}1$	new state	$\mathbf{0}\ 10\ 00\ 1{\cdot}\mathbf{0}$	$\mathbf{0}\ 01\ 00\ 1{\cdot}\mathbf{1}$
read $b/\bar{b}$ & lower w	$0\ \tilde{\mathbf{1}}\tilde{1}\ 00\ 1{\cdot}0$	$0\ \tilde{1}\tilde{\mathbf{1}}\ 00\ 1{\cdot}1$	lower w	$0\ 10\ 00\ 1{\cdot}\mathbf{0}$	$0\ 01\ 00\ 1{\cdot}\mathbf{1}$
raise $b/\bar{b}$	$0\ 11\ 00\ 1{\cdot}0$	$0\ 11\ 00\ 1{\cdot}1$	raise $b/\bar{b}$	$0\ 11\ 00\ 1{\cdot}\mathbf{0}$	$0\ 11\ 00\ 1{\cdot}\mathbf{1}$

unreliable output, as neither $b/\bar{b}$ is allowed to discharge. Operations w_1 when $s = 0$ and w_0 when $s = 1$ also fail, as a state transition does not occur.

***w-sa-1* fault.** The reader can verify using Table 7.1, that despite minor differences in the event-sequence model for this fault, all operations are correct in the FSM model of a single cell.

***c-sa-0* fault.** Again, this fault has no effect on r, w_0, w_1 and $w_\times$. As indicated in Table A.4, this fault affects only c_1 when $s = 0$. The fault prevents m from discharging resulting in an erroneous match. Analogous behavior is exhibited in the presence of the $\bar{c}$-*sa-0* fault.

Table A.4: Faulty CAM cell behavior due to the *c-sa-0* fault.

Compare operations ($s = 0$)	
Description	c_1 ($s = 0$) w $b\bar{b}$ $c\bar{c}$ $m{\cdot}s$
initial state	0 11 00 1·0
float m	0 11 00 $\tilde{1}$·0
set $c/\bar{c}$	0 11 **0**0 $\tilde{1}$·0
m discharges	0 11 **0**0 $\tilde{\mathbf{1}}$·0
read m & ground $c/\bar{c}$	0 11 00 $\tilde{\mathbf{1}}$·0
raise m	0 11 00 1·0

***c-sa-1* fault.** As shown in Table A.5, this fault only affects c_0 and $c_\times$ when $s = 0$. In this state the initial value of the m is I due to a conducting T_9 when m is driven

Table A.5: Faulty CAM cell behavior due to the *c-sa-1* fault.

Compare operations		
Description	c_0 ($s = 0$) w $b\bar{b}$ $c\bar{c}$ $m{\cdot}s$	$c_\times$ ($s = 0$) w $b\bar{b}$ $c\bar{c}$ $m{\cdot}s$
initial state	0 11 **1**0 **I**·0	0 11 **1**0 **I**·0
float m	0 11 **1**0 **0**·0	0 11 **1**0 **0**·0
set $c/\bar{c}$	0 11 **11** **0**·0	0 11 **1**0 **0**·0
m discharges	0 11 **11** **0**·0	0 11 **1**0 **0**·0
read m & ground $c/\bar{c}$	0 11 **1**0 **0**·0	0 11 **1**0 **0**·0
raise m	0 11 **1**0 **I**·0	0 11 **1**0 **I**·0

high by the peripheral circuitry. This fault also causes an increase in I_{DDQ}. Once m is floated, it is immediately discharged by the faulty line. Analogous behavior is exhibited in the presence of the $\bar{c}$-*sa-1* fault.

***m-sa-0* fault.** Table A.6 describes the faulty behaviors of a CAM cell in the presence of a *m-sa-0* fault. This fault only affects c_0 when $s = 0$, c_1 when $s = 1$, and $c_\times$ by returning an unconditional mismatch.

Table A.6: Faulty CAM cell behavior due to the *m-sa-0* fault.

	Compare operations			
Description	${}_0$ $(s = 0)$ w $b\bar{b}$ $c\bar{c}$ $m{\cdot}s$	$c_\times$ $(s = 0)$ w $b\bar{b}$ $c\bar{c}$ $m{\cdot}s$	c_1 $(s = 1)$ w $b\bar{b}$ $c\bar{c}$ $m{\cdot}s$	$c_\times$ $(s = 1)$ w $b\bar{b}$ $c\bar{c}$ $m{\cdot}s$
initial state	0 11 00 **0**·0	0 11 00 **0**·0	0 11 00 **0**·1	0 11 00 **0**·1
float m	0 11 00 **0**·0	0 11 00 **0**·0	0 11 00 **0**·1	0 11 00 **0**·1
set $c/\bar{c}$	0 11 01 **0**·0	0 11 00 **0**·0	0 11 10 **0**·1	0 11 00 **0**·1
m discharges	0 11 01 **0**·0	0 11 00 **0**·0	0 11 10 **0**·1	0 11 00 **0**·1
read m & ground $c/\bar{c}$	0 11 00 **0**·0	0 11 00 **0**·0	0 11 00 **0**·1	0 11 00 **0**·1
raise m	0 11 00 **0**·0	0 11 00 **0**·0	0 11 00 **0**·1	0 11 00 **0**·1

***m-sa-1* fault.** Table A.7 describes the faulty behaviors of a CAM cell in the presence of a *m-sa-1* fault. As expected, this fault only affects c_1 when $s = 0$ and c_0 when $s = 1$ by returning an unconditional match.

Table A.7: Faulty CAM cell behavior due to the *m-sa-1* fault.

	Compare operations	
Description	c_1 $(s = 0)$ w $b\bar{b}$ $c\bar{c}$ $m{\cdot}s$	c_0 $(s = 1)$ w $b\bar{b}$ $c\bar{c}$ $m{\cdot}s$
initial state	0 11 00 1·0	0 11 00 1·1
float m	0 11 00 **1**·0	0 11 00 **1**·1
set $c/\bar{c}$	0 11 10 **1**·0	0 11 01 **1**·1
m discharges	0 11 10 **1**·0	0 11 01 **1**·1
read m & ground $c/\bar{c}$	0 11 00 **1**·0	0 11 00 **1**·1
raise m	0 11 00 1·0	0 11 00 1·1

A.1.2 State Stuck-at Faults

***s-sa-0* fault.** Table A.8 describes the faulty behavior. Since s is permanently grounded, the cell can only only be in state, 0. As the voltages on $s/\bar{s}$ are fixed, r, c_0, c_1

Table A.8: Faulty CAM cell behavior due to the *s-sa-0* fault.

Write operations	
Description	w_1 $(s = 0)$ $w\ b\bar{b}\ c\bar{c}\ m{\cdot}s$
initial state	0 11 00 1·0
set $b/\bar{b}$	0 10 00 1·0
raise w	1 10 00 1·0
new state	1 10 00 1·**0**
lower w	0 10 00 1·**0**
raise $b/\bar{b}$	0 11 00 1·**0**

and $c_\times$ provide determinate, correct output. The only operation that fails is w_1. Identical behavior is exhibited by the *$\bar{s}$-sa-1* fault.

***s-sa-1* fault.** Analogously to the previous fault, the cell has only one state, 1, and the only operation that fails is w_0. Since the voltages on $s/\bar{s}$ are also fixed, r, c_0, c_1 and $c_\times$ provide determinate and correct output. Table A.9 describes the faulty behavior. Identical behavior is exhibited by the *$\bar{s}$-sa-0* fault.

Table A.9: Faulty CAM cell behavior due to the *s-sa-1* fault.

Write operations	
Description	w_0 $(s = 1)$ $w\ b\bar{b}\ c\bar{c}\ m{\cdot}s$
initial state	0 11 00 1·1
set $b/\bar{b}$	0 01 00 1·1
raise w	1 01 00 1·1
new state	1 01 00 1·**1**
lower w	0 01 00 1·**1**
raise $b/\bar{b}$	0 11 00 1·**1**

A.1.3 Transistor Faults

T_1-*on* fault. Although similar to the *s-sa-0* fault, the cell may temporarily remain in state I, and switch to state 0 after some time. In state I both T_1 and T_3 conduct; hence this fault is also detectable by I_{DD} tests. Table A.10 describes the faulty behaviors. During w_1, when $s = 0$, T_1 stays on after w has been asserted and

Table A.10: Faulty CAM cell behavior due to the T_1-*on* fault.

	Write operations		
	w_1 $(s=0)$	w_1 $(s=1)$	$w_\times$ $(s=1)$
Description	w $b\bar{b}$ $c\bar{c}$ $m{\cdot}s$	w $b\bar{b}$ $c\bar{c}$ $m{\cdot}s$	w $b\bar{b}$ $c\bar{c}$ $m{\cdot}s$
initial state	0 11 00 1·0	0 11 00 1·**I**	0 11 00 1·**I**
set $b/\bar{b}$	0 10 00 1·0	0 10 00 1·**I**	0 11 00 1·**I**
raise w	1 10 00 1·0	1 10 00 1·**I**	1 11 00 1·**I**
new state	1 10 00 1·**I**	1 10 00 1·**I**	1 11 00 1·**0**
lower w	0 10 00 1·**I**	0 10 00 1·**I**	0 11 00 1·**0**
raise $b/\bar{b}$	0 11 00 1·**I**|**0**	0 11 00 1·**I**|**0**	0 11 00 1·**0**

$\bar{s}$ driven to 0. Since T_3 is on and b is high at the same time, the voltage at s is intermediate. Once w is de-asserted the voltage at node s eventually becomes 0 again, as T_1 dominates T_3. The $w_\times$ accelerates this transition, by temporarily disabling T_3 and T_4, therefore allowing the faulty T_1 to discharge s. Analogous behavior is exhibited by the T_2-*on* fault.

T_1-*open* fault. In the presence of this fault, the cell cannot maintain state 0 and will flip to state 1. It may, however, remain in state I for a long time; hence this fault is a *data retention* fault. During w_0, s is pulled down solely by the bit-line driver. After w has been de-asserted, s floats; thus the T_2-T_4 inverter acts like a voltage divider. In effect, both $s/\bar{s}$ are in state I. A $w_\times$ forces both nodes to 1, which activates only T_2 (T_1 is open); hence $\bar{s}$ becomes 0 as soon as w is de-asserted. Table A.11 describes

Table A.11: Faulty CAM cell behavior due to the T_1-*open* fault.

	Write operations		
Description	w_0 $(s=0)$ w $b\bar{b}$ $c\bar{c}$ $m{\cdot}s$	$w_\times$ $(s=0)$ w $b\bar{b}$ $c\bar{c}$ $m{\cdot}s$	w_0 $(s=1)$ w $b\bar{b}$ $c\bar{c}$ $m{\cdot}s$
initial state	0 11 00 1·**I**	0 11 00 1·**I**	0 11 00 1·1
set $b/\bar{b}$	0 01 00 1·**I**	0 11 00 1·**I**	0 01 00 1·1
raise w	1 01 00 1·**I**	1 11 00 1·**I**	1 01 00 1·1
new state	1 01 00 1·**I**	1 11 00 1·**1**	1 01 00 1·**I**
lower w	0 01 00 1·**I**	0 11 00 1·**1**	0 01 00 1·**I**
raise $b/\bar{b}$	0 11 00 1·**I**|**1**	0 11 00 1·**1**	0 11 00 1·**I**|**1**

the faulty behaviors. Analogous behavior is exhibited by the T_2-*open* fault.

***T_3-on* fault.** In this case the cell exhibits anomalous behavior in what should be state 0, i.e., when T_3 does not normally conduct. Since T_3 is always on, the resulting state is I, which is detectable by I_{DDQ} tests. Because T_3 is usually weaker than T_1, the voltage on node s may be close to 0 immediately following a w_0. We assume, that the resistance of the faulty transistor is too large for this fault to be modeled by the *s-sa-1* fault, and the voltage on s will rise and remain at I. Since r in state I is unreliable, this fault is also a data retention fault. Table A.12 describes the faulty behaviors of a CAM cell in the presence of a T_3-*on* fault. During r, when

Table A.12: Faulty CAM cell behavior due to the T_3-*on* fault.

	Write operations	
Description	w_0 $(s=0)$ w $b\bar{b}$ $c\bar{c}$ $m{\cdot}s$	w_0 $(s=1)$ w $b\bar{b}$ $c\bar{c}$ $m{\cdot}s$
initial state	0 11 00 1·**I**	0 11 00 1·1
set $b/\bar{b}$	0 01 00 1·**I**	0 01 00 1·1
raise w	1 01 00 1·**I**	1 01 00 1·1
new state	1 01 00 1·0|**I**	1 01 00 1·0|**I**
lower w	0 01 00 1·0|**I**	0 01 00 1·0|**I**
raise $b/\bar{b}$	0 11 00 1·**I**	0 11 00 1·**I**

$s = I$, the differential voltages on the $b/\bar{b}$ lines will be smaller in magnitude; hence, depending on the sensitivity of the sense circuitry, r may yield an unpredictable response. Analogous behavior is exhibited by the T_4-*on* fault.

T_3-*open* fault. In the presence of this fault, the cell cannot maintain state 1 and will change to state 0. Since it may remain in state I for a long time, this is also a data retention fault. During w_1, s is pulled up by the bit-line driver. Since T_3 is

Table A.13: Faulty CAM cell behavior due to the T_3-*open* fault.

Write operations			
	w_1 $(s=0)$	w_1 $(s=1)$	$w_\times$ $(s=1)$
Description	w $b\bar{b}$ $c\bar{c}$ $m{\cdot}s$	w $b\bar{b}$ $c\bar{c}$ $m{\cdot}s$	w $b\bar{b}$ $c\bar{c}$ $m{\cdot}s$
initial state	0 11 00 1·0	0 11 00 1·1	0 11 00 1·**I**
set $b/\bar{b}$	0 10 00 1·0	0 10 00 1·1	0 11 00 1·**I**
raise w	1 10 00 1·0	1 10 00 1·1	1 11 00 1·**I**
new state	1 10 00 1·1	1 10 00 1·1	1 11 00 1·**I**
lower w	0 10 00 1·1	0 10 00 1·1	0 11 00 1·**0**
raise $b/\bar{b}$	0 11 00 1·**I**|**0**	0 11 00 1·**I**|**0**	0 11 00 1·**0**

open, T_1 acts like a DRAM cell; the charge stored at node s is dissipated due to leakage currents. Once the voltage on s drops below a certain threshold T_4 begins to conduct, raising the potential on $\bar{s}$. In turn, T_1 begins to conduct, and grounds s. A $w_\times$ forces both nodes to 1. After w is de-asserted, s will discharge more readily than $\bar{s}$ due to the fault; hence $s = 0$. Table A.13 describes the faulty behaviors. Analogous behavior is exhibited by the T_4-*open* fault.

T_5-*on* fault. This fault is similar to a *s-sa-1* fault. In a quiescent state b is driven high. Since T_5 is on, b is always connected to s; hence the cell always reverts to state 1, as indicated in Table A.14. However, the cell can temporarily find itself in state 0, which may affect any concurrent c_0 or c_1. This fault may also affect r of other cells

Table A.14: Faulty CAM cell behavior due to the T_5-*on* fault.

Write operations	
	w_0 $(s=1)$
Description	w $b\bar{b}$ $c\bar{c}$ $m{\cdot}s$
initial state	0 11 00 1·1
set $b/\bar{b}$	0 01 00 1·1
raise w	1 01 00 1·1
new state	1 01 00 1·0
lower w	0 01 00 1·0
raise $b/\bar{b}$	0 11 00 1·**1**

connected to the same $b/\bar{b}$ lines but different w lines, since b cannot be "floated" due to this fault. Analogous behavior is exhibited by the T_6-*on* fault.

T_5-***open*** **fault.** Here, s is permanently isolated from b. An r, when $s = 0$, will yield unreliable results, as b cannot discharge through s. When $s = 0$, w_1 also fails since

Table A.15: Faulty CAM cell behavior due to the T_5-*open* fault.

Read operations		Write operations		
	$(s=0)$		w_1 $(s=0)$	$w_\times$ $(s=1)$
Description	w $b\bar{b}$ $c\bar{c}$ $m{\cdot}s$	Description	w $b\bar{b}$ $c\bar{c}$ $m{\cdot}s$	w $b\bar{b}$ $c\bar{c}$ $m{\cdot}s$
initial state	0 11 00 1·0	initial state	0 11 00 1·0	0 11 00 1·1
float $b/\bar{b}$	0 $\tilde{1}\tilde{1}$ 00 1·0	set $b/\bar{b}$	0 10 00 1·0	0 11 00 1·1
raise w	1 $\tilde{1}\tilde{1}$ 00 1·0	raise w	1 10 00 1·0	1 11 00 1·1
b or $\bar{b}$ discharges	1 $\tilde{\mathbf{1}}\tilde{1}$ 00 1·0	new state	1 10 00 1·**I**	1 11 00 1·**0**
read $b/\bar{b}$ & lower w	0 $\tilde{\mathbf{1}}\tilde{1}$ 00 1·0	lower w	0 10 00 1·**I**	0 11 00 1·**0**
raise $b/\bar{b}$	0 11 00 1·0	raise $b/\bar{b}$	0 11 00 1·**0**	0 11 00 1·**0**

s is not driven high by b. Despite the conducting T_3, the change of state does not occur. However, $w_\times$, when $s = 1$ will cause the state to change, since a conducting T_1 will drive s to 0. These events are described in Table A.15. Analogous behavior is exhibited by the T_6-*open* fault.

***T_7-on* fault.** This fault affects c_1 when $s = 1$, i.e., when T_7 would normally be open. Table A.16 describes this behavior. As both T_7 and T_8 are on, they act as a voltage divider between $c/\bar{c}$. When the $c/\bar{c}$ are driven to $1/0$, the gate of T_9 is at potential I, roughly equal to $V_{dd}/2$. Since the threshold voltage V_T of T_9 is about $V_{dd}/6$, we assume that T_9 is on; hence, m will discharge. Analogous behavior is exhibited by the T_8-*on* fault.

Table A.16: Faulty CAM cell behavior due to the T_7-*on* fault.

Compare operations	
	c_1 $(s = 1)$
Description	w $b\bar{b}$ $c\bar{c}$ $m{\cdot}s$
initial state	0 11 00 1·1
float m	0 11 00 $\tilde{1}$·1
set $c/\bar{c}$	0 11 10 $\tilde{1}$·1
m discharges	0 11 10 **0**·1
read m & ground $c/\bar{c}$	0 11 00 **0**·1
raise m	0 11 00 1·1

***T_7-open* fault.** This fault affects c_1 in state 0, when T_7 would normally be on. Table A.17 describes the faulty behavior. Since T_7 is open, T_9 cannot be switched on when

Table A.17: Faulty CAM cell behavior due to the T_7-*open* fault.

Compare operations	
	c_1 $(s = 0)$
Description	w $b\bar{b}$ $c\bar{c}$ $m{\cdot}s$
initial state	0 11 00 1·0
float m	0 11 00 $\tilde{1}$·0
set $c/\bar{c}$	0 11 10 $\tilde{1}$·0
m discharges	0 11 10 $\tilde{\mathbf{1}}$·0
read m & ground $c/\bar{c}$	0 11 00 $\tilde{\mathbf{1}}$·0
raise m	0 11 00 1·0

$c = 1$, hence m is not discharged. Analogous behavior is exhibited by the T_8-*open* fault.

T_9-*on* fault. This fault affects c_0 when $s = 0$, c_1 when $s = 1$, and $c_\times$. Since T_9 is always

Table A.18: Faulty CAM cell behavior due to the T_9-*on* fault.

	Compare operations			
	c_0 $(s=0)$	$c_\times$ $(s=0)$	c_1 $(s=1)$	$c_\times$ $(s=1)$
Description	w $b\bar{b}$ $c\bar{c}$ $m{\cdot}s$	w $b\bar{b}$ $c\bar{c}$ $m{\cdot}s$	w $b\bar{b}$ $c\bar{c}$ $m{\cdot}s$	w $b\bar{b}$ $c\bar{c}$ $m{\cdot}s$
initial state	0 11 00 **0**·0	0 11 00 **0**·0	0 11 00 **0**·1	0 11 00 **0**·1
float m	0 11 00 **0**·0	0 11 00 **0**·0	0 11 00 **0**·1	0 11 00 **0**·1
set $c/\bar{c}$	0 11 01 **0**·0	0 11 00 **0**·0	0 11 10 **0**·1	0 11 00 **0**·1
m discharges	0 11 01 **0**·0	0 11 00 **0**·0	0 11 10 **0**·1	0 11 00 **0**·1
read m & ground $c/\bar{c}$	0 11 00 **0**·0	0 11 00 **0**·0	0 11 00 **0**·1	0 11 00 **0**·1
raise m	0 11 00 **0**·0	0 11 00 **0**·0	0 11 00 **0**·1	0 11 00 **0**·1

on, m is always connected to ground resulting in an unconditional mismatch, as shown in Table A.18. Also, in a quiescent state m is driven high; hence, this fault causes an increase in I_{DDQ}.

T_9-*open* fault. Here, c_0 and c_1 are affected, as shown in Table A.19. Since T_9 is open, m never discharges; hence, an unconditional match occurs.

Table A.19: Faulty CAM cell behavior due to the T_9-*open* fault.

	Compare operations	
	c_1 $(s=0)$	c_0 $(s=1)$
Description	w $b\bar{b}$ $c\bar{c}$ $m{\cdot}s$	w $b\bar{b}$ $c\bar{c}$ $m{\cdot}s$
initial state	0 11 00 1·0	0 11 00 1·1
float m	0 11 00 **1**·0	0 11 00 **1**·1
set $c/\bar{c}$	0 11 10 **1**·0	0 11 01 **1**·1
m discharges	0 11 10 **1**·0	0 11 01 **1**·1
read m & ground $c/\bar{c}$	0 11 00 **1**·0	0 11 00 **1**·1
raise m	0 11 00 1·0	0 11 00 1·1

A.1.4 Bridging Faults

Hard Bridging Faults

***b*-$\bar{b}$-*hrd* fault.** As indicated in Table A.20, this fault affects r, w_0 and w_1 in a manner that no reliable rest exists. During r both $b/\bar{b}$ are discharged, hence sense amplifiers return an unreliable output. Moreover, since s and $\bar{s}$ become shorted during this operation, the cell is forced into a metastable state. Since w_0 and w_1 also force the

Table A.20: Faulty CAM cell behavior due to the b-$\bar{b}$-*hrd* fault.

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

Write operations				
	w_0 $(s=0)$	w_1 $(s=0)$	w_0 $(s=1)$	w_1 $(s=1)$
Description	w $b\bar{b}$ $c\bar{c}$ $m{\cdot}s$	w $b\bar{b}$ $c\bar{c}$ $m{\cdot}s$	w $b\bar{b}$ $c\bar{c}$ $m{\cdot}s$	w $b\bar{b}$ $c\bar{c}$ $m{\cdot}s$
initial state	0 11 00 1·0	0 11 00 1·0	0 11 00 1·1	0 11 00 1·0
set $b/\bar{b}$	0 0**0** 00 1·0	0 **0**0 00 1·0	0 0**0** 00 1·1	0 **0**0 00 1·0
raise w	1 0**0** 00 1·0	1 **0**0 00 1·0	1 0**0** 00 1·1	1 **0**0 00 1·0
new state	1 0**0** 00 1·**I**	1 **0**0 00 1·**I**	1 0**0** 00 1·**I**	1 **0**0 00 1·**I**
lower w	0 0**0** 00 1·**I**	0 **0**0 00 1·**I**	0 0**0** 00 1·**I**	0 **0**0 00 1·**I**
raise $b/\bar{b}$	0 11 00 1·**1**$\vert$**0**	0 11 00 1·**1**$\vert$**0**	0 11 00 1·**1**$\vert$**0**	0 11 00 1·**1**$\vert$**0**

cell to a metastable state, there is no way to establish the next state of the cell. As a result, although $w_\times$, c_0, c_1 and $c_\times$ work correctly, it is impossible to generate a reliable faulty output, and hence this fault is not testable reliably. This fault causes an increase in I_{DD} during r, w_0 and w_1; hence it is detectable by parametric tests.

***b-w-hrd* fault.** Table A.21 describes the faulty behaviors. When $s = 0$, r will output a

Table A.21: Faulty CAM cell behavior due to the *b-w-hrd* fault.

Read operations		Write operations	
	$(s = 0)$		w_0 $(s = 1)$
Description	$w\ b\bar{b}\ c\bar{c}\ m{\cdot}s$	Description	$w\ b\bar{b}\ c\bar{c}\ m{\cdot}s$
initial state	$0\ \mathbf{01}\ 00\ 1{\cdot}0$	initial state	$0\ \mathbf{01}\ 00\ 1{\cdot}0$
float $b/\bar{b}$	$0\ \mathbf{0\tilde{1}}\ 00\ 1{\cdot}0$	set $b/\bar{b}$	$0\ 01\ 00\ 1{\cdot}0$
raise w	$1\ \mathbf{1\tilde{1}}\ 00\ 1{\cdot}0$	raise w	$\mathbf{0}\ 01\ 00\ 1{\cdot}0$
b or $\bar{b}$ discharges	$1\ \mathbf{10}\ 00\ 1{\cdot}\mathbf{1}$	new state	$\mathbf{0}\ 01\ 00\ 1{\cdot}\mathbf{1}$
read $b/\bar{b}$ & lower w	$0\ \mathbf{00}\ 00\ 1{\cdot}\mathbf{1}$	lower w	$0\ 01\ 00\ 1{\cdot}\mathbf{1}$
raise $b/\bar{b}$	$0\ \mathbf{01}\ 00\ 1{\cdot}\mathbf{1}$	raise $b/\bar{b}$	$0\ \mathbf{01}\ 00\ 1{\cdot}\mathbf{1}$

1 and the cell will change its state to 1. This behavior is the result of b being driven by w, instead of being floated high. When $s = 1$, w_0 will fail to change the state of the cell since w is grounded through b and the cell is never accessed. The *$\bar{b}$-w-hrd* fault exhibits analogous behavior.

***b-m-hrd* fault.** This fault alters the behavior of r, c_1 when $s = 0$, and c_0 when $s = 1$. Table A.22 describes the faulty behaviors. During r, b is driven through m to V_{dd}

Table A.22: Faulty CAM cell behavior due to the *b-m-hrd* fault.

Read operations		Compare operations		
	$(s = 0)$		c_1 $(s = 0)$	c_0 $(s = 1)$
Description	$w\ b\bar{b}\ c\bar{c}\ m{\cdot}s$	Description	$w\ b\bar{b}\ c\bar{c}\ m{\cdot}s$	$w\ b\bar{b}\ c\bar{c}\ m{\cdot}s$
initial state	$0\ 11\ 00\ 1{\cdot}0$	initial state	$0\ 11\ 00\ 1{\cdot}0$	$0\ 11\ 00\ 1{\cdot}1$
float $b/\bar{b}$	$0\ \mathbf{1\tilde{1}}\ 00\ 1{\cdot}0$	float m	$0\ 11\ 00\ \mathbf{1}{\cdot}0$	$0\ 11\ 00\ \mathbf{1}{\cdot}1$
raise w	$1\ \mathbf{1\tilde{1}}\ 00\ 1{\cdot}0$	set $c/\bar{c}$	$0\ 11\ 10\ \mathbf{1}{\cdot}0$	$0\ 11\ 00\ \mathbf{1}{\cdot}1$
b or $\bar{b}$ discharges	$1\ \mathbf{10}\ 00\ 1{\cdot}\mathbf{1}$	c discharges	$0\ 11\ 10\ \mathbf{1}{\cdot}0$	$0\ 11\ 00\ \mathbf{1}{\cdot}1$
read $b/\bar{b}$ & lower w	$0\ \mathbf{10}\ 00\ 1{\cdot}\mathbf{1}$	read m & ground $c/\bar{c}$	$0\ 11\ 00\ \mathbf{1}{\cdot}0$	$0\ 11\ 00\ \mathbf{1}{\cdot}1$
raise $b/\bar{b}$	$0\ 11\ 00\ 1{\cdot}\mathbf{1}$	raise m	$0\ 11\ 00\ 1{\cdot}0$	$0\ 11\ 00\ 1{\cdot}1$

instead of being floated high; hence the output is 1, and the cell changes its state from 0 to 1. During either mismatching 'compare' operation, m is driven to V_{dd} through b, and to ground through the conducting T_9. However, since the strength of the bit line driver is much greater than that of T_9, the output of m is assumed to be 1. Analogous behavior is exhibited by the $\bar{b}$-m-*hrd* fault.

c-$\bar{c}$-*hrd* fault. Table A.23 describes the faulty behaviors. As expected, this fault has no bearing on r, w_0, w_1 and $w_\times$, but it affects c_1 when $s = 0$ and c_0 when $s = 1$. During either mismatching 'compare' operation, both c and $\bar{c}$ are shorted to ground, hence m is not discharged.

Table A.23: Faulty CAM cell behavior due to the c-$\bar{c}$-*hrd* fault.

Compare operations		
Description	c_1 $(s=0)$ $w\ b\bar{b}\ c\bar{c}\ m{\cdot}s$	c_0 $(s=1)$ $w\ b\bar{b}\ c\bar{c}\ m{\cdot}s$
initial state	$0\ 11\ 00\ 1{\cdot}0$	$0\ 11\ 00\ 1{\cdot}1$
float m	$0\ 11\ 00\ \tilde{1}{\cdot}0$	$0\ 11\ 00\ \tilde{1}{\cdot}1$
set $c/\bar{c}$	$0\ 11\ \mathbf{0}0\ \tilde{1}{\cdot}0$	$0\ 11\ 0\mathbf{0}\ \tilde{1}{\cdot}1$
m discharges	$0\ 11\ \mathbf{0}0\ \tilde{\mathbf{1}}{\cdot}0$	$0\ 11\ 0\mathbf{0}\ \tilde{\mathbf{1}}{\cdot}1$
read m & ground $c/\bar{c}$	$0\ 11\ 00\ \tilde{\mathbf{1}}{\cdot}0$	$0\ 11\ 00\ \tilde{\mathbf{1}}{\cdot}1$
raise m	$0\ 11\ 00\ 1{\cdot}0$	$0\ 11\ 00\ 1{\cdot}1$

m-w-*hrd* fault. Table A.24 describes the faulty behaviors of a CAM cell in the presence of a m-w-*hrd* fault. This fault does not affect the behavior of r, w_0, w_1 or $w_\times$, since during their execution m is high and w can be raised. During c_0 when $s = 0$, c_1 when $s = 1$, and $c_\times$, m is always driven to ground through w; hence a match can never occur.

c-w-*hrd* fault. Table A.25 describes the faulty behaviors of a CAM cell in the presence of a c-w-*hrd* fault. From this table, we see that $b/\bar{b}$ remain floating high during r; hence the output is I. A w_1 when $s = 0$ and w_0 when $s = 1$ are affected as

Table A.24: Faulty CAM cell behavior due to the *m-w-hrd* fault.

Compare operations				
	c_0 $(s=0)$	$c_\times$ $(s=0)$	c_1 $(s=1)$	$c_\times$ $(s=1)$
Description	w $b\bar{b}$ $c\bar{c}$ $m{\cdot}s$	w $b\bar{b}$ $c\bar{c}$ $m{\cdot}s$	w $b\bar{b}$ $c\bar{c}$ $m{\cdot}s$	w $b\bar{b}$ $c\bar{c}$ $m{\cdot}s$
initial state	0 11 00 **0**·0	0 11 00 **0**·0	0 11 00 **0**·1	0 11 00 **0**·1
float m	0 11 00 **0**·0	0 11 00 **0**·0	0 11 00 **0**·1	0 11 00 **0**·1
set $c/\bar{c}$	0 11 01 **0**·0	0 11 00 **0**·0	0 11 10 **0**·1	0 11 00 **0**·1
m discharges	0 11 01 **0**·0	0 11 00 **0**·0	0 11 10 **0**·1	0 11 00 **0**·1
read m & ground $c/\bar{c}$	0 11 00 **0**·0	0 11 00 **0**·0	0 11 00 **0**·1	0 11 00 **0**·1
raise m	0 11 00 **0**·0	0 11 00 **0**·0	0 11 00 **0**·1	0 11 00 **0**·1

well, since the state transitions do not occur. A c_1 when $s = 0$ also fails, since c is grounded by w and fails to discharge m through T_9. This fault causes an increase if I_{DD} during r, w_0, w_1, $w_\times$ and c_1; hence is detectable by parametric tests. The $\bar{c}$-*w-hrd* fault manifests analogous behavior.

***c-m-hrd* fault.** Table A.26 describes the faulty behaviors of a CAM cell in the presence of a *c-m-hrd* fault. This fault affects the behavior of c_0 when $s = 0$ and $c_\times$, during which m is erroneously discharged. Analogous behavior is exhibited by the $\bar{c}$-*m-hrd* fault.

***s-$\bar{s}$-hrd* fault.** This fault affects all operations except $c_\times$. Since nodes s and $\bar{s}$ are shorted together the expected voltage on these nodes is about $V_{dd}/2$, denoted by the sole state I. An r generates an indeterminate output I, as $b/\bar{b}$ equally discharge to $V_{dd}/2$. All 'write' operations also fail. Since nodes s and $\bar{s}$ are at potential $V_{dd}/2$, transistors T_7 and T_8 conduct, as theit threshold voltage $V_{th} = V_{dd}/6$. Consequently, c_0 and c_1 always return a mismatch. Table A.27 describes the faulty behaviors.

***$\bar{b}'$-b-hrd* fault.** Note that for notational simplicity we use b', $\bar{b}'$, c', $\bar{c}'$ to denote input lines of cell $j+1$, and b, $\bar{b}$, c, $\bar{c}$ to denote those of cell j.

Table A.25: Faulty CAM cell behavior due to the *c-w-hrd* fault.

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

Write operations			Compare operations	
Description	$w_1\ (s=0)$ $w\ b\bar{b}\ c\bar{c}\ m{\cdot}s$	$w_0\ (s=1)$ $w\ b\bar{b}\ c\bar{c}\ m{\cdot}s$	Description	$c_1\ (s=0)$ $w\ b\bar{b}\ c\bar{c}\ m{\cdot}s$
initial state	$0\ 11\ 00\ 1{\cdot}0$	$0\ 11\ 00\ 1{\cdot}1$	initial state	$0\ 11\ 00\ 1{\cdot}0$
set $b/\bar{b}$	$0\ 10\ 00\ 1{\cdot}0$	$0\ 01\ 00\ 1{\cdot}1$	float m	$0\ 11\ 00\ \tilde{1}{\cdot}0$
raise w	$\mathbf{0}\ 10\ 00\ 1{\cdot}0$	$\mathbf{0}\ 01\ 00\ 1{\cdot}1$	set $c/\bar{c}$	$0\ 11\ 00\ \tilde{1}{\cdot}0$
new state	$\mathbf{0}\ 10\ 00\ 1{\cdot}\mathbf{0}$	$\mathbf{0}\ 01\ 00\ 1{\cdot}\mathbf{1}$	m discharges	$0\ 11\ 00\ \tilde{\mathbf{1}}{\cdot}0$
lower w	$0\ 10\ 00\ 1{\cdot}\mathbf{0}$	$0\ 01\ 00\ 1{\cdot}\mathbf{1}$	read m & ground $c/\bar{c}$	$0\ 11\ 00\ \tilde{\mathbf{1}}{\cdot}0$
raise $b/\bar{b}$	$0\ 11\ 00\ 1{\cdot}\mathbf{0}$	$0\ 11\ 00\ 1{\cdot}\mathbf{1}$	raise m	$0\ 11\ 00\ 1{\cdot}0$

This inter-cell fault manifests itself during r, when the two neighboring cells store the same value, i.e., their composite state $S = 00$ or $S = 11$. Suppose $S = 00$. During r, $\bar{b}'$ should remain floating high, while b is driven to ground. Due to the hard short, bothlines discharge, hence the sense amplifier for cell $j+1$ will return an unreliable output I, as both $b'/\bar{b}'$ are 0. An r when $S = 11$ exhibits analogous behavior. Table A.28 describes the faulty 'read' operations. This fault also exhibits erroneous behavior during w_{00} and w_{11} from any possible state, as described in Table A.29. Here, the arbitrary state is denoted as $S = \times\times$. For each of these operations their respective final state is always the same. Table A.30 describes 'write' operations whose faulty behaviors occur only in specific states. These operations are: $w_{\times 0}$ when $S = 00$ or $S = 01$ and $w_{1\times}$ when $S = 01$ or

Table A.26: Faulty CAM cell behavior due to the *c-m-hrd* fault.

Compare operations			
	c_0 $(s=0)$	$c_\times$ $(s=0)$	$c_\times$ $(s=1)$
Description	w $b\bar{b}$ $c\bar{c}$ $m{\cdot}s$	w $b\bar{b}$ $c\bar{c}$ $m{\cdot}s$	w $b\bar{b}$ $c\bar{c}$ $m{\cdot}s$
initial state	0 11 00 **0**·0	0 11 00 **0**·0	0 11 00 **0**·1
float m	0 11 00 **0**·0	0 11 00 **0**·0	0 11 00 **0**·1
set $c/\bar{c}$	0 11 01 **0**·0	0 11 00 **0**·0	0 11 00 **0**·1
m discharges	0 11 01 **0**·0	0 11 00 **0**·0	0 11 00 **0**·1
read m & ground $c/\bar{c}$	0 11 00 **0**·0	0 11 00 **0**·0	0 11 00 **0**·1
raise m	0 11 00 **0**·0	0 11 00 **0**·0	0 11 00 **0**·1

$S = 11$. This fault does not affect any of the 'compare' operations.

$\bar{c}'$-*c-hrd* fault. This inter-cell fault manifests itself only when the discharging of m depends on the two bridged lines, i.e., in states $S = 00$, $S = 11$ and $S = 10$. When $S = 00$ $c_{\times 1}$ fails, when $S = 11$ $c_{0\times}$. and when $S = 10$ c_{00}, c_{11}, $c_{0\times}$ and $c_{\times 1}$. In each case a faulty match is produced. Table A.31 describes the faulty operations. Note that for notational simplicity we use b', $\bar{b}'$, c', $\bar{c}'$ to denote input lines of cell $j+1$, and b, $\bar{b}$, c, $\bar{c}$ to denote those of cell j. A $C_{\times 1}$ when $S = 00$ fails, because c is grounded by $\bar{c}'$; hence m is not discharged. The other 'compare' operations fail in a similar manner.

Table A.27: Faulty CAM cell behavior due to the *s*-$\bar{s}$-*hrd* fault.

Read operations		Compare operations		
			c_1	c_0
Description	$w\ b\bar{b}\ c\bar{c}\ m{\cdot}s$	Description	$w\ b\bar{b}\ c\bar{c}\ m{\cdot}s$	$w\ b\bar{b}\ c\bar{c}\ m{\cdot}s$
initial state	$0\ 11\ 00\ 1{\cdot}\mathbf{I}$	initial state	$0\ 11\ 00\ 1{\cdot}\mathbf{I}$	$0\ 11\ 00\ 1{\cdot}\mathbf{I}$
float $b/\bar{b}$	$0\ \tilde{1}\tilde{1}\ 00\ 1{\cdot}\mathbf{I}$	float m	$0\ 11\ 00\ \tilde{1}{\cdot}\mathbf{I}$	$0\ 11\ 00\ \tilde{1}{\cdot}\mathbf{I}$
raise w	$1\ \tilde{1}\tilde{1}\ 00\ 1{\cdot}\mathbf{I}$	set $c/\bar{c}$	$0\ 11\ 10\ \tilde{1}{\cdot}\mathbf{I}$	$0\ 11\ 01\ \tilde{1}{\cdot}\mathbf{I}$
b or $\bar{b}$ discharges	$1\ \tilde{\mathbf{1}}\tilde{\mathbf{1}}\ 00\ 1{\cdot}\mathbf{I}$	c discharges	$0\ 11\ 10\ \mathbf{0}{\cdot}\mathbf{I}$	$0\ 11\ 01\ \mathbf{0}{\cdot}\mathbf{I}$
read $b/\bar{b}$ & lower w	$0\ \tilde{\mathbf{1}}\tilde{\mathbf{1}}\ 00\ 1{\cdot}\mathbf{I}$	read m & ground $c/\bar{c}$	$0\ 11\ 00\ \mathbf{0}{\cdot}\mathbf{I}$	$0\ 11\ 00\ \mathbf{0}{\cdot}\mathbf{I}$
raise $b/\bar{b}$	$0\ 11\ 00\ 1{\cdot}\mathbf{I}$	raise m	$0\ 11\ 00\ 1{\cdot}\mathbf{I}$	$0\ 11\ 00\ 1{\cdot}\mathbf{I}$

Write operations			
	w_0	w_1	$w_\times$
Description	$w\ b\bar{b}\ c\bar{c}\ m{\cdot}s$	$w\ b\bar{b}\ c\bar{c}\ m{\cdot}s$	$w\ b\bar{b}\ c\bar{c}\ m{\cdot}s$
initial state	$0\ 11\ 00\ 1{\cdot}\mathbf{I}$	$0\ 11\ 00\ 1{\cdot}\mathbf{I}$	$0\ 11\ 00\ 1{\cdot}\mathbf{I}$
set $b/\bar{b}$	$0\ 01\ 00\ 1{\cdot}\mathbf{I}$	$0\ 10\ 00\ 1{\cdot}\mathbf{I}$	$0\ 11\ 00\ 1{\cdot}\mathbf{I}$
raise w	$1\ 01\ 00\ 1{\cdot}\mathbf{I}$	$1\ 10\ 00\ 1{\cdot}\mathbf{I}$	$1\ 11\ 00\ 1{\cdot}\mathbf{I}$
new state	$1\ 01\ 00\ 1{\cdot}\mathbf{I}$	$1\ 10\ 00\ 1{\cdot}\mathbf{I}$	$1\ 11\ 00\ 1{\cdot}\mathbf{I}$
lower w	$0\ 01\ 00\ 1{\cdot}\mathbf{I}$	$0\ 10\ 00\ 1{\cdot}\mathbf{I}$	$0\ 11\ 00\ 1{\cdot}\mathbf{I}$
raise $b/\bar{b}$	$0\ 11\ 00\ 1{\cdot}\mathbf{I}$	$0\ 11\ 00\ 1{\cdot}\mathbf{I}$	$0\ 11\ 00\ 1{\cdot}\mathbf{I}$

Resistive Bridging Faults

***b*-$\bar{b}$-*res* fault.** This fault has no functional effect on 'read', 'write' or 'compare' operations. An example of how marginally the behavior of a 'read' is affected is given in Table A.32. When w is asserted b is driven to an opposite logical value than $\bar{b}$, as they are connected to $s/\bar{s}$, respectively. The potential difference between these two lines is dissipated on the resistive bridge, which allows the $b/\bar{b}$ to be read correctly. After $b/\bar{b}$ are read and w lowered, $\bar{b}$ begins to discharge, but that has no functional significance. During r, w_0 and w_1, as $b/\bar{b}$ are driven to opposite logical values, there will be an increase in I_{DD} that is inversely proportional to the resistance of the bridge; hence this fault can be detected with parametric tests.

***b*-*w*-*res* fault.** Just as the fault discussed above, this fault also does not affect the functional behavior of the CAM cell and hence, is not detectable by functional tests.

Table A.28: Faulty 'read' behavior due to the $\bar{b}'$-b-*hrd* fault.

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

Table A.29: State-independent faulty 'write' behavior due to the $\bar{b}'$-b-*hrd* fault.

Write operations	w_{00} $(S = \times\times)$						w_{11} $(S = \times\times)$					
Description	w	$b\bar{b}$	$c\bar{c}$	$b\bar{b}$	$c\bar{c}$	$m{\cdot}S$	w	$b\bar{b}$	$c\bar{c}$	$b\bar{b}$	$c\bar{c}$	$m{\cdot}S$
initial state	0	11	00	11	00	$1{\cdot}\times\times$	0	11	00	11	00	$1{\cdot}\times\times$
set $b/\bar{b}$	0	$0\mathbf{0}$	00	01	00	$1{\cdot}\times\times$	0	10	00	$\mathbf{0}0$	00	$1{\cdot}\times\times$
raise w	1	$0\mathbf{0}$	00	01	00	$1{\cdot}\times\times$	1	10	00	$\mathbf{0}0$	00	$1{\cdot}\times\times$
new state	1	$0\mathbf{0}$	00	01	00	$1{\cdot}\mathbf{I}0$	1	10	00	$\mathbf{0}0$	00	$1{\cdot}1\mathbf{I}$
lower w	0	$0\mathbf{0}$	00	01	00	$1{\cdot}\mathbf{I}0$	0	10	00	$\mathbf{0}0$	00	$1{\cdot}1\mathbf{I}$
raise $b/\bar{b}$	0	11	00	11	00	$1{\cdot}\mathbf{0}\vert\mathbf{1}0$	0	11	00	11	00	$1{\cdot}1\mathbf{0}\vert\mathbf{1}$

Increased I_{DD} can be observed during operations where b and w are at different voltage potential, i.e., during r when $s = 0$, and during w_0. An increase in I_{DDQ} can also be observed. The $\bar{b}$-w-*res* fault exhibits analogous behavior.

***b-m-res* fault.** This fault does not alter the functional behavior of the cell. Increased I_{DD} is expected when b and m have opposing logical values, i.e., during r, c_1 when $s = 0$, c_0 when $s = 1$ and w_0. Analogous behavior is exhibited by the $\bar{b}$-m-*res* fault.

***c-$\bar{c}$-res* fault.** This fault does not affect any operations on the cell, since due to the bridging resistance, it is possible to drive $c/\bar{c}$ to opposite logical values. This implies that this fault is not detectable by means of functional tests. However, the increase in I_{DD}, during c_1 when $s = 0$ and c_0 when $s = 1$ make this fault detectable by

Table A.30: State-specific faulty 'write' behavior due to the $\bar{b}'$-b-*hrd* fault.

Write operations												
	$w_{\times 0}$ $(S = 00)$						$w_{1\times}$ $(S = 11)$					
Description	w	$b\bar{b}$	$c\bar{c}$	$b\bar{b}$	$c\bar{c}$	$m{\cdot}S$	w	$b\bar{b}$	$c\bar{c}$	$b\bar{b}$	$c\bar{c}$	$m{\cdot}S$
initial state	0	11	00	11	00	1·00	0	11	00	11	00	1·11
set $b/\bar{b}$	0	**10**	00	01	00	1·00	0	10	00	**01**	00	1·11
raise w	1	**10**	00	01	00	1·00	1	10	00	**01**	00	1·11
new state	1	**10**	00	01	00	1·**10**	1	10	00	**01**	00	1·**10**
lower w	0	**10**	00	01	00	1·**10**	0	10	00	**01**	00	1·**10**
raise $b/\bar{b}$	0	11	00	11	00	1·**10**	0	11	00	11	00	1·**10**
Write operations												
	$w_{\times 0}$ $(S = 01)$						$w_{1\times}$ $(s = 01)$					
Description	w	$b\bar{b}$	$c\bar{c}$	$b\bar{b}$	$c\bar{c}$	$m{\cdot}S$	w	$b\bar{b}$	$c\bar{c}$	$b\bar{b}$	$c\bar{c}$	$m{\cdot}S$
initial state	0	11	00	11	00	1·01	0	11	00	11	00	1·01
set $b/\bar{b}$	0	**10**	00	01	00	1·01	0	10	00	**01**	00	1·01
raise w	1	**10**	00	01	00	1·01	1	10	00	**01**	00	1·01
new state	1	**10**	00	01	00	1·**10**	1	10	00	**01**	00	1·**10**
lower w	0	**10**	00	01	00	1·**10**	0	10	00	**01**	00	1·**10**
raise $b/\bar{b}$	0	11	00	11	00	1·10	0	11	00	11	00	1·**10**

parametric tests.

***m-w-res* fault.** Table A.33 describes the faulty behaviors of a CAM cell in the presence of a *m-w-res* fault. This fault does not affect the behavior of r, w_0, w_1 or $w_\times$, as the resistive bridge allows m and w to be at different voltage levels ath the cost if higher I_{DD}. During c_0 when $s = 0$, c_1 when $s = 1$, and $c_\times$, m is always discharged to ground through the bridging resistance; hence a match can never occur.

***c-w-res* fault.** In the presence of this fault it is possible to drive c and w to opposite logical values and hence any operation where such a condition occurs will be functionally correct. This implies that this fault is not detectable by means of functional tests. However, the increase in I_{DD} during r, w_0, w_1, $w_\times$ and c_1 makes this fault detectable by parametric tests. The $\bar{c}$-*w-res* fault manifests analogous behavior.

Table A.31: Faulty CAM cell pair behavior due to the $\bar{c}'$-c-*hrd* fault.

Compare operations		
	$c_{\times 1}$ $(S = 00)$	$c_{0\times}$ $(s = 11)$
Description	w $b\bar{b}$ $c\bar{c}$ $b\bar{b}$ $c\bar{c}$ $m{\cdot}S$	w $b\bar{b}$ $c\bar{c}$ $b\bar{b}$ $c\bar{c}$ $m{\cdot}S$
initial state	0 11 00 11 00 1·00	0 11 00 11 00 1·11
float m	0 11 00 11 00 $\tilde{1}$·00	0 11 00 11 00 $\tilde{1}$·11
set $c/\bar{c}$	0 11 00 11 **00** $\tilde{1}$·00	0 11 0**0** 11 00 $\tilde{1}$·11
m discharges	0 11 00 11 **00** $\tilde{\mathbf{1}}$·00	0 11 0**0** 11 00 $\tilde{\mathbf{1}}$·11
read m & ground $c/\bar{c}$	0 11 00 11 00 $\tilde{\mathbf{1}}$·00	0 11 00 11 00 $\tilde{\mathbf{1}}$·11
raise m	0 11 00 11 00 1·00	0 11 00 11 00 1·11
	c_{00} $(S = 10)$	$c_{0\times}$ $(s = 10)$
Description	w $b\bar{b}$ $c\bar{c}$ $b\bar{b}$ $c\bar{c}$ $m{\cdot}S$	w $b\bar{b}$ $c\bar{c}$ $b\bar{b}$ $c\bar{c}$ $m{\cdot}S$
initial state	0 11 00 11 00 1·10	0 11 00 11 00 1·10
float m	0 11 00 11 00 $\tilde{1}$·10	0 11 00 11 00 $\tilde{1}$·10
set $c/\bar{c}$	0 11 0**0** 11 01 $\tilde{1}$·10	0 11 0**0** 11 00 $\tilde{1}$·10
m discharges	0 11 0**0** 11 01 $\tilde{\mathbf{1}}$·10	0 11 0**0** 11 00 $\tilde{\mathbf{1}}$·10
read m & ground $c/\bar{c}$	0 11 00 11 00 $\tilde{\mathbf{1}}$·10	0 11 00 11 00 $\tilde{\mathbf{1}}$·10
raise m	0 11 00 11 00 1·10	0 11 00 11 00 1·10
	c_{11} $(S = 10)$	$c_{\times 1}$ $(s = 10)$
Description	w $b\bar{b}$ $c\bar{c}$ $b\bar{b}$ $c\bar{c}$ $m{\cdot}S$	w $b\bar{b}$ $c\bar{c}$ $b\bar{b}$ $c\bar{c}$ $m{\cdot}S$
initial state	0 11 00 11 00 1·10	0 11 00 11 00 1·10
float m	0 11 00 11 00 $\tilde{1}$·10	0 11 00 11 00 $\tilde{1}$·10
set $c/\bar{c}$	0 11 10 11 **00** $\tilde{1}$·10	0 11 00 11 **00** $\tilde{1}$·10
m discharges	0 11 10 11 **00** $\tilde{\mathbf{1}}$·10	0 11 00 11 **00** $\tilde{\mathbf{1}}$·10
read m & ground $c/\bar{c}$	0 11 00 11 00 $\tilde{\mathbf{1}}$·10	0 11 00 11 00 $\tilde{\mathbf{1}}$·10
raise m	0 11 00 11 00 1·10	0 11 00 11 00 1·10

***c-m-res* fault.** Table A.34 describes the faulty behaviors of a CAM cell in the presence of a *c-m-res* fault. This fault affects the behavior of c_0 when $s = 0$ and $c_\times$, during which m is erroneously discharged. In the initial state, m is driven to V_{dd} and c is driven to ground, hence there is a potential drop of V_{dd} across the resistive bridge, indicating an increase in I_{DDQ}. However, sa soon as m is disconnected from its driver, it begins to discharge through the bridging resistance; the rate of discharge is dependent on the bridging resistance and the capacitance of m. Analogous behavior is exhibited by the $\bar{c}$-*m-res* fault.

Table A.32: Faulty CAM cell behavior due to the *b*-$\bar{b}$-*res* fault.

Read operations		
Description	$(s = 0)$ $w\ b\bar{b}\ c\bar{c}\ m{\cdot}s$	$(s = 1)$ $w\ b\bar{b}\ c\bar{c}\ m{\cdot}s$
initial state	0 11 00 1·0	0 11 00 1·1
float $b/\bar{b}$	0 $\tilde{1}\tilde{1}$ 00 1·0	0 $\tilde{1}\tilde{1}$ 00 1·1
raise w	1 $\tilde{1}\tilde{1}$ 00 1·0	1 $\tilde{1}\tilde{1}$ 00 1·1
b or $\bar{b}$ discharges	1 01 00 1·0	1 10 00 1·1
read $b/\bar{b}$ & lower w	0 $\tilde{0}\tilde{\mathbf{1}}$ 00 1·0	0 $\tilde{\mathbf{1}}\tilde{0}$ 00 1·1
raise $b/\bar{b}$	0 11 00 1·0	0 11 00 1·1

Table A.33: Faulty CAM cell behavior due to the *m*-*w*-*res* fault.

Compare operations				
Description	c_0 $(s = 0)$ $w\ b\bar{b}\ c\bar{c}\ m{\cdot}s$	$c_\times$ $(s = 0)$ $w\ b\bar{b}\ c\bar{c}\ m{\cdot}s$	c_1 $(s = 1)$ $w\ b\bar{b}\ c\bar{c}\ m{\cdot}s$	$c_\times$ $(s = 1)$ $w\ b\bar{b}\ c\bar{c}\ m{\cdot}s$
initial state	0 11 00 **0**·0	0 11 00 **0**·0	0 11 00 **0**·1	0 11 00 **0**·1
float m	0 11 00 **I**·0	0 11 00 **I**·0	0 11 00 **I**·1	0 11 00 **I**·1
set $c/\bar{c}$	0 11 01 **0**·0	0 11 00 **0**·0	0 11 10 **0**·1	0 11 00 **0**·1
m discharges	0 11 01 **0**·0	0 11 00 **0**·0	0 11 10 **0**·1	0 11 00 **0**·1
read m & ground $c/\bar{c}$	0 11 00 **0**·0	0 11 00 **0**·0	0 11 00 **0**·1	0 11 00 **0**·1
raise m	0 11 00 **0**·0	0 11 00 **0**·0	0 11 00 **0**·1	0 11 00 **0**·1

***s*-$\bar{s}$-*res* fault.** This fault does not affect the functional behavior of any of the operations. Since nodes s and $\bar{s}$ are connected together by a resistive bridge, the potential difference between the two nodes is dissipated across this resistive bridge, which results in an increase in I_{DDQ}.

$\bar{b}'$-*b*-*res* fault. Note that for notational simplicity we use b', $\bar{b}'$, c', $\bar{c}'$ to denote input lines of cell $j+1$, and b, $\bar{b}$, c, $\bar{c}$ to denote those of cell j.

This inter-cell fault does not manifest itself during any of the nineteen operations. When $\bar{b}'$ and b are driven to opposite logical values, the potential difference between the two nodes is dissipated across this resistive bridge. Hence an increased I_{DD} can be observed during r when $S = 00$ or $S = 11$, w_{00}, $w_{\times 0}$, w_{11} and $w_{1\times}$.

Table A.34: Faulty CAM cell behavior due to the *c-m-res* fault.

Compare operations			
	c_0 $(s=0)$	$c_\times$ $(s=0)$	$c_\times$ $(s=1)$
Description	w $b\bar{b}$ $c\bar{c}$ $m{\cdot}s$	w $b\bar{b}$ $c\bar{c}$ $m{\cdot}s$	w $b\bar{b}$ $c\bar{c}$ $m{\cdot}s$
initial state	0 11 00 **0**·0	0 11 00 **0**·0	0 11 00 **0**·1
float m	0 11 00 **I**·0	0 11 00 **I**·0	0 11 00 **I**·1
set $c/\bar{c}$	0 11 01 **0**·0	0 11 00 **0**·0	0 11 00 **0**·1
m discharges	0 11 01 **0**·0	0 11 00 **0**·0	0 11 00 **0**·1
read m & ground $c/\bar{c}$	0 11 00 **0**·0	0 11 00 **0**·0	0 11 00 **0**·1
raise m	0 11 00 **0**·0	0 11 00 **0**·0	0 11 00 **0**·1

***$\bar{c}'$-c-res* fault.** This inter-cell fault does not alter the functional behavior of any of the operations. It will result in an increase in I_{DD} during c_{00}, $c_{0\times}$, c_{11} and $c_{\times 1}$; hence, this fault can be detected by parametric tests.

A.2 CAM Test Verification

In this section the T_{cell} test for a single cell is verified. In Table A.35 the fault-free response is provided for comparison with the faulty responses of all independently testable faults. The deviations from the fault-free response are indicated in boldface. Tests for individual faults that are included in T_{cell} are also given. In all cases except *w-sa-0* the initial state of the cell is not important because, after the execution of the first three write operations $w_0w_1w_\times$, the initial state is forgotten. Tests for some of the faults (ex. *b-sa-1*: $w_1w_0c_0$) appear separated by other operations. These interleaved operations are combinations of $w_\times$, c_0 and c_1 which, for these faults, do not change the intended state of the cell and, therefore, do not invalidate these tests.

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

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

Table A.35: Verification of the T_{cell} test.

Fault	T_{cell}	Test Sequence for Faults within T_{cell}
	$w_0\,w_1\,w_\times\,c_1\,c_0\,w_0\,w_\times\,c_0\,c_1$	
fault-free cell	— — — 10 — — 10	
b-sa-0	— — — **01** — — 10	$w_0\,\mathbf{w_1}\,\mathbf{w}_\times\,\mathbf{c_1}\,c_0\,w_0\,w_\times\,c_0\,c_1$
b-sa-1	— — — 10 — — **01**	$w_0\,\mathbf{w_1}\,w_\times\,c_1\,c_0\,\mathbf{w_0}\,w_\times\,\mathbf{c_0}\,c_1$
$\bar{b}$-*sa-0*	— — — 10 — — **01**	$w_0\,w_1\,w_\times\,c_1\,c_0\,\mathbf{w_0}\,\mathbf{w}_\times\,\mathbf{c_0}\,c_1$
$\bar{b}$-*sa-1*	— — — **01** — — 10	$\mathbf{w_0}\,\mathbf{w_1}\,w_\times\,\mathbf{c_1}\,c_0\,w_0\,w_\times\,c_0\,c_1$
w-sa-0 (s=0)	— — — **01** — — 10	$w_0\,\mathbf{w_1}\,w_\times\,\mathbf{c_1}\,c_0\,\mathbf{w_0}\,w_\times\,\mathbf{c_0}\,c_1$
w-sa-0 (s=1)	— — — 10 — — **01**	″
c-sa-0	— — — 10 — — **11**	$w_0\,w_1\,w_\times\,c_1\,c_0\,\mathbf{w_0}\,w_\times\,c_0\,\mathbf{c_1}$
c-sa-1	— — — 10 — — 00	$w_0\,w_1\,w_\times\,c_1\,c_0\,\mathbf{w_0}\,w_\times\,\mathbf{c_0}\,c_1$
$\bar{c}$-*sa-0*	— — — **11** — — 10	$w_0\,\mathbf{w_1}\,w_\times\,c_1\,\mathbf{c_0}\,w_0\,w_\times\,c_0\,c_1$
$\bar{c}$-*sa-1*	— — — **0**0 — — 10	$w_0\,\mathbf{w_1}\,w_\times\,\mathbf{c_1}\,c_0\,w_0\,w_\times\,c_0\,c_1$
m-sa-0	— — — **0**0 — — **0**0	$w_0\,w_1\,w_\times\,\mathbf{c_1}\,\mathbf{c_0}\,w_0\,w_\times\,c_0\,c_1$
m-sa-1	— — — **11** — — **11**	$w_0\,w_1\,w_\times\,\mathbf{c_1}\,\mathbf{c_0}\,w_0\,w_\times\,c_0\,c_1$
s-sa-0	— — — **01** — — 10	$w_0\,\mathbf{w_1}\,w_\times\,\mathbf{c_1}\,c_0\,w_0\,w_\times\,c_0\,c_1$
s-sa-1	— — — 10 — — **01**	$w_0\,w_1\,w_\times\,c_1\,c_0\,\mathbf{w_0}\,w_\times\,\mathbf{c_0}\,c_1$
$\bar{s}$-*sa-0*	— — — 10 — — **01**	$w_0\,w_1\,w_\times\,c_1\,c_0\,\mathbf{w_0}\,w_\times\,\mathbf{c_0}\,c_1$
$\bar{s}$-*sa-1*	— — — **01** — — 10	$w_0\,\mathbf{w_1}\,w_\times\,\mathbf{c_1}\,c_0\,w_0\,w_\times\,c_0\,c_1$
T_1-*on*	— — — **01** — — 10	$w_0\,\mathbf{w_1}\,\mathbf{w}_\times\,\mathbf{c_1}\,c_0\,w_0\,w_\times\,c_0\,c_1$
T_1-*open*	— — — 10 — — **01**	$w_0\,w_1\,w_\times\,c_1\,c_0\,\mathbf{w_0}\,\mathbf{w}_\times\,\mathbf{c_0}\,c_1$
T_2-*on*	— — — 10 — — **01**	$w_0\,w_1\,w_\times\,c_1\,c_0\,\mathbf{w_0}\,\mathbf{w}_\times\,\mathbf{c_0}\,c_1$
T_2-*open*	— — — **01** — — 10	$w_0\,\mathbf{w_1}\,\mathbf{w}_\times\,\mathbf{c_1}\,c_0\,w_0\,w_\times\,c_0\,c_1$
T_3-*open*	— — — **01** — — 10	$w_0\,\mathbf{w_1}\,\mathbf{w}_\times\,\mathbf{c_1}\,c_0\,w_0\,w_\times\,c_0\,c_1$
T_4-*open*	— — — 10 — — **01**	$w_0\,w_1\,w_\times\,c_1\,c_0\,\mathbf{w_0}\,\mathbf{w}_\times\,\mathbf{c_0}\,c_1$
T_5-*on*	— — — 10 — — **01**	$w_0\,w_1\,w_\times\,c_1\,c_0\,\mathbf{w_0}\,w_\times\,\mathbf{c_0}\,c_1$
T_5-*open*	— — — **01** — — 10	$\mathbf{w_0}\,\mathbf{w_1}\,w_\times\,\mathbf{c_1}\,c_0\,w_0\,w_\times\,c_0\,c_1$
T_6-*on*	— — — **01** — — 10	$w_0\,\mathbf{w_1}\,w_\times\,\mathbf{c_1}\,c_0\,w_0\,w_\times\,c_0\,c_1$
T_6-*open*	— — — 10 — — **01**	$w_0\,\mathbf{w_1}\,w_\times\,c_1\,c_0\,\mathbf{w_0}\,w_\times\,\mathbf{c_0}\,c_1$
T_7-*on*	— — — **0**0 — — 10	$w_0\,\mathbf{w_1}\,w_\times\,\mathbf{c_1}\,c_0\,w_0\,w_\times\,c_0\,c_1$
T_7-*open*	— — — 10 — — **11**	$w_0\,w_1\,w_\times\,c_1\,c_0\,\mathbf{w_0}\,w_\times\,c_0\,\mathbf{c_1}$
T_8-*on*	— — — 10 — — **0**0	$w_0\,w_1\,w_\times\,c_1\,c_0\,\mathbf{w_0}\,w_\times\,\mathbf{c_0}\,c_1$
T_8-*open*	— — — **11** — — 10	$w_0\,\mathbf{w_1}\,w_\times\,c_1\,\mathbf{c_0}\,w_0\,w_\times\,c_0\,c_1$
T_9-*on*	— — — **0**0 — — **0**0	$w_0\,\mathbf{w_1}\,w_\times\,\mathbf{c_1}\,c_0\,w_0\,w_\times\,c_0\,c_1$
T_9-*open*	— — — **11** — — **11**	$w_0\,w_1\,w_\times\,\mathbf{c_1}\,\mathbf{c_0}\,w_0\,w_\times\,c_0\,c_1$
b-w-hrd	— — — 10 — — **01**	$w_0\,\mathbf{w_1}\,w_\times\,c_1\,c_0\,\mathbf{w_0}\,w_\times\,\mathbf{c_0}\,c_1$
$\bar{b}$-*w-hrd*	— — — **01** — — 10	$\mathbf{w_0}\,\mathbf{w_1}\,w_\times\,\mathbf{c_1}\,c_0\,w_0\,w_\times\,c_0\,c_1$
b-m-hrd	— — — 10 — — **11**	$w_0\,w_1\,w_\times\,c_1\,c_0\,\mathbf{w_0}\,w_\times\,c_0\,\mathbf{c_1}$
$\bar{b}$-*m-hrd*	— — — 10 — — **11**	$w_0\,w_1\,w_\times\,c_1\,c_0\,\mathbf{w_0}\,w_\times\,c_0\,\mathbf{c_1}$
c-$\bar{c}$-*hrd*	— — — 10 — — **11**	$w_0\,w_1\,w_\times\,c_1\,c_0\,\mathbf{w_0}\,w_\times\,c_0\,\mathbf{c_1}$
c-m-hrd	— — — 10 — — **0**0	$w_0\,w_1\,w_\times\,c_1\,c_0\,\mathbf{w_0}\,w_\times\,\mathbf{c_0}\,c_1$
$\bar{c}$-*m-hrd*	— — — **0**0 — — 10	$w_0\,\mathbf{w_1}\,w_\times\,\mathbf{c_1}\,c_0\,w_0\,w_\times\,c_0\,c_1$
m-w-hrd	— — — **0**0 — — **0**0	$w_0\,\mathbf{w_1}\,w_\times\,\mathbf{c_1}\,c_0\,w_0\,w_\times\,c_0\,c_1$
s-$\bar{s}$-*hrd*	— — — **0**0 — — **0**0	$w_0\,w_1\,w_\times\,\mathbf{c_1}\,\mathbf{c_0}\,w_0\,w_\times\,c_0\,c_1$
c-m-res	— — — 10 — — **0**0	$w_0\,w_1\,w_\times\,c_1\,c_0\,\mathbf{w_0}\,w_\times\,\mathbf{c_0}\,c_1$
$\bar{c}$-*m-res*	— — — **0**0 — — 10	$w_0\,\mathbf{w_1}\,w_\times\,\mathbf{c_1}\,c_0\,w_0\,w_\times\,c_0\,c_1$
m-w-res	— — — **0**0 — — **0**0	$w_0\,\mathbf{w_1}\,w_\times\,\mathbf{c_1}\,c_0\,w_0\,w_\times\,c_0\,c_1$

Appendix B

SRAM Fault Analysis and Test Evaluation

B.1 SRAM Fault Analysis

In this section operations observably affected by input stuck-at faults will be discussed. The behaviors of operations not listed here may differ slightly from their fault-free counterparts in the event-sequence model; however in the FSM model these differences cannot be observed.

***b-sa-0* fault.** This fault does not affect operations where b is normally driven to 0. This includes w_0 and r when $s = 0$. Table B.1 describes the faulty behaviors in the presence of this fault. Deviations from the fault-free behavior are indicated by bold type.

During r when $s = 1$, the grounded b forces s to ground (and hence node $\bar{s}$ to V_{dd}), causing the cell's state to change before $\bar{b}$ has a chance to fully discharge. Afterwards $\bar{s}$ drives $\bar{b}$ through the pass transistor. Since b is already 0, the sense

Table B.1: Faulty SRAM cell behavior due to the *b-sa-0* fault.

Read operation		Write operations		
Description	$(s=1)$ $w\ b\bar{b}\cdot s$	Description	w_1 $(s=0)$ $w\ b\bar{b}\cdot s$	w_1 $(s=1)$ $w\ b\bar{b}\cdot s$
initial state	$0\ \mathbf{01}\cdot 1$	initial state	$0\ \mathbf{01}\cdot 0$	$0\ \mathbf{01}\cdot 1$
float $b/\bar{b}$	$0\ \mathbf{0\tilde{1}}\cdot 1$	set $b/\bar{b}$	$0\ \mathbf{00}\cdot 0$	$0\ \mathbf{00}\cdot 1$
raise w	$1\ \mathbf{0\tilde{1}}\cdot 1$	raise w	$1\ \mathbf{00}\cdot 0$	$1\ \mathbf{00}\cdot 1$
b or $\bar{b}$ discharges	$1\ \mathbf{0\tilde{1}}\cdot\mathbf{0}$	new state	$1\ \mathbf{00}\cdot\mathbf{I}$	$1\ \mathbf{00}\cdot\mathbf{I}$
read $b/\bar{b}$ & lower w	$0\ \mathbf{0\tilde{1}}\cdot\mathbf{0}$	lower w	$0\ \mathbf{00}\cdot\mathbf{I}$	$0\ \mathbf{00}\cdot\mathbf{I}$
raise $b/\bar{b}$	$0\ \mathbf{01}\cdot\mathbf{0}$	raise $b/\bar{b}$	$0\ \mathbf{01}\cdot\mathbf{0}\vert\mathbf{1}$	$0\ \mathbf{01}\cdot\mathbf{0}\vert\mathbf{1}$

amplifiers will always detect a 0. During w_1, when w is asserted, both $b/\bar{b}$ are 0, causing both pull-up transistors to conduct which, in turn, results in a metastable state. Once w is de-asserted one of the pull-down transistors dominates over the other and the cell eventually reverts back to one of the two quiescent states. Since the state of the cell cannot be predicted after w_1, this operation must be followed immediately by r which will force the cell into a determinate but faulty state.

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

***b-sa-1* fault.** The reader can verify that only r and w_0 are affected by this fault. This is because either b is always 1 by definition, as in the case of w_1 or as in the case of w_0 in state 0, when the fact that the b line is stuck-at 1 does not alter the cell's state. Table B.2 describes the faulty behaviors of an SRAM cell affected by the *b-sa-1* fault.

An r when $s = 0$ will be misinterpreted by the sense circuitry because both $b/\bar{b}$ remain high; therefore, r is not a reliable source of output. When $s = 1$, w_0 fails, because b does not force s to ground to initiate the state transition.

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

Table B.2: Faulty SRAM cell behavior due to the *b-sa-1* fault.

Read operation		Write operation	
Description	$(s=0)$ $w\ b\bar{b}{\cdot}s$	Description	$w_0\ (s=1)$ $w\ b\bar{b}{\cdot}s$
initial state	$0\ 11{\cdot}0$	initial state	$0\ 11{\cdot}1$
float $b/\bar{b}$	$0\ \mathbf{1}\tilde{1}{\cdot}0$	set $b/\bar{b}$	$0\ \mathbf{11}{\cdot}1$
raise w	$1\ \mathbf{1}\tilde{1}{\cdot}0$	raise w	$1\ \mathbf{11}{\cdot}1$
b or $\bar{b}$ discharges	$1\ \mathbf{1}\tilde{1}{\cdot}0$	new state	$1\ \mathbf{11}{\cdot}\mathbf{1}$
read $b/\bar{b}$ & lower w	$0\ \mathbf{1}\tilde{1}{\cdot}0$	lower w	$0\ \mathbf{11}{\cdot}\mathbf{1}$
raise $b/\bar{b}$	$0\ 11{\cdot}0$	raise $b/\bar{b}$	$0\ 11{\cdot}\mathbf{1}$

***w-sa-0* fault.** Since this fault affects w, it alters the behavior of all operations. Table B.3 describes the faulty behaviors of an SRAM cell in the presence of a *w-sa-0* fault. From this table, we see that r will fail, since neither $b/\bar{b}$ line is allowed to discharge.

Table B.3: Faulty SRAM cell behavior due to the *w-sa-0* fault.

Read operations			Write operations		
Description	$(s=0)$ $w\ b\bar{b}{\cdot}s$	$(s=1)$ $w\ b\bar{b}{\cdot}s$	Description	$w_1\ (s=0)$ $w\ b\bar{b}{\cdot}s$	$w_0\ (s=1)$ $w\ b\bar{b}{\cdot}s$
initial state	$0\ 11{\cdot}0$	$0\ 11{\cdot}1$	initial state	$0\ 11{\cdot}0$	$0\ 11{\cdot}1$
float $b/\bar{b}$	$0\ \tilde{1}\tilde{1}{\cdot}0$	$0\ \tilde{1}\tilde{1}{\cdot}1$	set $b/\bar{b}$	$0\ 10{\cdot}0$	$0\ 01{\cdot}1$
raise w	$\mathbf{0}\ \tilde{1}\tilde{1}{\cdot}0$	$\mathbf{0}\ \tilde{1}\tilde{1}{\cdot}1$	raise w	$\mathbf{0}\ 10{\cdot}0$	$\mathbf{0}\ 01{\cdot}1$
b or $\bar{b}$ discharges	$\mathbf{0}\ \tilde{\mathbf{1}}\tilde{1}{\cdot}0$	$\mathbf{0}\ \tilde{1}\tilde{\mathbf{1}}{\cdot}1$	new state	$\mathbf{0}\ 10{\cdot}\mathbf{0}$	$\mathbf{0}\ 01{\cdot}\mathbf{1}$
read $b/\bar{b}$ & lower w	$0\ \tilde{\mathbf{1}}\tilde{1}{\cdot}0$	$0\ \tilde{1}\tilde{\mathbf{1}}{\cdot}1$	lower w	$0\ 10{\cdot}\mathbf{0}$	$0\ 01{\cdot}\mathbf{1}$
raise $b/\bar{b}$	$0\ 11{\cdot}0$	$0\ 11{\cdot}1$	raise $b/\bar{b}$	$0\ 11{\cdot}\mathbf{0}$	$0\ 11{\cdot}\mathbf{1}$

Lack of differential voltages on $b/\bar{b}$ yields an unpredictable response of the sense circuitry. Hence, r is not a reliable source of output. Since w is never asserted, w_1 when $s=0$ and w_0 when $s=1$ are faulty, as they do not result in a transition to the desired state. A w_0 when $s=0$ and w_1 when $s=1$ are not affected, as they effectively "do nothing".

***w-sa-1* fault.** As the reader can verify using Table 10.1, in the presence of this fault, despite minor differences in the event-sequence model, all operations are correct in the FSM model of a single cell.

B.2 SRAM Test Evaluation

Here we present the complete analyses of the MATS++, MARCH Y and MARCH C- tests. The reader is reminded that these analyses were performed under the single-fault assumption.

B.2.1 Evaluation of the MATS++ test

The MATS++ test [20] for a n-word by 1-bit SRAM, is of length $6n$, and is presented below:

$$\text{MATS++} = (w_0^i)^{\updownarrow}(r^i w_1^i)^{n\downarrow 1}(r^i w_0^i r^i)^{1\uparrow n}.$$

First, all the words are initialized to 0. Then, in the fault-free SRAM, each w_1^i in the second march element is preceded by an r^i which should produce a 0. This march element is performed in the direction from n to 1. In the third march element the first r^i should produce a 1. It is followed by a w_0^i and a second r^i producing a 0. The third march element is performed in the direction from 1 to n.

Evaluation of the MATS++ test for a single cell

Each cell in this test is subject to the following sequence of operations: $w_0 r w_1 r w_0 r$. Table B.4 provides the comparison of the fault-free response with the faulty responses of all independently testable faults. The deviations from the fault-free response, as well as the elementary tests within MATS++ are indicated in boldface. From this table it is clear that all independently testable faults will be detected reliably. The reader can

Table B.4: Evaluation of the MATS++ test for a single cell.

Fault	MATS++ $w_0 r w_1 r w_0 r$	Elementary tests for input stuck-at faults within MATS++
fault-free cell	$-0-1-0$	
b-sa-0	$-0-\mathbf{0}-0$	$w_0 r \mathbf{w_1 r} w_0 r$
b-sa-1	$-\mathbf{1}-1-\mathbf{1}$	$w_0 r \mathbf{w_1} r \mathbf{w_0 r}$
$\bar{b}$*-sa-0*	$-\mathbf{1}-1-\mathbf{1}$	$\mathbf{w_0 r} w_1 r \mathbf{w_0 r}$
$\bar{b}$*-sa-1*	$-0-\mathbf{0}-0$	$\mathbf{w_0} r \mathbf{w_1 r} w_0 r$

verify that for an inverted MATS++ test where each cell undergoes the complementary sequence of operations $w_1 r w_0 r w_1 r$, all independently testable faults will also be detected.

Evaluation of the MATS++ test for n-word by 1-bit SRAM

It is easy to verify that MATS++ detects w^i*-sa-1* faults by noting that MATS++ is an extension of MATS+. Since MATS+ has been shown to detect w^i*-sa-1* faults, and since we can insert any number of 'read' operations, as they do not corrupt the contents of the cells, we conclude that MATS++ also detects these faults.

Given that the w^i*-sa-0* faults are not reliably detectable[1], we conclude that in a n-word by 1-bit SRAM this test can reliably detect $\frac{n+4}{2n+4} \cdot 100\%$ possible input stuck-at faults, which is roughly 50% of faults (for large n) in the input stuck-at model.

Evaluation of the MATS++ test for n-word by l-bit SRAM

The MATS++ test can be extended for a n-word by l-bit SRAM in the following manner:

$$\text{MATS++} = (w^i_{0\ldots0})^{\updownarrow}(r^i w^i_{1\ldots1})^{n\downarrow1}(r^i w^i_{0\ldots0} r^i)^{1\uparrow n},$$

[1] See Section 10.2.1

where $w_{0\ldots0}$ denotes the writing of an all-0 data background, etc. Since, this extended test detects the same faults per cell, as the test for a single cell and that changing the data background does not affect fault coverage.

From the above analyses we conclude that a word-oriented extension of the MATS++ test will reliably detect all the detectable input stuck-at faults, which constitute $\frac{n+4l}{2n+4l} \cdot$ 100% of all the faults in the fault model; thus for large n the fault coverage is roughly 50%.

B.2.2 Evaluation of the MARCH Y test

The MARCH Y test [20] for a n-word by 1-bit SRAM, is of length $8n$, and is presented below:

$$\text{MARCH Y} = (w_0^i)^{\updownarrow}(r^i w_1^i r^i)^{n\downarrow 1}(r^i w_0^i r^i)^{1\uparrow n}(r^i)^{\updownarrow}.$$

First, all the words are initialized to 0. Then, in the fault-free SRAM, each w_1^i in the second march element is preceded and succeeded by an r^i where the former should produce a 0 and the latter a 1. This march element is performed in the direction from n to 1. The third march element is similar, with 0 and 1 interchanged and the march element direction reversed. In the last march element each r^i should produce a 0.

Evaluation of the MARCH Y test for a single cell

Each cell in this test is subject to the following sequence of operations: $w_0 r w_1 r r w_0 r r$. Table B.5 provides the comparison of the fault-free response with the faulty responses of all independently testable faults. As before, the deviations from the fault-free response, as well as the elementary tests within MATS++ are indicated in boldface. From this table it is clear that all independently testable faults will be detected reliably. It is worth noting that for our fault model MARCH Y is a redundant test, as the additional 'read'

Table B.5: Evaluation of the March Y test for a single cell.

Fault	March Y	Elementary tests for input stuck-at faults within March Y
	$w_0 r w_1 r r w_0 r r$	
fault-free cell	$-\ 0 - 11 - 00$	
b-sa-0	$-\ 0 - \mathbf{00} - 00$	$w_0 r \mathbf{w_1} \mathbf{r} r w_0 r r$
b-sa-1	$-\ \mathbf{1} - 11 - \mathbf{11}$	$w_0 r \mathbf{w_1} r r \mathbf{w_0} \mathbf{r} r$
$\bar{b}$-sa-0	$-\ \mathbf{1} - 11 - \mathbf{11}$	$\mathbf{w_0} \mathbf{r} r w_1 r \mathbf{w_0} \mathbf{r} r$
$\bar{b}$-sa-1	$-\ 0 - \mathbf{00} - 00$	$\mathbf{w_0} r \mathbf{w_1} \mathbf{r} r w_0 r r$

operations (in comparison with Mats++) do not increase the fault coverage and their removal would not decrease the existing coverage. The reader also can verify that for an inverted March Y test where each cell undergoes the complementary sequence of operations $w_1 r w_0 r r w_1 r r$, all independently testable faults will also be detected.

Evaluation of the March Y test for n-word by 1-bit SRAM

It is easy to verify that March Y detects w^i*-sa-1* faults by noting that March Y is an extension of Mats+. Since Mats+ has been shown to detect w^i*-sa-1* faults, and since we can insert any number of 'read' operations, as they do not corrupt the contents of the cells, we conclude that March Y also detects these faults.

Given that w^i*-sa-0* faults are not reliably detectable, we conclude that in a n-word by 1-bit SRAM this test can reliably detect $\frac{n+4}{2n+4} \cdot 100\%$ possible input stuck-at faults.

Evaluation of the March Y test for n-word by l-bit SRAM

The combination of the two extensions yield a possible March Y test for a n-word by l-bit SRAM is presented below:

$$\text{March Y} = (w^i_{0\ldots0})^{\updownarrow}(r^i w^i_{1\ldots1} r^i)^{n\downarrow1}(r^i w^i_{0\ldots0} r^i)^{1\uparrow n}(r^i)^{\updownarrow},$$

where $w_{0\ldots0}$ denotes the writing of an all-0 data background, etc. As before, this extended test detects the same faults per cell, as the test for a single cell and that changing the data background does not affect fault coverage.

The results of the above analyses indicate that a word-oriented extension of the MARCH Y test will reliably detect all the detectable input stuck-at faults, which constitute $\frac{n+4l}{2n+4l} \cdot 100\%$ of all the faults in the fault model. This coverage is identical to that of the MATS++, yet the MARCH Y is longer, as it contains redundant elements (for our fault model).

B.2.3 Evaluation of the MARCH C- test

The MARCH C- test [20] for a n-word by 1-bit SRAM, is of length $10n$, and is presented below:

$$\text{MARCH C-} = (w_0^i)^{\updownarrow}(r^i w_1^i)^{n\downarrow 1}(r^i w_0^i)^{n\downarrow 1}(r^i w_1^i)^{1\uparrow n}(r^i w_0^i)^{1\uparrow n}(r^i)^{\updownarrow}.$$

First, all the words are initialized to 0. Then, in the fault-free SRAM, each w_1^i in the second march element is preceded by an r^i where it should produce a 0. This march element is performed in the direction from n to 1. The next three march elements are similar, with 0 and 1 interchanged and/or the march element direction reversed. In the last march element each r^i should produce a 0.

Evaluation of the MARCH C- test for a single cell

Each cell in this test is subject to the following sequence of operations: $w_0 r w_1 r w_0 r w_1 r w_0 r$. Table B.6 provides the comparison of the fault-free response with the faulty responses of all independently testable faults. From this table it is clear that all independently testable faults will be detected reliably. It is worth noting that for our fault model MARCH C- is also a redundant test, as every elementary test is repeated at least twice.

Table B.6: Evaluation of the MARCH C- test for a single cell.

Fault	MARCH C- $w_0rw_1rw_0rw_1rw_0r$	Elementary tests for input stuck-at faults within MARCH C-
fault-free cell	$-0-1-0-1-0$	
b-sa-0	$-0-\mathbf{0}-0-\mathbf{0}-0$	$w_0r\mathbf{w_1r}w_0r\mathbf{w_1r}w_0r$
b-sa-1	$-\mathbf{I}-1-\mathbf{1}-1-\mathbf{1}$	$w_0r\mathbf{w_1}r\mathbf{w_0rw_1}r\mathbf{w_0r}$
$\bar{b}$*-sa-0*	$-\mathbf{1}-1-\mathbf{1}-1-\mathbf{1}$	$\mathbf{w_0r}w_1r\mathbf{w_0r}w_1r\mathbf{w_0r}$
$\bar{b}$*-sa-1*	$-0-\mathbf{0}-0-\mathbf{0}-0$	$\mathbf{w_0}r\mathbf{w_1rw_0}r\mathbf{w_1r}w_0r$

The reader also can verify that for an inverted MARCH C- test where each cell undergoes the complementary sequence of operations $w_1rw_0rw_1rw_0rw_1r$, all independently testable faults will also be detected.

Evaluation of the MARCH C- test for n-word by 1-bit SRAM

It is easy to verify that MARCH C- detects w^i*-sa-1* faults by noting that MARCH C- is an extension of MATS+. In fact, the first, second and fifth match element of MARCH C- constitute MATS+. It suffices to show that the removal of march elements three and four from MARCH C- is possible.

After the second march element the memory is filled with 1s. The third march element fills the memory with 0, and the fourth fills it back with 1s again. Since the effects of march elements three and four cancel each other, i.e. the contents of the memory cells after the fourth march element in MARCH C- is in the same as it was after the second march element, march elements three and four can be removed. By removing these two march elements, as well as the sixth one, from MARCH C-, we obtain MATS+. Since MATS+ has been shown to detect w^i*-sa-1* faults we conclude that MARCH C- also detects these faults.

Since MARCH C- test cannot reliably detect w^i*-sa-0* faults, we conclude that in a

n-word by 1-bit SRAM, this test can reliably detect $\frac{n+4}{2n+4} \cdot 100\%$ possible input stuck-at faults, which is roughly 50% of faults (for large n) in the input stuck-at model.

Evaluation of the MARCH C- **test for n-word by l-bit SRAM**

The combination of the two extensions yield a possible MARCH C- test for a n-word by l-bit SRAM is presented below:

$$\text{MARCH C-} = (w^i_{0\ldots0})^{\updownarrow}(r^i w^i_{1\ldots1})^{n\downarrow1}(r^i w^i_{0\ldots0})^{n\downarrow1}(r^i w^i_{1\ldots1})^{1\uparrow n}(r^i w^i_{0\ldots0})^{1\uparrow n}(r^i)^{\updownarrow},$$

where $w_{0\ldots0}$ denotes the writing of an all-0 data background, etc. The reader should note that, for input stuck-at faults, this extended test detects the same faults per cell, as the test for a single cell and that changing the data background does not affect fault coverage.

The results of the above analyses indicate that a word-oriented extension of the MARCH C- test will also reliably detect all the detectable input stuck-at faults, which constitute $\frac{n+4l}{2n+4l} \cdot 100\%$ of all the faults in the fault model. This coverage of input stuck-at faults is identical to that of the MATS++ and MARCH Y, yet the MARCH C- is longer than either of these two tests.

Bibliography

[1] M. Abramovici, M.A. Breuer, and A. D. Friedman. *Digital Systems Testing and Testable Design*. Computer Science Press, 1990.

[2] S. J. Adams, M. J. Irwin, and R. M. Owens. A parallel, general purpose CAM architecture. In *Proc. 4th MIT Conf. Advanced Research in VLSI*, pages 51–71. MIT Press, 1986.

[3] M. Akata, S. Karube, T. Sakamoto, T. Saito, S. Wakasugi, S. Yoshida, H. Matsuno, and H. Shibata. A scheduling content-addressable memory for ATM space-division switch control. In *International Solid State Circuits Conference*, pages 244–245. IEEE Computer Society Press, 1991.

[4] W. K. Al-Assadi, A. P. Jayasumana, and Y. K. Malaiya. On fault modelling and testing of content-addressable memories. In *Records of the IEEE Workshop on Memory Technology, Design and Testing*, pages 78–83. IEEE Computer Society Press, August 1994.

[5] W. K. Al-Assadi, Y. K. Malaiya, and A. P. Jayasumana. Modeling of intra-cell defects in CMOS SRAM. In *Records of the IEEE Workshop on Memory Technology, Design and Testing*, pages 78–81. IEEE Computer Society Press, August 1993.

[6] R. G. Bennetts and F. P. M. Beenker. Partial scan: what problem does it solve? In *Proc. European Design and Test Conference (EDAC-ETC-EuroASIC)*, pages 99–106. IEEE Computer Society Press, 1993.

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

[8] M. A. Breuer and A. D. Friedman. *Diagnosis and Reliable Design of Digital Systems.* Computer Science Press, 1976.

[9] J. A. Brzozowski and B. F. Cockburn. Detection of coupling faults in RAMs. *Journal of Electronic Testing: Theory and Applications*, 1:151–162, 1990.

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

[11] J. A. Brzozowski and H. Jürgensen. Applications of automata and languages to testing. *Publicationes Mathematicae (Debrecen)*, 48(3–4):201–215, 1996.

[12] V. Chickermane and J. H. Patel. A fault oriented partial scan design approach. *Proc. International Conf. Computer-Aided Design (ICCAD)*, pages 400–403, 1991.

[13] L. Chisvin and R. J. Duckworth. Content-addressable and associative memory: Alternatives to the ubiquitous RAM. *Computer*, 22:51–64, July 1989.

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

[15] R. David, J. A. Brzozowski, and H. Jürgensen. Testing for bounded faults in RAMs. *Journal of Electronic Testing: Theory and Applications*, 10:197–214, 1997.

[16] R. Dekker, F. Beenker, and L. Thijssen. A realistic fault model and test algorithms for static random access memories. *IEEE Transactions on Computer-Aided Design*, 9(6):567–572, June 1990.

[17] H. Fujiwara. *Logic Testing and Design for Testability.* Computer Systems Series. MIT Press, 1985.

[18] J. D. Garside, S. Temple, and R. Mehra. The AMULET2e cache system. In *Proc. International Symposium on Advanced Research in Asynchronous Circuits and Systems*, pages 208–217. IEEE Computer Society Press, March 1996.

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

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

[21] A. J. van de Goor and S. Hamdioui. Fault models and tests for two-port memories. In *IEEE VLSI Test Symposium*, pages 401–410, Monterrey, CA, April 1998. IEEE Computer Society Press.

[22] K. E. Grosspietsch. Associative processors and memories: A survey. *IEEE Micro*, pages 12–19, June 1992.

[23] R. Gupta and M. A. Breuer. Ordering storage elements in a single scan chain. *Proc. International Conf. Computer-Aided Design (ICCAD)*, pages 408–411, 1991.

[24] T. Hanyu, S. Aragaki, and T. Higuchi. Functionally separated, multiple-valued content-addressable memory and its applications. *IEE Proceedings, Part G, Circuits, Devices and Systems*, 142(3):165–172, June 1995.

[25] H. Hulgaard, S. M. Burns, and G. Borriello. Testing asynchronous ciruits: A survey. Technical Report FR-35, Dept. of Comp. Sc. and Eng., Univ, of Washington, Seattle, 1994.

[26] S. M. S. Jalaleddine and L. G. Johnson. Associative memory integrated circuit based on neural mutual inhibition. *IEE Proceedings, Part G, Circuits, Devices and Systems*, 139(4):445–449, August 1992.

[27] J-Y. Jou and K-T. Cheng. Timing-driven partial scan. *Proc. International Conf. Computer-Aided Design (ICCAD)*, pages 404–407, 1991.

[28] H. Kadota, J. Miyake, Y. Nishimichi, H. Kudoh, and K. Kagawa. An 8-kbit content-addressable and reentrant memory. *IEEE Journal of Solid-State Circuits*, sc-20(5):951–957, October 1985.

[29] Z. Kohavi. *Switching and Finite Automata Theory.* McGraw-Hill, second edition, 1978.

[30] T. Kohonen. *Content-Addressable Memories.* Springer-Verlag, New York, 1980.

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

[32] D. Lamet and J. F. Frenzel. Defect-tolerant cache memory design. In *IEEE VLSI Test Symposium*, pages 159–163. IEEE Computer Society Press, 1994.

[33] K.-J Lin and C.-W. Wu. Functional testing of content-addressable memories. In *Records of the IEEE Workshop on Memory Technology, Design and Testing*, pages 70–75, San Jose, CA, August 1998. IEEE Computer Society Press.

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

[35] P. Mazumder, J. H. Patel, and W. K. Fucks. Methodologies for testing embedded content addressable memories. *IEEE Transactions on Computer-Aided Design*, 7(1):11–20, January 1988.

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

[37] A. Meixner and J. Banik. Weak write test mode: An SRAM cell stability design for test technique. In *Proc. International Test Conference*, pages 309–318. IEEE Computer Society Press, 1996.

[38] J. L. Mundy, J. F. Burgess, R. E. Joynson, and C. Neugebauer. Low-cost associative memory. *IEEE Journal of Solid-State Circuits*, SC-7(5):364–369, October 1972.

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

[40] T. Ogura, S. Yamada, and T. Nikaido. A 4-kbit associative memory LSI. *IEEE Journal of Solid-State Circuits*, SC-20(6):1277–1282, December 1985.

[41] T. Ogura, S. Yamada, and M. Tan-no. A 20-kbit associative memory LSI for artificial intelligence machines. *IEEE Journal of Solid-State Circuits*, 24(4):1014–1020, August 1989.

[42] M. Sachdev. *Defect Oriented Testing for CMOS Analog and Digital Circuits*. Kluwer Academic Publishers, 1998.

[43] M. Sachdev. *Integrated Circuit Manufacturability, The Art of Process and Design Integration*, chapter Digital CMOS Fault Modeling and Inductive Fault Analysis. IEEE Computer Society Press, 1998.

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

[45] Y-C. Shin, R. Sridhar, V. Demjanenko, P. W. Palumbo, and S. N. Srihari. A special-purpose content-addressable memory chip for real-time image processing. *IEEE Journal of Solid-State Circuits*, 27(5):737–744, May 1992.

[46] P. R. Sidorowicz. Evaluating tests for input stuck-at faults in word-oriented static random-access memories. Technical Report CS-99-05, University of Waterloo, March 1999. ftp://cs-archive.uwaterloo.ca/cs-archive/CS-99-05/CS-99-05.ps.gz.

[47] P. R. Sidorowicz. Modeling and testing transistor faults in content-addressable memories. In *Records of the IEEE Workshop on Memory Technology, Design and Testing*, pages 83–90, San Jose, CA, August 1999. IEEE Computer Society Press.

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

[49] P. R. Sidorowicz and J. A. Brzozowski. Verification of CAM tests for input stuck-at faults. In *Records of the IEEE Workshop on Memory Technology, Design and Testing*, pages 76–82. IEEE Computer Society Press, August 1998.

[50] L. R. Tamura, T-S. Yang, D. E. Wingard, M. A. Horowitz, and B. A. Wooley. A 4-ns BiCMOS translation-lookaside buffer. *IEEE Journal of Solid-State Circuits*, 25(5):1093–1101, October 1990.

[51] E. Trischler. Incomplete scan path with an automatic test generation methodology. In *IEEE Test Conference*, pages 153–162. IEEE Computer Society Press, 1980.

[52] G. A. Uvieghara, Y. Nakagome, D. K. Jeong, and D. A. Hodges. An on-chip smart memory for a data flow CPU. In *Symposium on VLSI Circuits*, pages 121–122, May 1989.

[53] J. C. Wade and C. G. Sodini. Dynamic cross-coupled bit-line content-addressable memory cell for high-density arrays. *IEEE Journal of Solid-State Circuits*, SC-22(1):119–121, February 1987.

[54] N. H. E. Weste and K. Eshraghian. *Principles of CMOS VLSI Design. A Systems Perspective*. Addison-Wesley, second edition, 1993.

[55] P. T. Wong. A language theoretic approach to fault coverage and fault testing. Master's thesis, University of Western Ontario, March 1992.

[56] T. Yamagata, M. Mihara, T. Hamamoto, Y. Murai, T. Kobayashi, and M. Yamada. A 288-kb fully parallel content addressable memory using a stacked-capacitor cell structure. *IEEE Journal of Solid-State Circuits*, 27(12):1927–1933, December 1992.