Lecture Notes in Electrical Engineering

Volume 115

For further volumes:
http://www.springer.com/series/7818

Smita Krishnaswamy · Igor L. Markov
John P. Hayes

Design, Analysis and Test of Logic Circuits Under Uncertainty

Springer

Smita Krishnaswamy
Department of Biology
Columbia University
New York, NY
USA

John P. Hayes
Department of EECS
University of Michigan
Ann Arbor, MI
USA

Igor L. Markov
Department of EECS
University of Michigan
Ann Arbor, MI
USA

ISSN 1876-1100 ISSN 1876-1119 (electronic)
ISBN 978-90-481-9643-2 ISBN 978-90-481-9644-9 (eBook)
DOI 10.1007/978-90-481-9644-9
Springer Dordrecht Heidelberg New York London

Library of Congress Control Number: 2012943638

Printed on acid-free paper

Springer is part of Springer Science+Business Media (www.springer.com)

Foreword

Mainstream electronic systems typically assume that transistors and interconnects operate correctly over their useful lifetime. With enormous complexity and significantly increased vulnerability to failures compared to the past, future system designs cannot rely on such assumptions. Robust design is now required to ensure that such systems perform correctly despite rising complexity and increasing disturbances.

The coming generations of silicon technologies have remarkably small circuit geometries. Consequently, several causes of hardware failures, largely benign in the past, are becoming significant at the system level. Factors such as transient errors, device degradation, and variability induced by manufacturing and operating conditions are becoming much more important. Even if error rates stay constant on a per bit basis, total chip-level error rates grow with the scale of integration. Furthermore, several emerging nanotechnologies are inherently highly subject to imperfections.

Approaches exist for building electronic systems that can tolerate hardware failures, but the associated costs are too high for many applications. For example, expensive redundancy techniques have been used in space electronics and high-end mainframes. The most significant challenge is to design future electronic systems that can tolerate errors in their underlying hardware at very low (power, performance, area) costs—much lower than traditional redundancy. It will be impossible to meet this challenge unless we address the following issues:

1. Understand the characteristics of various failure mechanisms at the circuit level.
2. Analyze their effects on system behavior.
3. Create very low-cost methods to handle a wide range of failure mechanisms.
4. Optimize an overall design to achieve the required levels of reliability at the lowest cost.
5. Validate actual hardware prototypes to ensure that the designs function correctly.

This book by Dr. Smita Krishnaswamy, Prof. Igor Markov, and Prof. John P. Hayes introduces powerful analysis frameworks that will enable designers to address the above issues. Their formalisms are flexible because they are able to represent a variety of erroneous behaviors. Their systematic methods for reasoning about reliability allow designers to incorporate efficient error protection techniques into their designs. Also, their testing techniques provide insights into effective design of experiments for validating error protection techniques using hardware prototypes. Overall, the techniques presented here will enable Electronic Design Automation tool developers to create new design analysis and optimization tools that have reliability as a primary design objective together with power, performance and area.

Of course, there are numerous open questions in this area. This book introduces researchers and practitioners to some of these open questions as well. I hope it will inspire its readers to further advance the state of the art. I am excited about this book, and I hope that you will be as well!

Stanford, CA, April 2011 Subhasish Mitra

Preface

Integrated circuits (ICs) are becoming increasingly susceptible to uncertainty caused by soft errors, inherently probabilistic devices, and manufacturing variability. As device technologies scale, these effects can be detrimental to circuit reliability. In order to address this issue, we develop methods for analyzing, designing, and testing circuits subject to probabilistic effects. The main contributions of this work are: (1) a matrix-based reliability analysis framework that can capture probabilistic behavior at the logic level, (2) test generation and test compaction methods aimed at probabilistic faults in logic circuits, (3) a fast, soft-error rate (SER) analyzer that uses functional-simulation signatures to capture error effects, and (4) novel design techniques that improve reliability using little area and performance overhead.

First, we develop a formalism to represent faulty gate behavior by means of stochastic matrices called probabilistic transfer matrices (PTMs). PTM algebra provides a stochastic alternative to the deterministic role played by Boolean algebra. To improve computational efficiency, PTMs are, in turn, compressed into algebraic decision diagrams (ADDs), and ADD algorithms are developed for the corresponding matrix operations.

We propose new algorithms for circuit testing under probabilistic faults. This context requires a reformulation of existing techniques for circuit testing. For instance, a given fault may remain undetected by a given test vector, unless the test vector is repeated sufficiently many times. Also, since different vectors detect the same fault with different probabilities, the number of repetitions required is a key issue in probabilistic testing. We develop test generation methods that account for these differences, and linear programming (LP) formulations to optimize test sets for various objectives.

Next, we propose simulation-based methods to approximately compute reliability under probabilistic faults in practical settings. We observe that the probability of an error being logically masked is closely related to node testability. We use functional-simulation signatures, i.e., partial truth tables, to efficiently compute testability measures (signal probability and observability). To account for timing masking, we compute error-latching windows from timing analysis information.

Electrical masking is incorporated into our estimates through derating factors for gate error probabilities. The SER of a circuit is computed by combining the effects of all three masking mechanisms within a SER analyzer called AnSER.

Based on AnSER, we develop several low-overhead techniques that increase reliability, including: (1) a design method called SiDeR to enhance circuit reliability using partial redundancy already present within the circuit, (2) a guided local rewriting technique to resynthesize small windows of logic to improve area and reliability simultaneously, and (3) a post-placement gate-relocation technique that increases timing masking by decreasing the error-latching window of each gate, and (4) ILP algorithms to retime sequential circuits for increased reliability and testability.

Smita Krishnaswamy
Igor L. Markov
John P. Hayes

Contents

Chapter 1
Introduction

Digital computers have always been vulnerable to a variety of manufacturing and wear-out defects. Integrated circuit (IC) chips, which lie at the heart of modern computers, are subject to silicon-surface imperfections, contaminants, wire shorts, etc. Due to the prevalence of such defects, various forms of fault tolerance have been built into digital systems since the 1960s. For example, the first computers NASA sent to space were equipped with triple-modular redundancy (TMR) [1] to protect their internal logic from defects.

Over time, IC technology scaling has brought forth heightened device sensitivity to a different kind of error, known as a soft, or transient, error. Soft errors are caused by external noise or radiation that temporarily affects circuit behavior without permanently damaging the hardware. They first became problematic in the 1970s, when scientists at Intel noticed that DRAM cells experienced spontaneous bit-flips that could not be replicated. May and Woods [2] discovered that these errors were a result of α-particles emitted by trace amounts of radioactive material in ceramic chip packaging. Although the α-particle problem was eliminated for a period of time by using plastic packaging material, other sources of soft error soon became apparent. Later that year, Ziegler et al. [3] at IBM, showed that neutrons produced by cosmic rays from outer space, could also cause errors. The neutrons could strike the p-n junctions of transistors and create enough electron–hole pairs for current to flow through the junctions.

With the advent of nanoscale computing, soft errors are beginning to affect not only memory but also combinational logic. Unlike errors in memory, errors in combinational logic cannot be easily corrected and can lead to system failures, with potentially disastrous results in error-critical systems such as pacemakers, spacecraft, and servers.

New device technologies such as carbon nanotubes (CNTs), resonant tunneling diodes (RTDs), and quantum computers exhibit inherently probabilistic behavior due to various nanoscale and quantum–mechanical effects. Resilience under these sources of uncertainty is vital for technology and performance improvements. Due to the cost and high power consumption of modern ICs, the widespread addition of

S. Krishnaswamy et al., *Design, Analysis and Test of Logic Circuits Under Uncertainty*,
Lecture Notes in Electrical Engineering 115, DOI: 10.1007/978-90-481-9644-9_1,

redundancy is not a practical option for curtailing error rates. Instead, careful circuit analysis and low-cost methods of improving reliability are necessary. Further, circuits must be tested post-manufacture for their vulnerability to transient faults as well as to manufacturing defects.

In the remainder of this chapter, we review technology trends that lead to increased uncertainty in circuit behavior. We also survey previous work on soft-error rate (SER) analysis, fault-tolerant design, SER testing, and probabilistic-circuit analysis. After stating the goals of our research, we outline the remaining chapters.

1.1 Background and Motivation

Soft errors are one of the main causes of uncertainty and failure in logic circuits [4]. Current trends in circuit technology exacerbate the frequency and impact of soft errors. In this section, we describe soft errors and how they affect circuit behavior. We also survey technology trends, from current CMOS ICs to quantum devices.

1.1.1 Soft Errors

A soft error is a *transient* signal with an incorrect logic value. Soft errors can be caused by cosmic rays including α-particles, and even thermal noise. Cosmic particles can originate from supernovas or solar flares, and enter the Earth's atmosphere. They are estimated to consist of 92 % protons, 6 % α-particles, and 2 % heavy nuclei [5]. When primary cosmic particles enter the atmosphere, they can create a shower of secondary and tertiary particles, as shown in Fig. 1.1. Some of these particles can eventually reach the ground and disturb circuit behavior.

While cosmic rays are more problematic at higher altitudes, α-particles can affect circuits at any altitude. An α-particle (or equivalently, a helium nucleus) consists of two protons and two neutrons that are bound together. They are emitted by radioactive elements, such as the uranium or lead isotopes in chip-packaging materials. When packaging materials were improved in the 1980s, the problem was eliminated to a large extent; however, as device technologies scale down towards 32 nm, the particle energy required to upset the state of registers and memory circuits becomes smaller. Heidel et al. [6] show that even at 1.25 MeV, incident particles can alter the state of latches, depending on the angle of incidence. As the energy threshold for causing an error decreases, the number of particles with sufficient energy to cause errors increases rapidly [7]. For instance, even the lead in solder balls or trace amounts of radioactive contaminants in tin can affect CMOS circuits at lower energy levels [8].

When a particle actually strikes an integrated semiconductor circuit and lands in the sensitive area of a logic gate, it can cause an ionized track in silicon, known as a single-event upset (SEU), as illustrated in Fig. 1.2. An SEU is a transient fault, or soft fault as opposed to a permanent fault. The effects of an SEU do not propagate if

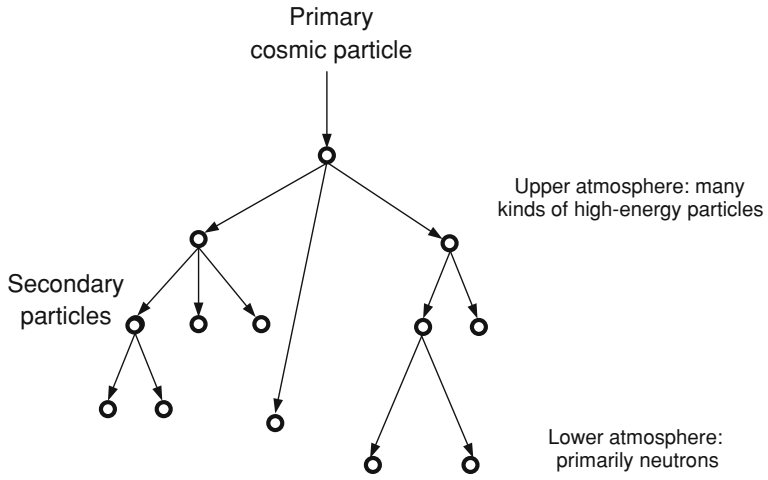


Fig. 1.1 Shower of error-inducing particles caused by a primary particle entering the atmosphere

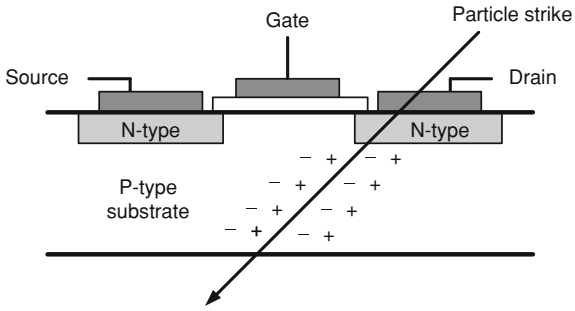


Fig. 1.2 Ionized track in a transistor caused by cosmic radiation [9]

the electric charge deposited is below the critical charge Q_{crit} required to switch the corresponding transistor on or off [4]. If an SEU deposits enough charge to cause a spurious signal pulse or glitch in the circuit, it produces a soft error. Error propagation from the site of fault occurrence to a flip-flop or primary output is stopped if there is no logically sensitized path for the error to pass through. If a soft error is propagated to and captured or latched by a flip-flop, then it can persist in a system for several clock cycles.

A single latched error can also fan out to multiple flip-flops. Unlike errors in memory, errors in combinational logic cannot be rectified using error-correcting codes (ECCs) without incurring significant area overhead. Hence, it becomes vital to find ways to accurately analyze and decrease the SER of a circuit through careful design. This is especially true of circuits in mission-critical applications, such as servers, aircraft electronics, and medical devices.

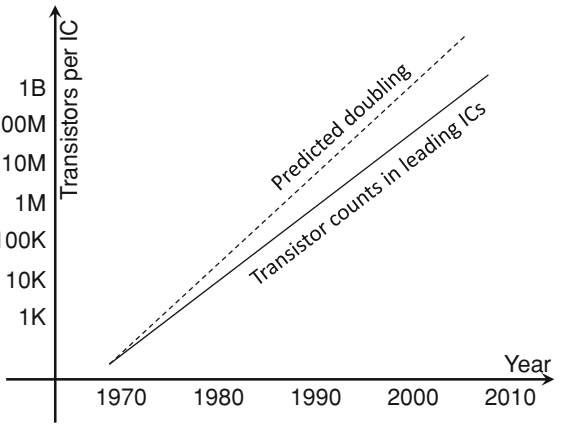


Fig. 1.3 Moore's law, showing IC density increase per year

1.1.2 Trends in CMOS

As described by Moore's law in 1965, the number of transistors in an IC tends to double every two years—a trend that has continued to the present; see Fig. 1.3. In order to facilitate this growth, chip features like gate dimensions have become smaller, along with the amount of charge stored and transferred between gates during computation. Consequently, the various sources of uncertainty described in the previous section can disrupt circuit functionality with greater ease. Other technology trends affecting the SER include decreasing power supply voltage and increasing operating frequency.

The power supply voltage has steadily decreased to improve the power-performance of ICs. Additionally, dynamic voltage scaling is now being employed to further reduce power consumption. Keyes and Landauer [10] lower bound the energy required to switch a logic gate by KT ln 2, where K is the Boltzmann constant and T is the temperature. A more accurate estimate is defined by CV^2, where V is the supply voltage and C is the capacitance of the gate given by $C = WC_{\text{out}} + \sum_{\text{fanout}} C_{\text{in}} W_j + C_L$. Here, C_{in}, C_{out}, and C_L are the input, output, and load capacitance of the gate, respectively, while W is the width of a transistor. Therefore, as W and V decrease, the switching energy approaches KT ln 2, causing logic gates to become more susceptible to noise.

Increased operating frequency—another technology trend—can lead to designs with smaller logic depth, i.e., fewer levels of logic gates. This means that fewer errors are masked by the intermediate gates between the site of fault occurrence and a flip-flop. Engineers at IROC Technologies have observed that the SER in logic circuits increases proportionally with operating frequency [11]. Processors with 40 MHz operating frequency were tested, and 400 MHz processors were simulated. The results indicate that at higher frequencies, the SER of logic is only 10 times smaller than the SER of memories—despite the additional sources of masking present in logic circuits.

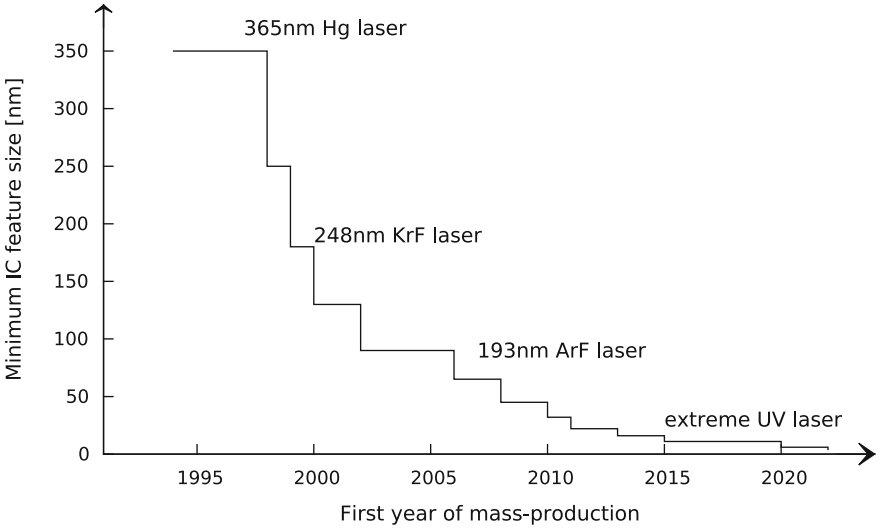


Fig. 1.4 Minimum-feature size (transistor-gate length) trends in ICs by year. Laser light sources for lithography listed correspond to the highest-resolution layers, while coarser layers may be produced by older generation lasers

Finally, technology scaling also makes devices harder to manufacture. Process variations cause stochastic behavior, in the sense that device parameters are not accurately known after manufacture. While most process parameters do not change after manufacture, those that do can often be modeled probabilistically. Figure 1.4 illustrates the lithography associated with smaller IC feature sizes by year. As the gap between the wavelength and feature sizes continues to widen, it becomes difficult for IC manufacturers to control gate and wire widths. Neighboring wires can suffer from crosstalk, the capacitive and inductive coupling that occurs when two adjacent wires run parallel to each other. Crosstalk can delay or speedup signal transitions, and sometimes causes glitches that resembles SEUs to appear [12]. Also, as the number of dopant atoms in transistors decreases, a difference of just a few atoms can lead to large variations in threshold voltage [13]. These variations cause inherent uncertainty in circuit behavior.

1.1.3 Technologies Beyond CMOS

As CMOS technologies reach the limits of their scalability, new fundamentally probabilistic computational device technologies and paradigms are arising. Examples of new device technologies under active investigation include CNTs, RTDs, quantum-dot cellular automata (QCA), and various quantum computing technologies, like ion traps that handle quantum bits (qubits). Quantum computing devices are inherently

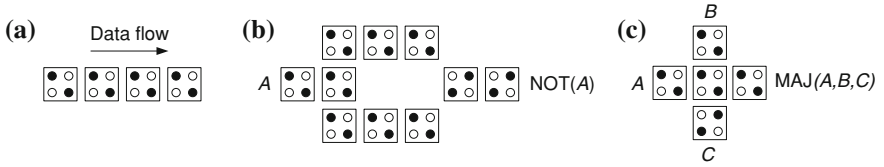


Fig. 1.5 Circuit elements implemented as quantum-dot cellular automata: **a** wire segment; **b** NOT gate; and **c** Majority gate [15]

probabilistic (even during fault-free operation) because qubits exist in superposition states and collapse to either 0 or 1, with different probabilities upon measurement.

Quantum-dot cellular automata were first proposed by Lent et al. in 1993 [14]. They defined the now well-known model consisting of four quantum dots that occupy the corners of a square cell with potential barriers between pairs of dots. Within this cell, two extra electrons are available, and by raising and lowering potential barriers, the electrons can localize on a dot. Due to Coulombic interactions, the electrons will occupy "antipodal" or diagonally opposite sites in the square cells. By positioning cells in different orientations and configurations, universal sets of logic gates can be constructed. Figure 1.5 shows several circuit elements constructed from quantum-dot cellular automata.

QCA are thought to have two main sources of error: (1) decay—when electrons that store information are lost to the environment, and (2) switching error—when the electrons do not properly switch from one state to another due to background noise or voltage fluctuations [16, 17]. A key feature of QCAs is that logical components and wires are both created from the same basic cells. Since quantum-dots are highly susceptible to thermal noise and other sources of errors, computations performed by QCA devices may need to be repeated multiple times in order to achieve a desired level of reliability. Further, errors can combine in complex ways that can be difficult to analyze. For instance, longer wires or gates utilizing redundant cells may have higher error propensity. For instance, if each cell in a 3-cell wire segment has probability of error p, then the probability of an erroneous output is $3p + p^3$, but the error of a 4-cell wire would be $4p + \binom{4}{3}p^2$ (these formulas account for the cancelation of paired errors).

Generally, a gate with n cells can experience $O(2^n)$ combinations of errors. Therefore, analyzing QCA devices requires an automated and efficient method to compute overall error probabilities and error trends under different configurations. Our work directly addresses this issue by providing a framework for representing error-prone gates, along with algorithms for computing overall error probabilities for devices composed of such gates. It has been extensively used to derive reliability information about QCA devices [15].

1.2 Related Work

This section discusses related work on probabilistic circuit analysis, SER analysis, fault-tolerance techniques, and soft-error testing.

1.2.1 Probabilistic Analysis of Circuits

We have developed a novel probabilistic matrix-based model for logic elements, which is used with matrix operations and symbolic methods to evaluate error probabilities in general logic circuits [18, 19]. This is covered in detail in Chaps. 2 and 3. Rejimon et al. [17] proposed capturing errors in nano-domain logic circuits by Bayesian networks. A Bayesian network is a directed graph with nodes representing variables and edges representing dependence relations among the variables. If there is an edge from node a to another node b, then we say that a is a parent of b. If there are n variables, $x_1 \ldots x_n$, then the joint-probability distribution for x_1 through x_n is represented as the product of the conditional probability distributions

$$\prod_{i=1}^{n} P[x_i|\text{parents}(x_i)]$$

If x_i has no parents, its probability distribution is said to be unconditional. In order to carry out numerical calculations on a Bayesian network, each node x_i is labeled with a probability distribution, conditioned on its parents. Certain nodes such as those corresponding to primary inputs are given predefined probabilities. The probabilities of other nodes are then computed using a technique called belief propagation. Joint probabilities are computed in large Bayesian networks using sampling methods such as importance sampling. Many software tools [20, 21] exist for Bayesian network analysis.

Bahar et al. [22] proposed to model and design CNT-based neural networks using Markov random fields (MRFs). MRFs are similar to Bayesian networks in that they specify joint probability distributions in terms of local conditional probabilities, but they can also describe cyclic dependences. In [22], the neural network is described by an MRF with node values computed by a weighted sum of conditional probabilities of a neighboring clique of nodes. This formulation of an MRF is known as the Gibbs formulation and lends itself to optimizing for clique energy, which is translated into low probabilities of node error in [22]. Related to this, Nepal et al. [23] present a method for implementing MRF-based circuits in CMOS, and Bhadhuri et al. [24] describe a software tool Nanolab, which uses the algorithm from [22] to automate the design of fault-tolerant architectures like CTMR in nanotechnologies.

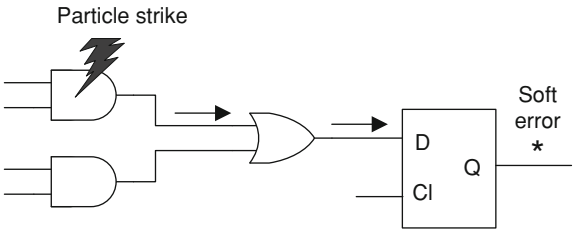


Fig. 1.6 Illustration of transient-fault propagation in combinational logic

1.2.2 Soft-Error Rate Analysis

This section focuses on techniques developed for the problem of SER estimation. We first introduce the problem and then discuss solutions that appear in the literature, often alongside our work. Here, we aim to reveal the intuition behind SER analysis methods and to motivate techniques introduced in later chapters.

There are several factors to consider in determining the SER of a logic circuit. Figure 1.6 illustrates the three main conditions that are required for an SEU to be latched, and these conditions are explained below.

- The SEU must exercise sufficient energy to change a logic value and propagate it through subsequent gates. If not, the fault is electrically masked.
- The change in a signal's value must propagate through the logic circuit and affect a primary output. If not, the fault is logically masked.
- The fault must reach a flip-flop during the sensitive portion of a clock cycle, known as the latching window. If not, the fault is temporally masked.

The probability of electrically masking a fault depends on the electrical characteristics of the gates it encounters on its way to a primary output, i.e., it is path-dependent. Similarly, the propagation delay of the SEU before reaching a latch or a primary output depends on the gate and interconnect delays along the path it takes. Any path the SEU takes has to have non-controlling values on side inputs. Therefore, different input vectors can sensitize different sets of paths.

Assuming a single particle strike per clock cycle, the SER can be computed using the brute-force algorithm given in Fig. 1.7. In this algorithm, P_{err} is the probability of an error on a gate. It is computed using the following variables.

- $P(i)$, the probability of vector i being applied to the input,
- $P_{\text{strike}}(n)$, the probability of a fault at location n,
- $P_{\text{attenuate}}(\text{path}(p))$, the probability of attenuation along path p, and
- $P_{\text{latch}}(p, o)$, the probability of an error signal or *glitch* arriving on path p at output o during a latching phase of a clock cycle.

Neglecting to model any of these probabilities can lead to overestimation of the SER. Figure 1.8 shows an example of an SEU in the ISCAS-85 circuit C17, along with paths logically sensitized by a specific input vector.

Fig. 1.7 Basic SER computation algorithm

```
compute_SER(circuit C)
{
 for(input vector i)
  for(node n ∈ C)
    for(output o ∈ C)
      for(sensitized path p ∈ path(i,n))
        P_prop(n) = (1 − P_attenuate(p))
        Perr(C) += P(i)P_strike(n)P_prop(n)P_latch(p,o)
 return Perr(C)
}
```

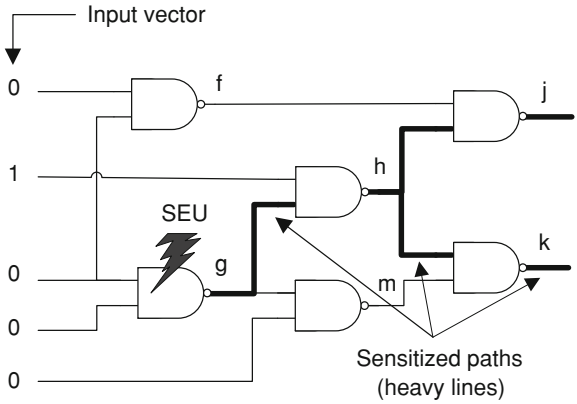


Fig. 1.8 Illustration of logically sensitized paths *(heavy lines)* for error propagation in response to a specific input vector

The algorithm of Fig. 1.7 is only practical for the smallest of circuits. The number of possible input vectors is exponential in the number of circuit inputs, and the number of sensitized paths can grow exponentially with the circuit size [25]. Therefore, even determining the probability of logical masking is as difficult as counting the number of solutions to a SAT instance—a problem in the ♯P-hard complexity class.

Several software tools have been recently constructed to approximate the SER for combinational circuits. Below, we describe three of these tools and their SER computation techniques [26–28]. Of the three algorithms, SERA is closest to that of Fig. 1.7. The SERA algorithm, which is outlined in Fig. 1.9a, relies on user-specified input patterns and analyzes each input vector individually. For each gate, SERA finds all the paths from the gate to an output. Then, SEU-induced glitches are simulated on inverter chains of the same lengths as the paths in order to determine the probability of electrical masking. In general, there can be many paths of the same length, but only one representative inverter chain of each length is simulated. Since the number of paths can be exponential in the size of the circuit, this algorithm runs in exponential time in the worst case. However, the average runtime is much smaller since SERA only simulates paths of unique length.

Unlike SERA, the FASER tool [27], whose algorithm is shown in Fig. 1.9b, uses binary decision diagrams (BDDs) to enumerate all possible input vectors. A BDD is

(a)

```
compute_SER_SERA(circuit C, vectors V)
{
 for (v ∈ V)
  for (nodes n ∈ C)
   for (output o ∈ C)
    for (sensitized path p ∈ path(n,v))
     l = length(p)
     Perr(n) = simulate_inverter_chain(l)
     K = Area(n)/Area(C)
     Perr(C)+ = Perr(n) * K
 return Perr(C)
}
```

(b)

```
compute_SER_FASER(circuit C)
{
 for (n ∈ C)
  create_strike_BDD(n)
  for (output o ∈ C)
   for (gate g ∈ C
    create_static_BDD(g)
    if (g ∈ fanout(n))
     modify_terminals(g)
    sort_topological(C)
    for (g ∈ C)
     attenuate(inputs(g))
     merge_BDD(inputs(g))
    Perr(C)+ = (Area(n)/Total) * Flux * Perr(BDD(o))
 return Perr(C)
}
```

(c)

```
compute_SER_SET(circuit C)
{
 sort_topological(C)
 for (gate g ∈ G)
  SERD(g) = calculate_strike_SERD(g)
  SERD(g) = merge_input_SERD(SERD(g),inputs(g))
 for (output o ∈ C)
  Perr(C)+ = Perr(SERD(o))
 return Perr(C)
}
```

Fig. 1.9 Algorithms for SER computation used by **a** SERA, **b** FASER, and **c** SET

created for each gate in a circuit—a static BDD for gates outside the fanout cone of the glitch location, and duration and amplitude BDDs for gates in the fanout cone of the glitch location. Then, these BDDs are merged in topological order. During the process of merging, the width and amplitude of glitches at inputs are decreased

Table 1.1 Summary of differences between three SER evaluation tools

Attribute	SERA	FASER	SET
Logic masking	Vector simulation	BDD-based analysis	Vector simulation
Timing masking	SER derating	No details given	SER derating
Electrical masking	Inverter-chain simulation	Gate characterization	Gate characterization
Fault assumptions	Single	Single	Multiple

according to FASER's estimation of electrical masking. Due to its complete input-vector enumeration, FASER's BDD representations can require a lot of memory for practical circuits, especially multipliers. FASER attempts to lessen the amount of memory used by partitioning the circuit into smaller subcircuits and then treating the inputs to these subcircuits as pseudo-primary inputs.

SET's algorithm [28], shown in Fig. 1.9c, proceeds in topological order and considers each gate only once. For each gate, SET encodes the probability and shape of a glitch as a Weibull probability-density function. This distribution over the Weibull parameters is known as an SER descriptor (SERD). The SERD for a gate is combined with those of its inputs, to produce the output SERD. The Weibull parameters are slightly changed at each gate to account for electrical attenuation, and the new output SERDs are passed onto their successor gates. The SET algorithm is similar to static timing analysis (STA) and does not consider false paths. The authors of SET do provide another vector-driven mode that computes SER vector-by-vector to account for input-pattern dependence.

Table 1.1 summarizes the main characteristics of the tools described above, as well as their methods for incorporating masking mechanisms. These tools have vastly different methods of computing SER, and their different assumptions can yield very different SER values for the same circuit. Therefore, depending on the degree of validity of the assumptions made in particular situations, they can vary greatly in the accuracy of their SER estimates.

Our work aims to build SER analysis tools that are scalable, accurate, and can be used early, i.e., during logic design [29–31]. In addition, they allow for various assumptions when evaluating SER. For instance, they can be used to analyze SER under multiple or single fault assumptions based on the context in which the SER estimate is being made. Due to our emphasis on reliability-driven logic design, we focus on modeling logical masking accurately and efficiently. We then use our tools to guide several design techniques to improve circuit resilience against soft errors.

1.2.3 Fault-Tolerant Design

Techniques for transient fault-tolerance have been developed for use at nearly all stages of the design flow. Generally, these techniques rely on enhancing masking mechanisms to mitigate error propagation. Below, we discuss several of the techniques and highlight their masking mechanisms.

Faults can be detected at the architectural level via some form of redundancy and can be corrected by rolling back to a checkpoint to replay instructions from that checkpoint. Redundant multi-threading (RMT) [32, 33] is a common method of error detection at the architectural level. RMT refers to running multiple threads of the same program and comparing the results. The first or leading thread often executes ahead of other threads to allow time for transient glitches to dissipate.

The DIVA method [34, 35] advocates the use of a functional checker to augment detect-and-replay by recomputing results before they are committed. Since the data fetch is assumed to be error-free (and memory is assumed to be protected by ECC), the functional checkers simply rerun computations on pre-fetched data. Other methods attempt to detect errors using symptoms that are unusual behaviors for specific programs. An example is an instruction that accesses data spatially far from previous executions of the same instruction. Another example is a branch predictor that misspeculates with unusually high frequency [36, 37]. The main masking mechanism in all these techniques is functional masking. Components are selected for the addition of fault tolerance using a metric called the architectural vulnerability factor (AVF) of the component in question, which is usually computed by statistical fault injection or other forms of performance analysis [38].

At the logic level, designers have complete information about the function of a circuit and its decomposition into gates or other low-level functional modules. At this level, one can assess logic masking in more detail. Traditionally, logic masking has been increased by adding gate-level redundancy to the circuit. John von Neumann [39], in his classic paper on reliability, showed that it is possible to build reliable circuits with unreliable components, using schemes like cascaded triple modular redundancy (CTMR) and NAND multiplexing which are illustrated in Figs. 1.10 and 1.11 respectively. CTMR contains TMR units that are, in turn, replicated thrice, and this process is repeated until the required reliability is reached.

In NAND multiplexing, which is depicted in Fig. 1.11, each unreliable signal is replicated N times. Then, a set of NAND gates, each of which takes two of the N redundant signals as inputs, implements a simple majority function. Some of these NANDs may produce incorrect outputs due to an unfortunate combination of inputs; however, such instances are rare since a random permutation changes the gate-pairs between stages of multiplexers. von Neumann concluded that as long as component error probabilities are below a certain threshold, redundancy can increase the reliability of a system to any required degree.

Techniques that involve replicating an entire circuit increase chip area significantly, and therefore decrease chip yield. Mohanram and Touba [40] proposed to partially triplicate logic by selecting regions of the circuit that are especially susceptible to soft errors. Such regions are determined by simulating faults with random test vectors. Dominant-value reduction [40] is also used to duplicate, rather than triplicate, selected logic. This technique mitigates the soft errors that cause only one of the erroneous transitions 0–1 or 1–0, depending on which is more common. More recently, Almukhaizim et al. [41] used a design modification technique called rewiring to increase reliability. In the spirit of [41], our work focuses on lightweight modifica-

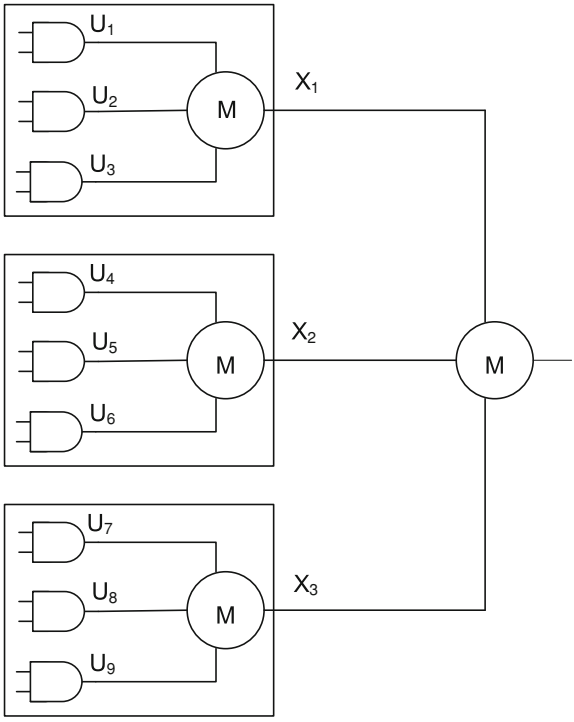


Fig. 1.10 The cascaded TMR scheme; *M* denotes a Majority gate [39]

tions to the circuit that increase reliability without requiring significant amounts of redundancy. These types of modifications will be discussed further in Chap. 6.

At the transistor level, gates can be characterized in electrical terms and electrical masking can be used as an error-mitigation mechanism. Gate sizing is a common way to increase electrical masking. Increasing the area of a gate increases its internal capacitance and therefore the critical charge Q_{crit} necessary for a particle strike to alter a signal. However, this technique adds to overall circuit area and can also increase critical path delay. Therefore, gates are usually selected for hardening according to their susceptibility to error, which requires error-susceptibility analysis at the logic level.

Another transistor-level technique for soft-error mitigation is the dual-port design style proposed by Baze et al. [42], and later by Zhang et al. [43]. Dual-port gates decrease charge-collection efficiency by using two extra transistors placed in a separate well from the original transistors.

In the 1990s, Nicolaidis proposed multiple-sampling as a method of increasing electrical masking [44]. Three latches sample a signal with small delays between them, and a voter decides the correct value of the signal. Since stray glitches tend to have short duration, the erroneous value induced by a glitch is likely to be sampled by only one of the three latches. RAZOR [45] uses this idea for dynamic voltage scaling, sampling signals twice, and when an error is found, restoring program correctness

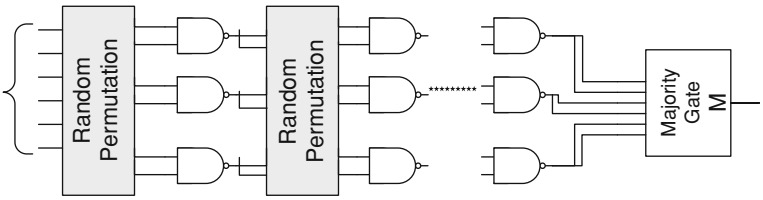


Fig. 1.11 The NAND-multiplexing scheme [39]

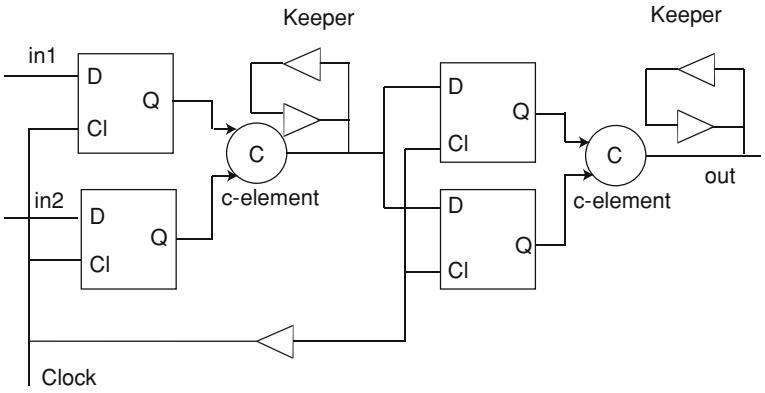


Fig. 1.12 The error-correcting BISER flip-flop design with a *C*-element and a keeper circuit [46]

via a detect-and-playback scheme. The BISER [46] architecture duplicates flip-flops and feeds the outputs to a *C*-element and a keeper circuit. At each clock cycle, if the new flip-flop values are the same, the *C*-element forwards the new value to the primary outputs; otherwise, the *C*-element retains the value from the previous cycle. Figure 1.12 shows BISER's flip-flop design.

After the placement and routing of a circuit are completed, gate and interconnect delays can be determined. In earlier IC technologies, timing was usually analyzed at the gate level since wire delay contributed only a small (often negligible) fraction of the critical path delay. However, wire delay currently dominates gate delay and must be incorporated into any accurate timing analyzer. Once we can analyze the timing, we can also obtain information about timing masking [4, 47]. To date, few techniques that decrease timing masking have been proposed.

In summary, faults can be mitigated at several levels of abstraction including the architecture, logic, transistor (electrical), and physical levels. Solutions at the logic and transistor levels tend to be more general and do not depend on the function of the circuit. Our work shows that such fine-grained, accurate SER analysis is computationally feasible and decreases overhead [29–31]. Table 1.2 summarizes the fault-tolerance techniques and masking mechanisms discussed in this section.

Table 1.2 Error-masking and fault-tolerance at several levels of abstraction

Level	Masking mechanism	Fault-tolerance techniques
Architecture/RTL	Functional masking	Multithreading, functional checkers, replay
Logic	Logic masking	TMR, NAND-mux, partial replication, rewiring
Transistor	Electrical masking	Gate hardening, dual-port gates, dual sampling
Physical	Timing masking	No known techniques

1.2.4 Soft-Error Testing

Chip manufacturers including IBM, Intel, and Toshiba, as well as medical equipment manufacturers like Medtronics, routinely test their chips for SER [2, 5, 48, 49]. SER testing is normally done in one of two ways: field testing or accelerated testing. In field testing, a large number of devices are connected to testers and evaluated for several months under normal operating conditions. In accelerated testing [3], devices are irradiated with neutron or α-particle beams, thus shortening the test time to a few hours. Accelerated tests can be further sped up by reducing the power-supply voltage, which changes the Q_{crit} of transistors.

It is difficult, however, to translate the SER figures obtained by accelerated testing into that of field testing [48]. For instance, the SER may vary over time due to solar activity, which can be difficult to replicate in a laboratory setting. Also, intense radiation beams can cause multiple simultaneous errors, triggering system failures more often than normal. Therefore, it is necessary to field-test some devices to calibrate the accelerated tests.

Since field testing requires a large number of devices, and dedicated testers for each device, Hayes et al. [50] have proposed a non-concurrent built-in self-test (BIST) approach to online testing. They define the impact of various soft faults on a circuit in terms of frequency, observability and severity. For instance, more frequent and observable faults are considered more impactful than rare faults. With this fault characterization, integer linear programming (ILP) is used to generate tests that satisfy various objectives, such as ensuring a minimum fault-detection probability.

Other researchers have sought to accelerate testing by selecting test patterns that sensitize faults. Conceptually, the main difference between testing for hard rather than soft errors is that soft errors are only present for a fraction of the testing time. Therefore, test vectors must be repeated to detect soft faults and they must be selected to sensitize the most frequent faults. Sanyal et al. [51] accelerate testing by selecting a set of error-critical nodes and deriving test sets that, using ILP, sensitize the maximum number of these faults. In work that preceded [51], we developed a way of identifying error-sensitive test vectors for multiple faults, which can include other masking mechanisms like electrical masking, and we devised algorithms for generating test sets to accelerate SER testing [52, 53].

1.3 Organization

This book focuses on probabilistic circuit analysis, testing for probabilistic faults, and fault-tolerant logic design. We first develop methods to model inherently probabilistic methods in logic circuits and to test circuits for determining their reliability after manufacture. We carefully study various error-masking mechanisms inherent in logic circuits in order to design circuits with better reliability. Our main goals are:

- To develop general and accurate methods for modeling and reasoning about probabilistic behavior in logic circuits.
- To develop test methods that accurately and efficiently measure soft-error susceptibility in manufactured circuits.
- To develop scalable and accurate methods of SER and susceptibility analysis, usable during the CAD flow at the gate level.
- To devise methods that improve the resilience of logic desigs against soft errors.

The remainder of this book is organized as follows. Chapter 2 presents a general matrix-based reliability analysis technique, the probabilistic transfer matrix (PTM) framework, to model faulty gate and circuit behavior. PTMs form an algebra for reasoning about uncertain behavior in logic circuits. This algebra defines several specific types of matrices to describe gates and wires, along with matrix operations that can be used symbolically or numerically to combine the matrices. Several new matrix operations that are useful in modifying and combining PTMs are then used to derive information about circuit reliability and output error probabilities under various types of faults.

Chapter 3 develops decision diagram-based methods for compressing and computing with PTMs. Several heuristics are presented for improving the scalability of PTM-based computations, including dynamic evaluation ordering, partitioning, hierarchical computation, and sampling.

Chapter 4 introduces a new method to test for probabilistic faults. We discuss the differences among traditional testing methods geared towards identifying structural defects and assessing circuit susceptibility to probabilistic faults. We also present algorithms for compacting a test-vector set.

Chapter 5 presents an efficient way to analyze SER at the logic level. Here, we derive probabilistic fault models from the standard stuck-at model used in the testing literature. We propose ways to account for the three basic masking mechanisms via probabilistic reasoning and functional simulation. We also present techniques in the spirit of static-timing analysis to estimate timing masking, and use derating factors to account for electrical masking. These analysis methods are also extended to sequential circuits.

Chapter 6 applies the analysis techniques from the previous chapter to the design of reliable circuits. The methods employed include logic rewriting, gate hardening, and a novel technique we call SiDeR. This technique exploits functional relationships among signals to partially replicate areas of logic with low redundancy. We also present a gate relocation technique that targets timing masking, a factor which

has often been overlooked in fault-tolerant design. This technique entails no area overhead and negligible performance overhead. For sequential circuits, we derive integer linear programs for retiming, which move latches to positions where they are less likely to propagate errors to primary outputs. Chapter 7 summarizes the book and discusses some possible directions for future research.

References

1. Siewiorek DP, Swarz RS (1992) Reliable computer systems, 2nd edn. Digital Press.
2. May TC, Woods MH (1979) Alpha-particle-induced soft errors in dynamic memories. IEEE Trans Electron Devices 26:2–9
3. Ziegler JF et al (1996) Terrestrial cosmic rays. IBM J of Res Dev 40(1):19–39
4. Shivakumar P et al (2002) Modeling the effect of technology trends on soft error rate of combinational logic. In: Proceedings of ICDSN, pp 389–398
5. Ziegler JF et al (1996) IBM experiments in soft fails in computer electronics (1978–1994). IBM J of Res Dev 40(1):3–18
6. Heidel DF et al (2006) Single-event-upset critical charge measurements and modelling of 65 nm Silicon-on-Insulator Latches and Memory Cells. IEEE Trans Nucl Sci 53(6):3512–3517
7. Rodbell K et al (2007) Low-energy proton-induced single-event-upsets in 65 nm node, silicon-on-insulator, latches and memory cells. IEEE Trans Nucl Sci 54(6):2474–2479
8. Heidel DF et al (2008) Alpha-particle induced upsets in advanced CMOS circuits and technology. IBM J of Res Dev 52(3):225–232
9. Grove A (2002) Changing Vectors of Moore's Law. International electronic devices meeting, http://www.intel.com/research/silicon/mooreslaw.htm
10. Keyes RW, Landauer R (1970) Minimal energy dissipation in computation. IBM J of Res Dev 14(2):152-157
11. IROC Technologies (2002) White paper on VDSM IC logic and memory signal integrity and soft errors. http://www.iroctech.com
12. Rabaey JM, Chankrakasan A, Nikolic B (2003) Digital integrated circuits. Prentice Hall
13. Borkar S et al (2003) Parameter variations and impact on circuits and microarchitecture. In: Proceedings of DAC, pp 328–342
14. Lent CS, Tougaw PD, Porod W (1993) Bistable saturation in coupled quantum dots for quantum cellular automata. Appl Phys Lett 62(7):714–716
15. Dysart TJ, Kogge PM (2007) Probabilistic analysis of a molecular quantum-dot cellular automata adder. In: Proceedings of DFT, pp 478–486
16. Kummamuru RK et al (1993) Operation of Quantum-Dot Cellular Automata (QCA), shift registers and analysis of errors. IEEE Trans Electron Devices 50–59:1906–1913
17. Rejimon T, Bhanja S (2006) Probabilistic error model for unreliable Nano-logic Gates. In: Procedings of NANO, pp 47–50
18. Krishnaswamy S, Viamontes GF, Markov IL, Hayes JP (2005) Accurate reliability evaluation and enhancement via probabilistic transfer matrices. In: Proceedings of DATE, pp 282–287
19. Krishnaswamy S, Viamontes GF, Markov IL, Hayes JP (2008) Probabilistic transfer matrices in symbolic reliability analysis of logic circuits. ACM Trans Des Autom of Electron Syst, 13(1), article 8
20. Genie SMILE software, http://genie.sis.pitt.edu/
21. Openbayes software, http://www.openbayes.org/
22. Bahar RI, Mundy J, Chan J (2003) A Probabilistic based design methodology for nanoscale computation. In: Proceedings of ICCAD, pp 480–486
23. Nepal K et al (2005) Designing logic circuits for probabilistic computation in the presence of noise. In: Proceedings of DAC, pp 485–490

24. Bhaduri D, Shukla S (2005) NANOLAB-A tool for evaluating reliability of defect-tolerant nanoarchitectures. IEEE Trans Nanotechnol 4(4):381–394
25. Ramalingam A et al (2006) An accurate sparse matrix based framework for statistical static timing analysis. In: Proceedings of ICCAD, pp 231–236
26. Zhang M, Shanbhag NR (2004) A soft error rate analysis (SERA) methodology. In: Proceedings of ICCAD, pp 111–118
27. Zhang B, Wang WS, Orshansky M (2006) FASER: fast analysis of soft error susceptibility for cell-based designs. In: Proceedings of ISQED, pp 755–760
28. Rao R, Chopra K, Blaauw D, Sylvester D (2006) An efficient static algorithm for computing the soft error rates of combinational circuits. In: Proceedings of DATE, pp 164–169
29. Krishnaswamy S, Plaza SM, Markov IL, Hayes JP (2007) Enhancing design robustness with reliability-aware resynthesis and logic simulation. In: Proceedings of ICCAD, pp 149–154
30. Krishnaswamy S, Plaza SM, Markov IL, Hayes JP (2009) Signature-based SER analysis and design of logic circuits. IEEE Trans CAD, 28(1):59–73
31. Krishnaswamy S, Markov IL, Hayes JP (2008) On the role of timing masking in reliable logic circuit design. In: Proceedings of DAC, pp 924–929
32. Mukherjee SS, Kontz M, Reinhardt SK (2002) Detailed design and evaluation of redundant multithreading alternatives. In: Proceedings of ISCA, pp 99–110
33. Rotenberg E (1999) AR-SMT: A microarchitectural approach to fault tolerance in microprocessor. In: Proceedings of Fault-Tolerant, Computing Systems, pp 84–91
34. Austin T (1999) DIVA: A reliable substrate for deep submicron microarchitecture design. In: Proceedings of MICRO, pp 196–207
35. Weaver C, Austin T (2001) A fault tolerant approach to microprocessor design. In: Proceedings of DSN, pp 411–420
36. Racunas P, Constantinides K, Manne S, Mukherjee SS (2007) Perturbation-based fault screening. In: Proceedings of HPCA, pp 169–180
37. Armstrong DN, Kim H, Mutlu O, Patt YN (2004) Wrong path events: Exploiting unusual and illegal program behavior for early misprediction detection and recovery. In: Proceedings of MICRO, pp 119–128
38. Mukherjee SS et al (2003) A systematic methodology to compute the architectural vulnerability factors for a high-performance microprocessor. In: Proceedings of MicroArch, pp 29–40
39. von Neumann J (1956) Probabilistic logics & synthesis of reliable organisms from unreliable components. Autom stud, 34:43–98
40. Mohanram K, Touba NA (2003) Partial error masking to reduce soft error failure rate in logic circuits. In: Proceedings of DFT, pp 433–440
41. Almukhaizim S et al (2006) Seamless integration of SER in rewiring-based design space exploration. In: Proceedings of ITC, pp 1–9
42. Baze MP, Buchner SP, McMorrow D (2000) A digital CMOS design technique for SEU hardening. IEEE Trans Nucl Sci 47(6):2603–2608
43. Zhang M, Shanbhag N (2005) A CMOS design style for logic circuit hardening. In: Proceedings of International Reliability Physics, Symposium, pp 223–229
44. Nicolaidis M (1999) Time redundancy based soft-error tolerance to rescue nanometer technologies. In: Proceedings of VTS, pp 86–94
45. Ernst D et al (2003) Razor: circuit-level correction of timing errors for low power operation. IEEE Micro 24(6):10–20
46. Zhang M et al (2006) Sequential element design with built-in soft error resilience. IEEE Trans VLSI 14(12):1368–1378
47. Miskov-Zivanov N, Marculescu D, (2006) MARS-C: Modeling and reduction of soft errors in combinational circuits. In: Proceedings of DAC, pp 767–772
48. Kobayashi H et al (2004) Comparison between neutron-induced system-SER and accelerated-SER in SRAMs. In: Proceedings of International Reliability Physics, Symposium, pp 288–293
49. Wilkinson J, Hareland S (2005) A cautionary tale of soft errors induced by SRAM packaging materials. IEEE Trans Device Mater Reliab 5(3): 448–433

50. Hayes JP, Polian I, Becker B (2007) An analysis framework for transient-error tolerance. In: Proceedings of VTS, pp 249–255
51. Sanyal A, Ganeshpure K, Kundu S (2008) On accelerating soft-error detection by targeted pattern generation. In: Proceedings of ISQED, pp 723–728
52. Krishnaswamy S, Markov IL, Hayes JP (2007) Tracking uncertainty with probabilistic logic circuit testing. IEEE Des Test 24(4):312–321
53. Krishnaswamy S, Markov IL, Hayes JP (2005) Testing logic circuits for transient faults. In: Proceedings of ETS, pp 102–107

Chapter 2
Probabilistic Transfer Matrices

In this chapter, we present a general framework for reliability analysis that treats circuits entirely probabilistically. While this is useful for analyzing soft errors, it is also useful for analyzing devices that periodically fail or behave probabilistically during regular operation. Quantum dot cellular automata (QCA), where gates and wires are made from "quantum dots", are examples of such devices; see Fig. 1.5. Each dot consists of a pair of electrons that can be configured in two different ways to represent a single bit of information. In QCA, both gates and wires are created from planar arrangements of dots. QCA have an inherent propensity for faults because the electrons can easily be absorbed into the atmosphere or arrange themselves in an ambiguous configuration [1, 2]. Other examples of inherently probabilistic devices include probabilistic CMOS, molecular logic circuits, and quantum computers.

Historically, the probabilistic analysis of circuits has centered around signal-probability estimation, which was motivated by random-pattern testability concerns [3–5]. In short, the probability of a signal being a 0 or 1 gives some indication of the difficulty in controlling (and therefore testing) the signal. In this chapter, we treat circuits probabilistically to analyze circuit reliability. As opposed to signal probability estimation, reliability analysis deals with complex probabilistic failure modes and error propagation conditions.

In general, accurate reliability analysis involves computing not just a single output distribution but, rather, the output error probability for each input pattern. In cases where each gate experiences input-pattern dependent errors—even if the input distribution is fixed—simply computing the output distribution does not give the overall circuit error probability. For instance, if an XOR gate experiences an output bit-flip error, then the output distribution is unaffected, but the wrong output is paired with each input. Therefore, we need to separately compute the error associated with each input vector.

Consider the circuit in Fig. 2.1. Given that each gate experiences an error with probability $p = 0.1$, the circuit's output error probability for the input combination 000 is 0.244. The input combination 111 leads to an output error probability of 0.205. The overall error rate of the circuit is the sum of the error probabilities, weighted by

S. Krishnaswamy et al., *Design, Analysis and Test of Logic Circuits Under Uncertainty*, Lecture Notes in Electrical Engineering 115, DOI: 10.1007/978-90-481-9644-9_2,

the input combination probabilities. The probability of error for the circuit in Fig. 2.1, given the uniform input distribution, is therefore 0.225. Note that joint probabilities of input combinations, rather than individual input probabilities, are necessary to capture correlations among inputs.

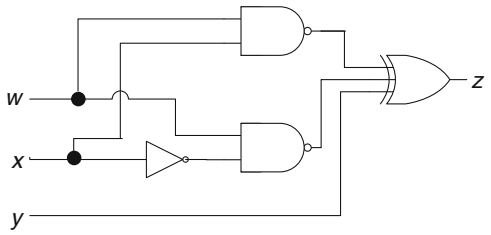
We analyze circuit reliability and other aspects of non-deterministic behavior, using a representation called a probabilistic transfer matrix (PTM). A PTM for a gate (or a circuit) gives the probability of each output combination, conditioned upon the input combinations. PTMs can model gates exhibiting varying input-dependent error probabilities. PTMs form an algebra—a set closed under specific operations—where the operations in question are matrix multiplication and tensor products. These operations may be used to compute overall circuit behavior by combining gate PTMs to form circuit PTMs. Matrix products capture serial connections, and tensor products capture parallel connections.

For those familiar with Bayesian inference, a PTM for a gate is essentially a conditional probability table and reliability analysis is a specific, although, complex form of Bayesian inference. Our aim is to compute the joint probability of the primary outputs given joint probabilities of primary inputs. Circuits offer a very natural way of decomposing the joint probability distribution because essentially, the output probability of a gate only depends on its immediate input probability. Therefore, each gate can be represented by a conditional probability table. However, unlike traditional Bayesian analysis, in this chapter we discuss operations necessary to combine these probability distributions to form a joint probability distribution utilizing algebraic analysis. In the next chapter, we show how to scale this computation to simple circuits using decision-diagram-based compression. Most of the concepts and results described in this chapter also appear in [6, 7].

2.1 PTM Algebra

This section describes the PTM algebra and some key operations for manipulating PTMs. First, we discuss the basic operations needed to represent circuits and to compute circuit PTMs from gate PTMs. Next, we define additional operations to extract reliability information, eliminate variables, and handle fan-out efficiently.

Consider a circuit C with n inputs and m outputs. We order the inputs for the purposes of PTM representation and label them $\text{in}_0, \ldots \text{in}_{n-1}$; similarly, the m outputs are labeled $\text{out}_0, \ldots \text{out}_{m-1}$. The circuit C can be represented by a $2^n \times 2^m$ PTM M. The rows of M are indexed by an n-bit vector whose values range from $\underbrace{000\ldots0}_{n}$ to $\underbrace{111\ldots1}_{n}$. The row indices correspond to input vectors, i.e. 0/1 truth assignments of the circuit's input signals. Therefore, if $\mathbf{i} = i_0 i_1 \ldots i_{n-1}$ is an n-bit input vector, then row $M(\mathbf{i})$ gives the output probability distribution for n input values $\text{in}_0 = i_0, \text{in}_1 = i_1 \ldots \text{in}_{n-1} = i_{n-1}$. Similarly, column indices correspond to truth assignments of the circuit's m output signals. If $\mathbf{j}$ is an m-bit vector, then entry $M(\mathbf{i}, \mathbf{j})$ is the conditional probability that the out-



$$(F_2 \otimes F_2 \otimes I)(I \otimes swap \otimes NOT_p \otimes I)(NAND2_p \otimes NAND2_p \otimes I)(XOR3_p)$$

Fig. 2.1 Sample logic circuit and its symbolic PTM formula

Fig. 2.2 **a** ITM for the circuit in Fig. 2.1; **b** circuit PTM where each gate experiences error with probability $p = 0.1$

	0	1
000	1	0
001	0	1
010	1	0
011	0	1
100	0	1
101	1	0
110	0	1
111	1	0

(a)

	0	1
000	0.756	0.244
001	0.244	0.756
010	0.756	0.244
011	0.244	0.756
100	0.295	0.705
101	0.705	0.295
110	0.295	0.705
111	0.705	0.295

(b)

puts have values $\text{out}_0 = j_0, \text{out}_1 = j_1 \ldots \text{out}_{m-1} = j_{m-1}$ given input values $\text{in}_0 = i_0, \text{in}_1 = i_1 \ldots \text{in}_{n-1} = i_{n-1}$, i.e, $P[\text{outputs} = \mathbf{j} | \text{inputs} = \mathbf{i}]$. Therefore, each entry in M gives the conditional probability that a certain output combination occurs given a certain input combination.

Definition 2.1 Given a circuit C with n inputs and m outputs, the *PTM* for C is a $2^n \times 2^m$ matrix M whose entries are conditional probabilities of the form shown here: $M(\mathbf{i}, \mathbf{j}) = P[\text{outputs} = \mathbf{j} | \text{inputs} = \mathbf{i}]$.

Definition 2.2 A fault-free circuit has a PTM called an *ideal transfer matrix (ITM)* in which the correct logic value of each output occurs with probability 1.

The PTM for a circuit represents its functional behavior for all input and output combinations. An input vector for an n-input circuit is a row vector with dimensions $2^n \times 1$. Entry $v(\mathbf{i})$ of an input vector v represents the probability that the input values $\text{in}_0 = i_0, \text{in}_1 = i_1 \ldots \text{in}_{n-1} = i_{n-1}$ occur. When an input vector is right-multiplied by the PTM, the result is an output vector of size 1×2^m. The output vector gives the resulting output distribution. Examples of an ITM and PTM are shown in Fig. 2.2.

2.1.1 Basic Operations

PTMs can be defined for all the gates of a logic circuit by taking into account errors affecting the gates. A PTM for the entire circuit can then be derived from the PTMs

of the gates and their interconnections. The basic operations needed to compute the circuit PTM from component PTMs are the matrix and tensor products.

Consider the circuit C formed by connecting two gates g_1 and g_2 in series, i.e., the outputs of g_1 are connected to the inputs of g_2. Suppose these gates have PTMs M_1 and M_2; then the entry $M(\mathbf{i}, \mathbf{j})$ of the resulting PTM M for C represents the probability that g_2 produces output j, given g_1 has input i. This probability is computed by summing over all values of intermediate signals (outputs of g_1 which are also inputs of g_2) for input $\mathbf{i}$ of g_1 and output $\mathbf{j}$ of g_2. Therefore, each entry $M(\mathbf{i}, \mathbf{j}) = \sum_{\text{all } \mathbf{h}} M_1(\mathbf{i}, \mathbf{h}) M_2(\mathbf{h}, \mathbf{j})$. This operation corresponds to the ordinary matrix product $M_1 M_2$ of the two component PTMs.

Now suppose that circuit C is formed by two parallel gates g_1 and g_2 with PTMs M_1 and M_2. An entry in the resulting matrix M should represent the joint conditional probability of a pair of input–output values from g_1 and a pair of input–output values from g_2. Each such entry is therefore a product of independent conditional probabilities from M_1 and M_2, respectively. These joint probabilities are given by the tensor product operation.

Definition 2.3 Given two matrices M_1 and M_2, with dimensions $2^k \times 2^l$ and $2^m \times 2^n$, respectively, the *tensor product* $M = M_1 \otimes M_2$ of M_1 and M_2 is a $2^{km} \times 2^{ln}$ matrix whose entries are:

$$M(i_0 \ldots i_{k+m-1}, j_0 \ldots j_{l+n-1}) = M_1(i_0 \ldots i_{k-1}, i_0 \ldots j_{l-1}) \times M_2(i_k \ldots i_{k+m-1}, j_l \ldots j_{l+n-1})$$

Figure 2.3 shows the tensor product of an *AND* ITM with an *OR* ITM. Note that the *OR* ITM appears once for each occurrence of a 1 in the *AND* ITM; this is a basic feature of the tensor product.

Besides the usual logic gates (AND, OR, NOT, etc.), it is useful to define three special gates for circuit PTM computation. These are (i) the n-input identity gate with ITM denoted I_n; (ii) the n-output fan-out gate F_n; and (iii) the swap gate *swap*. These wiring PTMs are shown in Fig. 2.4 .

An n-input *identity gate* simply outputs its input values with probability 1. It corresponds to a set of independent wires or buffers and has the 2×2 identity matrix as its ITM. Larger identity ITMs can be formed by the tensor product of smaller identity ITMs. For instance, the ITM for a 2-input, 2-output identity gate is $I_2 = I \otimes I$. More generally, $I_{m+n} = I_m \otimes I_n$. An n-output *fan-out gate*, F_n, copies an input signal to its n outputs. The ITM of a 2-output fan-out gate, shown in Fig. 2.4b, has entries of the form $F_2(i_0, j_0 j_1) = 1$, where $i_0 = j_0 = j_1$ and all other entries are 0. Therefore, the 5-output fan-out ITM, F_5, has entries $F_5(0, 00000) = F_5(1, 11111) = 1$, with all other entries 0. Wire permutations such as crossing wires are represented by *swap gates*. The ITM for an adjacent-wire swap (a simple two-wire crossover) is shown in Fig. 2.4c. Any permutation of wires can be modeled by a network of swap gates.

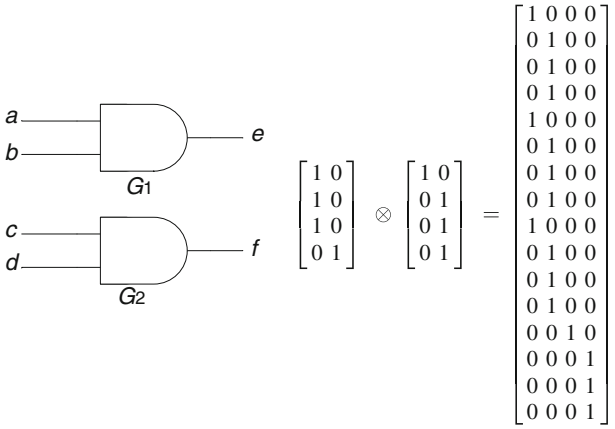


Fig. 2.3 Illustration of the tensor product operation: **a** circuit with parallel AND and OR gates; **b** circuit ITM formed by the tensor product of the AND and OR ITMs

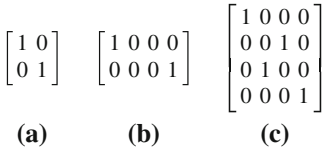
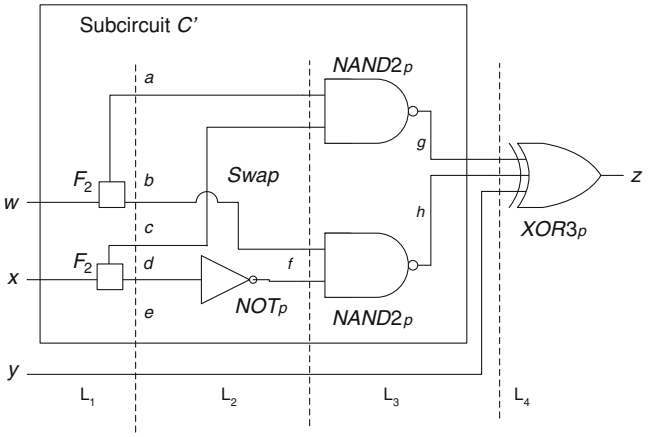


Fig. 2.4 Wiring PTMs: **a** identity gate I; **b** 2-output fan-out gate F_2; **c** wire-swap gate denoted *swap*

Example 2.1 Consider the circuit in Fig. 2.5—this is the circuit of Fig. 2.1 with its wiring gates made explicit. The PTMs for the gates with error probability p are as follows:

$$\underset{NAND2_p}{\begin{bmatrix} p & 1-p \\ p & 1-p \\ p & 1-p \\ 1-p & p \end{bmatrix}} \quad \underset{XOR3_p}{\begin{bmatrix} 1-p & p \\ p & 1-p \\ p & 1-p \\ 1-p & p \\ p & 1-p \\ 1-p & p \\ 1-p & p \\ p & 1-p \end{bmatrix}} \quad \underset{NOT_p}{\begin{bmatrix} p & 1-p \\ 1-p & p \end{bmatrix}}$$

The corresponding circuit PTM is expressed symbolically by the formula in Fig. 2.5. Each parenthesized term in this formula corresponds to a level in the circuit. The advantage of evaluating the circuit PTM using such an expression is that the error probabilities for the entire circuit can be extracted from it.



$$(F_2 \otimes F_2 \otimes I)(I \otimes swap \otimes NOT_p \otimes I)(NAND2_p \otimes NAND2_p \otimes I)(XOR3_p)$$

Fig. 2.5 Circuit to illustrate PTM calculation; vertical lines separate levels of the circuit; the parenthetical subexpressions correspond to logic levels

2.1.2 Additional Operations

In addition to the basic operations of matrix multiplication and tensor product, we introduce the following three operations to increase the scope and efficiency of PTM-based computation:

- *fidelity*. This operation measures the similarity between an ITM and a corresponding PTM. It is used to evaluate the reliability of a circuit.
- *eliminate_variables*. This operation computes the PTM of a subset of inputs or outputs, starting from a given PTM. It can also be used to compute the probability of error of individual outputs.
- *eliminate_redundant_variables*. This operation eliminates redundant input variables that result from tensoring matrices of gates that are in different fan-out branches of the same signal

We now formally define and describe these operations in more detail. First, we define the *element-wise* product used in computing *fidelity*.

Definition 2.4 The element-wise product of two matrices A and B, both of dimension $n \times m$, is denoted $A.*B = M$ and defined by $M(i, j) = A(i, j) \times B(i, j)$.

To obtain the *fidelity*, the element-wise product of the ITM and the PTM is multiplied on the left by the input vector, and the norm of the resulting matrix is computed. In the definition below, $\|\mathbf{v}\| = \Sigma_i \|v_i\|$ denotes the l_1 norm of vector $\mathbf{v}$.

Definition 2.5 Given a circuit C with PTM M, ITM J, and input vector $\mathbf{v}$, the fidelity of M is defined as

$$\mathit{fidelity}(v, M, J) = ||\mathbf{v}(M. * J)||$$

The fidelity of a circuit is a measure of its *reliability* against transient errors. Fig. 2.6 illustrates the *fidelity* computation on the circuit of Fig. 2.1.

Example 2.2 Consider the circuit C from Fig. 2.1, with inputs $\{w, x, y\}$ and output $\{z\}$. The ITM is denoted J, and the PTM, shown in Fig. 2.2b, is denoted M. The circuit PTM is calculated using the PTMs from Example 2.1, with probability of error $p = 0.1$ at each gate, on all inputs. Fig. 2.6 shows intermediate matrices needed for this computation. The quantity $\mathit{fidelity}(v, M, J)$ is found by first element-wise multiplying J and M, then left-multiplying by an input vector v. The l_1 norm of the resulting matrix is $\mathit{fidelity}(v, M, J) = (0.3716 + 0.3716) = 0.7432$. The probability of error is $1 - 0.7432 = 0.2560$.

The *eliminate_variables* operation is used to compute the "sub-PTM" of a smaller set of input and output variables. We formally define it for 1-variable elimination as follows.

Definition 2.6 Given a PTM matrix M that represents a circuit C with inputs $\mathrm{in}_0, \ldots, \mathrm{in}_{n-1}$, $\mathit{eliminate\_variables}(M, \mathrm{in}_k)$ is the matrix M' with $n - 1$ input variables $\mathrm{in}_0, \ldots, \mathrm{in}_{k-1}, \mathrm{in}_{k+1}, \ldots, \mathrm{in}_{n-1}$ whose rows are

$$\begin{aligned} M'(i_0 \ldots i_{k-1} i_{k+1} \ldots i_{n-2}, \mathbf{j}) = {} & M(i_0 \ldots i_{k-1}\ 0\ i_{k+1} \ldots i_{n-2}, \mathbf{j}) \\ & + M(i_0 \ldots i_{k-1}\ 1\ i_{k+1} \ldots i_{n-2}, \mathbf{j}) \end{aligned}$$

The *eliminate_variables* operation is similarly defined for output variables.[1] The elimination of two variables can be achieved by eliminating each of the variables individually in arbitrary order. Fig. 2.7 demonstrates the elimination of column variables from a subcircuit C' of the circuit in Fig. 2.5, formed by the logic between inputs w, x and outputs g, h. The PTM for C' with probability of error $p = 0.1$ on all its gates is given by:

$$(F_2 \otimes F_2)(swap \otimes NOT_p)(NAND2_p \otimes NAND2_p)$$

If we eliminate output h, then we can isolate the conditional probability distribution of output g, and vice versa. Output h corresponds to the second column variable of the PTM in Fig. 2.7b. To eliminate this variable, columns with indices 00 and 01 of Fig. 2.7b are added, and the result is stored in the column 0 of the resultant matrix (Fig. 2.7c). Columns 10 and 11 of M are also added, and the result is stored in column 1 of the resultant matrix. The final PTM gives the probability distribution of output variable g in terms of the inputs w and x. A similar process is undertaken for

[1] The *eliminate_variables* operation is analogs to the existential abstraction of a set of variables x in a Boolean function f [8], given by the sum of the positive and negative cofactors of f, with respect to x: $\exists x\ f = f_x + f_{\overline{x}}$. The *eliminate_variables* operation for PTMs relies on arithmetic addition of matrix entries instead of the Boolean disjunction of cofactors.

$$v^T = \begin{bmatrix} 0.125 \\ 0.125 \\ 0.25 \\ 0.25 \\ 0 \\ 0 \\ 0.125 \\ 0.125 \end{bmatrix} \quad J.*M = \begin{bmatrix} 0.756 & 0 \\ 0 & 0.756 \\ 0.756 & 0 \\ 0 & .756 \\ 0 & 0.705 \\ 0.705 & 0 \\ 0 & 0.705 \\ 0.705 & 0 \end{bmatrix} \quad v(J.*M) = \begin{bmatrix} 0.3716 & 0.3716 \end{bmatrix}$$

(a) **(b)** **(c)**

Fig. 2.6 Matrices used to compute *fidelity* for the circuit in Fig. 2.1: **a** input vector; **b** result of element-wise product of its ITM and PTM; **c** result of left-multiplication by the input vector

$$\begin{array}{c} \begin{matrix} 00 & 01 & 10 & 11 \end{matrix} \\ \begin{bmatrix} 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix} \end{array} \quad \begin{array}{c} \begin{matrix} 00 & 01 & 10 & 11 \end{matrix} \\ \begin{bmatrix} 0.010 & 0.090 & 0.090 & 0.810 \\ 0.010 & 0.090 & 0.090 & 0.810 \\ 0.082 & 0.018 & 0.738 & 0.162 \\ 0.162 & 0.738 & 0.018 & 0.082 \end{bmatrix} \end{array}$$

(a) **(b)**

$$\begin{array}{c} \begin{matrix} 0 & & & 1 \end{matrix} \\ \begin{bmatrix} 0.010+ & 0.090 & 0.090+ & 0.810 \\ 0.010+ & 0.090 & 0.090+ & 0.810 \\ 0.082+ & 0.018 & 0.738+ & 0.162 \\ 0.162+ & 0.738 & 0.018+ & 0.082 \end{bmatrix} \end{array} \quad \begin{array}{c} \begin{matrix} 0 & & & 1 \end{matrix} \\ \begin{bmatrix} 0.010+ & 0.090 & 0.090+ & 0.810 \\ 0.010+ & 0.090 & 0.090+ & 0.810 \\ 0.082+ & 0.738 & 0.018+ & 0.162 \\ 0.162+ & 0.018 & 0.738+ & 0.082 \end{bmatrix} \end{array}$$

(c) **(d)**

Fig. 2.7 Example of the *eliminate_variables* operation: **a** ITM of subcircuit C' from Fig. 2.5; **b** PTM of C'; **c** output variable h eliminated; **d** output variable g eliminated

elimination of g in the PTM of Fig. 2.7d. However, this time the first column variable is eliminated.

Often, parallel gates have common inputs, due to fan-out at an earlier level of logic. An example of this appears in level L_3 of Fig. 2.5 due to fan-out at level L_1. The fan-out gate was introduced to handle such situations; therefore, the PTM for level L_1 in Example 2.1 is composed of two copies of the fan-out PTM F_2 tensored with an identity PTM I. However, this method of handling fan-out can be computationally inefficient because it requires numerous matrix multiplications. Therefore, in either inputs or outputs we introduce a new operation called *eliminate_redundant_variables* to remove redundant signals that are due to fan-out or other causes. This operation is more efficient than matrix multiplication because it is linear in PTM size, whereas matrix multiplication is cubic.

Definition 2.7 Given a circuit C with n inputs $\text{in}_0, \ldots \text{in}_{n-1}$ and PTM M, let in_k and in_l be two inputs that are identified with (connected to) each other. Then $eliminate\_redundant\_variables(M, \text{in}_k, \text{in}_l) = M'$, where M' is a matrix with $n - 1$ input variables whose rows are

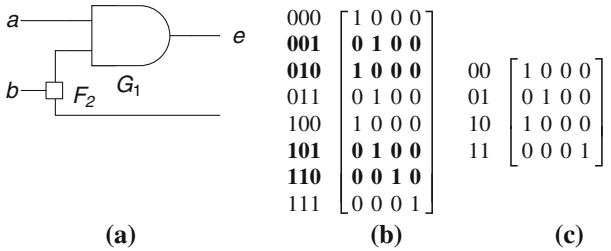


Fig. 2.8 Signal forwarding using *eliminate_redundant_variables*: **a** circuit with signal b fanning out to two different levels; **b** $AND \otimes I$, adding b as an input and output; **c** final ITM for circuit computed by removing rows in boldface

$$M'(i_1 \ldots i_k \ldots i_{l-1}\ i_{l+1} \ldots i_{n-1}, \mathbf{j}) = M(i_1 \ldots i_k \ldots i_{l-1}\ i_k\ i_{l+1} \ldots i_{n-1}, \mathbf{j})$$

The definition of *eliminate_redundant_variables* can be extended to a set of input variables that are redundant. Fig. 2.8 shows an example of this operation.

PTMs yield correct output probabilities despite reconvergent fan-out because the joint probabilities of signals on different fan-out branches are computed correctly using the tensor product and *eliminate_redundant_variables* operations. Suppose two signals on different fan-out branches reconverge at the same gate in a subsequent circuit level. Since the joint probability distribution of these two signals is computed correctly, the serial composition of the fan-out branches with the subsequent gate is also correct, by the properties of matrix multiplication. On the other hand, if the individual signal probabilities are computed separately, then these probabilities cannot be recombined into the joint probability without some loss of accuracy.

The *eliminate_redundant_variables* operation can efficiently handle fan-out to different levels by "signal forwarding," as seen in Fig. 2.8. Signal b is required at a later level in the circuit; therefore, b is added to the ITM as an output variable by tensoring the AND ITM with an identity matrix. However, tensoring with the identity ITM adds both an input and output to the level. Hence, the additional input is redundant with respect to the second input of the AND gate and is removed using *eliminate_redundant_variables*. Note that the removed rows correspond to assigning contradictory values on identical signals.

2.1.3 Handling Correlations

There are many cases of errors where input and output values cannot be separated and combinations of these values must be taken into account. For example, using the *eliminate_variables* operation, the conditional probabilities of the inputs or outputs cannot always be stored separately in different matrices. While such storage can alleviate the input-space explosion inherent in storing all possible combinations of inputs and outputs, it may not capture correlations within the circuit.

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0.75 & 0.25 & 0 \\ 0 & 0.25 & 0.75 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad \begin{bmatrix} 1 & 0 \\ 0.75 & 0.25 \\ 0.25 & 0.75 \\ 0 & 1 \end{bmatrix} \quad \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0.75^2 & (0.75)(0.25) & (0.25)(0.75) & 0.25^2 \\ 0.25^2 & (0.75)(0.25) & (0.25)(0.75) & 0.75^2 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

(a) **(b)** **(c)**

Fig. 2.9 Example of output inseparability: **a** PTM for a probabilistic wire-swap; **b** PTM for each individual output after applying *eliminate_variables*; **c** incorrect result from tensoring two copies of the PTM from part **b** and applying *eliminate_redundant_variables*

Example 2.3 Suppose two wires have a 0.25 probability of swapping. The matrix corresponding to this error is given in Fig. 2.9a. If we try to separate the probability of each output, using *eliminate_variables*, the output probabilities both have the PTM of Fig. 2.9b. If these outputs are tensored (with redundant inputs eliminated), they result in the erroneous combined matrix of Fig. 2.9c. This demonstrates that these two outputs cannot be correctly separated; their joint conditional distributions are, in fact, inseparable.

Just as some errors cannot be separated, some faults affect multiple gates simultaneously. In this case, the combined PTM cannot be built from individual PTMs, and the joint probabilities must be obtained (or the exact correlation determined). This same effect can occur with input vectors that cannot always be separated into probabilities of individual inputs. An example is given below.

$$\begin{matrix} 00 & 01 & 10 & 11 \end{matrix}$$
$$\begin{bmatrix} 0.5 & 0 & 0 & 0.5 \end{bmatrix}^T$$

PTMs have the advantage that, at every level, they can represent and manipulate joint probabilities from the inputs to the outputs. If necessary, individual output distributions can be obtained using the *eliminate_variables* operation.

So far, we have introduced the PTM representations of gate and wire constructs, and the operations needed to combine them into circuit PTMs. In the next section, we give examples of the various kinds of faults that PTMs can capture, as well as the application of PTMs to soft-error analysis and error-threshold computation.

2.2 Applications

In this section, we discuss applications of PTMs to various fault types as well as in determining the error-transfer behavior of logic circuits.

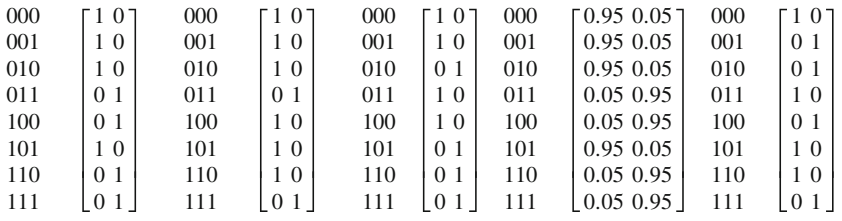


Fig. 2.10 PTMs for various types of gate faults: **a** a fault-free ideal 2-1 MUX gate (select line is the 3rd input); **b** first input signal stuck-at 1; **c** first two input signals swapped; **d** probabilistic output bit-flip with $p = 0.05$; **e** wrong gate: MUX replaced by 3-input XOR gate.

2.2.1 Fault Modeling

The PTM model can represent a wide variety of faulty circuit behaviors, including both hard and soft physical faults, and design errors. The fact that there are separate probabilities for each input and output, and the fact that they are propagated simultaneously make this possible. Fig. 2.10 lists some fault/error types that can be represented by PTMs.

Figure 2.10a shows the ITM for a fault-free ideal 2-1 multiplexer (MUX). Fig. 2.10b shows the first data input signal of the MUX stuck-at 1, i.e., row 000 is replaced with row 100 of the ITM, row 010 with row 111, and so forth. Fig. 2.10c shows an example where the first two wires have been swapped; this is captured by permuting the rows of the ITM, accordingly. Fig. 2.10d shows the first example of a probabilistic error, an output bit-flip where the wrong value occurs with probability $p = 0.05$ in each row. Fig. 2.10e shows a design error where a MUX has been replaced by a 3-input XOR gate. As these examples indicate, PTMs can capture both gate and wiring errors.

PTMs can also represent faults that are likely to occur in nanoscale circuits. For instance, in QCA, the wires themselves are made of "quantum dots," and so, like gates, wires can experience bit-flips. Such bit-flips on wires can be represented by the 1-input identity gate I, with probabilities as shown below.

$$\begin{bmatrix} 1-p & p \\ 1 & 1-q \end{bmatrix}$$

2.2.2 Modeling Glitch Attenuation

Thus far, signals have been described by their logic value, with each signal represented by a 1×2 vector that indicates the probability of it being 0 or 1. While retaining

the discreteness of our model, we now expand signal representation to incorporate some necessary electrical characteristics.

For instance, we can differentiate between signals of long and short duration, just as we differentiate between signals with high and low amplitude by their logic value. We can represent a signal by a vector w which has four entries instead of two, $w = [p_{0s}\ \ p_{0l}\ \ p_{1s}\ \ p_{1l}]$. The second bit of the row index represents short ("s") or long ("l") duration, so p_{0s} is the probability of a logic 0 with short duration. Extraneous glitches, such as those induced by SEUs, are likely to have short duration, while driven logic signals are likely to have relatively long duration.

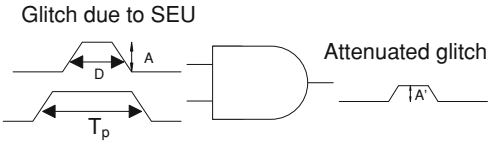
Each gate in a circuit has a probability of an SEU strike that depends upon various environmental factors, such as neutron flux and temperature. We call this the *probability of occurrence* for a gate (or node) g, and denote it by $p_{\text{occur}}(g)$. However, SEU strikes create glitches which can be differentiated by a combination of shape and amplitude. These distinctions are important for the propagation of a glitch through circuit gates. Therefore, we utilize a modified identity matrix denoted $I_{1,n}(p_{\text{occur}})$ to represent a probability distribution on a glitch induced by an SEU strike.

We use the glitch-propagation model from [9] to determine which signal characteristics to capture; a different model might require other characteristics to be represented. In [9], glitches are classified into three types depending on their duration D and amplitude A, relative to the gate propagation delay T_p, as well as the threshold voltage V_t. Glitches are assumed to change only logic 0 to logic 1 when they occur, but they can be inverted later.

- Glitches of type 1 have amplitude $A > V_t$ and duration $D > 2T_p$. Glitches of this type are propagated without attenuation.
- Glitches of type 2 have amplitude $A > V_t$ and duration $2T_p > D > T_p$. Glitches of this type are propagated with an attenuated amplitude of $A' < A$.
- Glitches of type 3 have $A < V_t$. Glitches of this type are not propagated, i.e., they are electrically masked.

Since amplitude is already indicated by the logic value, an additional bit is used to indicate whether the duration is larger or smaller than the propagation delay of the gate (when the amplitude is higher than the threshold voltage). The duration is irrelevant for glitches with amplitude lower than the threshold voltage, since these are likely to be attenuated. Fig. 2.11a shows the probability distribution of an SEU strike when the correct logic value is 0. Glitches of type 1 are indicated by row labels 11, glitches of types 2 are indicated by labels 10, and glitches of type 3 are indicated by 01. In particular, Fig. 2.11a assumes uniform distribution with respect to glitches.

Once an SEU strikes a gate g and induces a glitch, the electrical characteristics of the circuit gates determine whether the glitch is propagated. Glitches with long duration and high energy relative to the gate propagation delay and threshold voltage are generally propagated; other glitches are normally quickly attenuated. We call the probability that a glitch is propagated $p_{\text{prop}}(g)$. The glitch-transfer characteristics of a logic gate are described by a modified gate PTM that represents relevant characteristics of the glitch. For instance, Fig. 2.11b shows a modified *AND* PTM, denoted $AND_{2,2}(p_{\text{prop}})$.



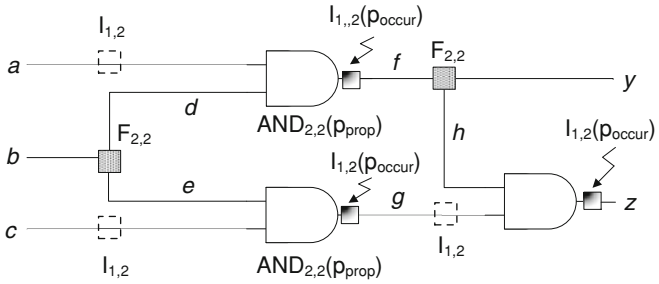
$$\begin{array}{c} \\ 00 \\ 01 \\ 10 \\ 11 \end{array} \begin{bmatrix} 1-p_{occur} & p_{occur}/3 & p_{occur}/3 & p_{occur}/3 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

(a)

$$\begin{array}{c} 00,00 \\ 00,01 \\ 00,10 \\ 00,11 \\ 01,00 \\ 01,01 \\ 01,10 \\ 01,11 \\ 10,00 \\ 10,01 \\ 10,10 \\ 10,11 \\ 11,00 \\ 11,01 \\ 11,10 \\ 11,11 \end{array} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 1-p_{prop} & p_{prop} & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 1-p_{prop} & p_{prop} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

(b)

Fig. 2.11 PTMs for SEU modeling where the row labels indicate input signal type: **a** $I_{2,2}(p_{\text{occur}})$ describes a probability distribution on the energy of an SEU strike at a gate output, **b** $AND_{2,2}(p_{\text{prop}})$ describes SEU-induced glitch propagation for a 2-input AND gate. The type-2 glitches become attenuated to type 3 with a probability $1 - p_{\text{prop}}$



$$CircuitITM = (I_{1,2} \otimes F_{2,2} \otimes I_{1,2})(AND_{2,2}(p_{prop}) \otimes AND_{2,2}(p_{prop}))(F_{2,2} \otimes I_{1,2})(I_{1,2} \otimes AND_{2,2}(p_{prop}))$$
$$CircuitPTM = (I_{1,2} \otimes F_{2,2} \otimes I_{1,2})(AND_{2,2}(p_{prop})I_{1,2}(p_{occur}) \otimes AND_{2,2}(p_{prop})I_{1,2}(p_{occur}))$$
$$(F_{2,2} \otimes I_{1,2})(I_{1,2} \otimes AND_{2,2}(p_{prop})I_{1,2}(p_{occur}))$$

Fig. 2.12 Circuit with ITM and PTMs describing an SEU strike and the resultant glitch propagation with multi-bit signal representations

In the selected glitch model [9], attenuation acts by transforming sensitized glitches of type 2 with a certain probability, into glitches of type 3. All other signals retain their original output values given by the logic function of the gate. This transfer function can be described by the PTM of Fig. 2.11b. This PTM shows an AND gate which propagates an input glitch (only if the other input has a non-controlling value), with certainty if the glitch is of type 1 (in which case it is indistinguishable from a driven logic value) or with probability p_{prop} if the glitch is of type 2.

When using 2-bit signal representations, the probability of a logic 1 value for a signal is computed by *marginalizing*, or summing-out, over the second bit. For

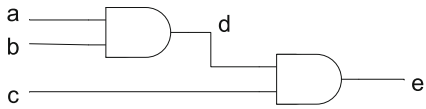


Fig. 2.13 Circuit used in Example 2.4 to illustrate the incorporation of electrical masking into PTMs

Fig. 2.14 PTM incorporating electrical properties of gates for the circuit in Example 2.4

$$fidelity = .99994791$$

$$Perror = 1 - fidelity = .000052083$$

$$\begin{bmatrix}
1 & 0 & 0 & 0 \\
\vdots & & & \\
1 & 0 & 0 & 0 \\
0.9996 & 0.0001 & 0.0003 & 0 \\
0.9996 & 0.0002 & 0.0002 & 0 \\
1 & 0 & 0 & 0 \\
1 & 0 & 0 & 0 \\
1 & 0 & 0 & 0 \\
1 & 0 & 0 & 0 \\
1 & 0 & 0 & 0 \\
1 & 0 & 0 & 0 \\
0.2500 & 0.1875 & 0.5625 & 0 \\
0.2500 & 0.3750 & 0.3750 & 0 \\
1 & 0 & 0 & 0 \\
1 & 0 & 0 & 0 \\
0.5000 & 0.1250 & 0.3750 & 0 \\
0.5000 & 0.2500 & 0.2500 & 0 \\
1 & 0 & 0 & 0 \\
1 & 0 & 0 & 0 \\
0.9995 & 0.0002 & 0.0003 & 0 \\
0.9995 & 0.0001 & 0.0001 & 0.0003 \\
1 & 0 & 0 & 0 \\
1 & 0 & 0 & 0 \\
1 & 0 & 0 & 0 \\
1 & 0 & 0 & 0 \\
1 & 0 & 0 & 0 \\
1 & 0 & 0 & 0 \\
0 & 0.1250 & 0.3750 & 0 \\
0 & 0.2500 & 0.2500 & 0 \\
1 & 0 & 0 & 0 \\
1 & 0 & 0 & 0 \\
0 & 0.5000 & 0.5000 & 0 \\
0 & 0 & 0 & 1
\end{bmatrix}$$

instance, if a signal has 2-bit distribution [0.2 0.1 0.3 0.4], since the second bit indicates duration, the probability of a logic 0 is $0.2 + 0.1$ and the probability of a logic 1 is $0.3 + 0.4$. Fig. 2.12 shows a circuit with the corresponding ITM and PTMs with multi-bit signal representations.

Example 2.4 For the circuit in Fig. 2.13, suppose an SEU strike produces a glitch at input b. By inspection, we see that this glitch will only propagate to primary output e for the primary input combination 101. In other words, the glitch propagates if the input sensitizes the appropriate path to d and then e. If we let $p_{\text{occur}} = 0.001$ and $p_{\text{prop}} = 0.5$, and $AND_{2,2}(p_{\text{prop}})$ is as shown in Fig. 2.11, then the circuit PTM is given by:

Table 2.1 Polynomial approximations of circuit and residual errors. The fitted polynomials are of the form $e(x) \approx a_0 + a_1 x + a_2 x^2 + a_3 x^3 \ldots$

Circuit	Error	Polynomial coefficients						
		a_0	a_1	a_2	a_3	a_4	a_5	a_6
Majority	2.5 E−7	0.2080	0.1589	0	0	0	0	0
MUX	6.6 E−6	0.0019	1.9608	−2.8934	1.9278	0	0	0
Parity	0.0040	0.0452	5.4892	−21.4938	31.9141	−4.2115	−30.3778	19.5795
tcon	0.0019	0.0152	6.2227	−13.5288	7.1523	9.2174	−9.0851	0
9symml	0.0010	0.0250	2.4599	−3.7485	1.5843	0	0	0
XOR5	0.0043	0.0716	5.9433	−26.4666	51.1168	−44.6143	14.4246	0

$$(I_2 \otimes I_{2,2}(p_{\text{occur}}) \otimes I_2)(AND_{2,2(p_{\text{prop}})} \otimes I_2)(AND_{2,2(p_{\text{prop}})})$$

The corresponding PTM and fidelity are given in Fig. 2.14.

2.2.3 Error Transfer Functions

In this section, we analyze circuit reliability as a function of gate reliability. Using data points for various gate error values, we derive low-degree polynomial approximations for the error transfer functions of some benchmark circuits. Such functions can be used to derive upper bounds for tolerable levels of gate error.

Definition 2.8 The *error transfer function* $e(x)$ on $0 \leq x \leq 1$ of a circuit C is the *fidelity* of C with output-error probability x on all gates.

Figure 2.15 illustrates the error-transfer functions for several standard benchmark circuits, determined by introducing varying amounts of error into gates and then calculating the circuit fidelity according to Definition 2.5. Generally, such error transfer curves can be described by polynomials. If two gates experience errors with probability $p > 0$, then their serial and parallel compositions experience errors with probability $O(p^2)$. If a circuit has n gates, each with error p, then its fidelity is a polynomial in p of degree n. Realistically, only gate error values under 0.5 are useful since the gate can simply be viewed as its negated version for higher error values. However, Fig. 2.15 uses probabilities of gate error up to 1 to emphasize the polynomial nature of the curves.

Table 2.1 gives low-degree polynomials that estimate error transfer functions with high accuracy. Such functional approximations are useful in determining the upper bounds on gate error probability necessary to achieve acceptable levels of circuit error. For instance, it has been shown that replication techniques such as TMR or NAND-multiplexing only decrease circuit error if the gate error is strictly less than 0.5 [10]. However, Fig. 2.15 suggests that for most circuits, replicating the entire circuit at gate errors of 0.20 or more will only increase circuit error.

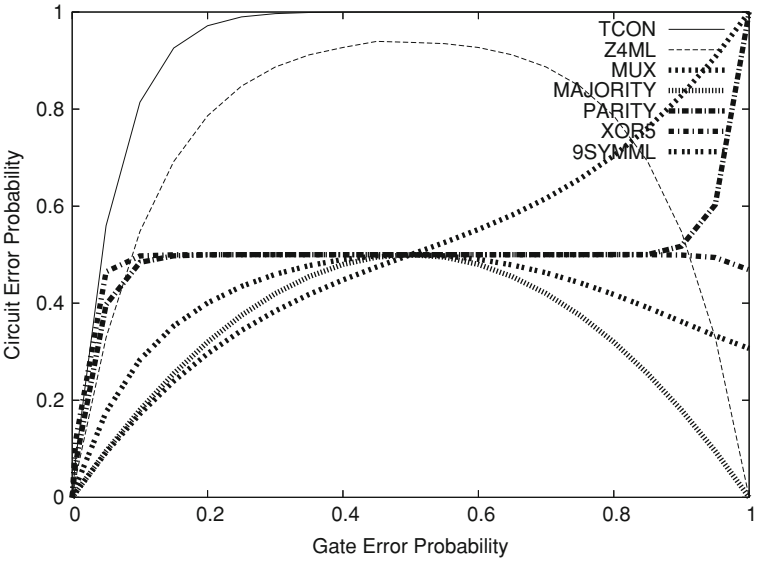


Fig. 2.15 Circuit error probability under various gate error probabilities

References

1. Kummamuru RK et al (1993) Operation of quantum-dot cellular automata (QCA), shift registers and analysis of errors. IEEE Trans Electron Devices 50–59:1906–1913
2. Rejimon T, Bhanja S, (2006) Probabilistic error model for unreliable nano-logic gates. In: Proceedings of NANO, pp 47–50
3. Parker KP, McCluskey EJ (1975) Probabilistic treatment of general combinational networks. IEEE Trans Comput C-24(6):668–670
4. Ercolani S et al. (1989) Estimate of signal probability in combinational logic networks. In: Proceedings of European test conference, pp 132–138
5. Savir J, Ditlow G, Bardell PH (1983) Random pattern testability. In: Proceedings of FTCS, pp 80–89
6. Krishnaswamy S, Viamontes GF, Markov IL, Hayes JP (2005) Accurate reliability evaluation and enhancement via probabilistic transfer matrices. In: Proceedings of DATE, pp 282–287
7. Krishnaswamy S, Viamontes GF, Markov I L, Hayes JP (2008) Probabilistic transfer matrices in symbolic reliability analysis of logic circuits. ACM Trans Des Autom Electron Syst 13(1):1–35 article 8
8. Hachtel G, Somenzi F (1996) Logic synthesis and verification algorithms. Kluwer Academic Publishers, Boston
9. Omana M et al. (2003) A model for transient fault propagation in combinatorial logic. In: Proceedings of IOLTS, pp 11–115
10. Pippenger N (1998) Reliable computation by formulas in the presence of noise. IEEE Trans Inf Theory 34(2):194–197

Chapter 3
Computing with Probabilistic Transfer Matrices

Circuit PTMs have exponential space complexity in the worst case, because they contain information about all possible input vectors. This makes numerical computation with PTMs impractical for circuits with more than 10–15 inputs. To address this, we develop an implementation of the PTM framework that uses algebraic decision diagrams (ADDs) to represent ADDs in compressed form. We further derive ADD algorithms to combine PTMs directly in their compressed forms.

Figure 3.1 gives a PTM for the circuit in Fig. 2.1 along with the corresponding ADD for the case where all gates experience output bit-flips with probability $p = 0.1$. It can be seen that the terminal nodes at the bottom of the ADD comprise all distinct entries of the PTM. As explained later, the ADD contains exactly the same information about input–output error probabilities as the PTM. However, an ADD often has far fewer nodes than the number of entries in the PTM, and so requires much less storage space and processing time.

Developing efficient ADD algorithms for PTM operations is a significant technical challenge that we address in this chapter. We first adapt previous ADD algorithms from [1, 2] for tensor and matrix products. The original versions of these algorithms handle only square matrices, while PTMs are generally rectangular. In addition, we develop ADD algorithms for the new operations defined in Chap. 2. These operations are necessary for computing marginal probability distributions, reconciling dimensions, and estimating overall circuit-error probabilities.

In the second part of this chapter, we develop several methods to further improve the scalability of PTM-based analysis. These methods employ the following techniques: partitioning and hierarchical computation, dynamic evaluation ordering, and input-vector sampling. Most of the techniques and results described in this chapter also appear in [3, 4].

3.1 Compressing Matrices with Decision Diagrams

This section discusses the compression of PTMs using algebraic decision diagrams ADDs, and develops a procedure for computing circuit PTMs from gate PTMs.

S. Krishnaswamy et al., *Design, Analysis and Test of Logic Circuits Under Uncertainty*, Lecture Notes in Electrical Engineering 115, DOI: 10.1007/978-90-481-9644-9_3,

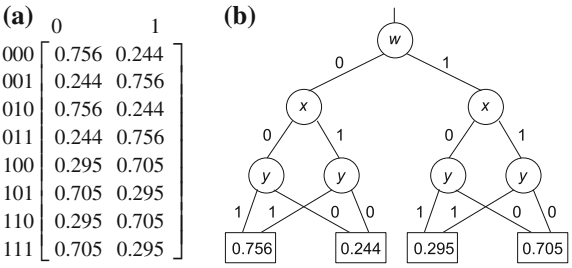


Fig. 3.1 **a** PTM for the circuit in Fig. 2.1 where each gate has error probability $p = 0.1$; **b** the corresponding ADD

Recall [5, 6] that a binary decision diagram (BDD) is a directed acyclic graph representing a Boolean function $f(x_0, x_1, x_2, \ldots, x_n)$ with root node x_0. The subgraph formed by the outgoing edge labeled 0 represents the negative cofactor $f_{x_0'}(x_1, \ldots, x_n)$, or the *else* BDD. The subgraph formed by the outgoing edge labeled 1 represents the positive cofactor $f_{x_0}(x_1, \ldots, x_n)$, or the *then* BDD. Boolean constants are represented by terminal nodes that are labeled 0 or 1. ADDs are variants of BDDs in which terminal nodes can take on any real numerical values. When used to represent a PTM, the terminal nodes are labeled with the probability values that appear as entries in the PTM, as the following example illustrates.

Example 3.1 Consider the 8×2 PTM in Fig. 3.1a, which represents a certain faulty circuit (Fig. 2.1) with three inputs w, x and y, and a single output z. This particular PTM assumes that every gate has an error probability $p = 0.1$. It explicitly gives the probability of each possible output value of z for every possible input combination wxy. For example, if $wxy = 010$, the probability of $z = 1$ is $P\{z(0, 1, 0) = 1\} = 0.244$, as defined by the entry in the third row and second column of the PTM. The same information can be extracted from the ADD of Fig. 3.1b by tracing a path defined by the given wxy values from the top (root) node of the ADD to the bottom, where the terminal node gives the probability of $z = 1$. In the present instance with $wxy = 010$, the path starts at ADD node w, follows the 0-branch to x, then follows the 1-branch from x to y, and finally takes the 0-branch from y to a terminal node, which is labeled 0.244, the correct value of $P\{z(0, 1, 0) = 1\}$. The value of $P\{z(0, 1, 0) = 0\}$ is implicitly given by $1 - P\{z(0, 1, 0) = 1\} = 1 - 0.244 = 0.756$. Note that the number of terminal nodes is the same as the number of distinct entries of the PTM, which is four in the case of Fig. 3.1a. Hence, the more repeated entries in the PTM (an indication of its regularity), the smaller the corresponding ADD. Any top-to-bottom ordering of the variables can be used in the ADD, but the size and structure of the resulting graph will vary. Thus, Fig. 3.1b is just one of the many possible ADD representations of the given PTM.

Bahar et al. [1] present a method of encoding a real-valued matrix using an ADD. The ADD encoding of a matrix M is a rooted directed acyclic graph whose entries depend on the interleaved row- and column-index variables $(r_0, c_0, r_1, c_1, \ldots, r_n, c_n)$ of M; branches of the ADDs correspond to portions of the matrix. The root of the ADD is the node labeled r_0. The subgraph formed by the outgoing edge labeled 0

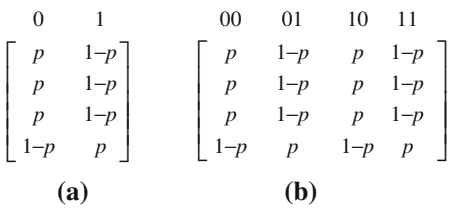


Fig. 3.2 PTMs with identical ADDs without zero padding: **a** matrix with only one column variable; **b** matrix without dependency on the second column variable

represents the top half of M, i.e., the half corresponding to $r_0 = 0$. The subgraph formed by the outgoing edge labeled 1 represents the bottom half of M, which has $r_0 = 1$. As in BDDs, the same path can encode several entries if variables are skipped. The input variables are queried in a predefined order and facilitate reductions, through the use of a single subgraph for identical submatrices.

We use the QuIDDPro library [2] to encode PTMs as ADDs. We also added functions to this library for performing operations on PTMs. QuIDDPro includes the CUDD library [7] and uses interleaved row and column variable ordering, which facilitates fast tensor products and matrix multiplications—key operations in the quantum–mechanical simulations for which QuIDDPro was designed. The basic ADD functions used in PTM computations are as follows:

- $topvar(Q)$: returns the root node of an ADD Q
- $then(Q)$: returns the 1 branch
- $else(Q)$: returns the 0 branch
- $ite(Q, T, E)$: refers to the $if\text{-}then\text{-}else$ operation, which takes a node Q corresponding to the root and two ADDs, T and E, corresponding to the $then$ and $else$ branches, and combines them into a larger ADD.

All matrix algorithms for ADDs that we are aware of, assume square matrices but can represent non-square matrices using zero padding [1, 8]. Zero padding is necessary in ADDs to distinguish between missing row or column variables and those that do not exist because of matrix dimensions—a non-square matrix has fewer row variables than column variables, or vice versa. Recall that ADD variables are ordered, and nodes are levelized by decision variables. Any variable missing from the ADD can be wrongly interpreted as marking replicated matrix entries; Fig. 3.2 illustrates a situation in which missing variables can create ambiguity.

Figure 3.3 describes an algorithm for padding matrices with zeros. This algorithm assumes that there are more row variables than column variables, but can easily be modified to handle cases with more column variables than row variables. Suppose a PTM with ADD A has 2^{m+1} rows and 2^m columns. The zero padding of A is done by introducing a new node, q, with $then(q)$ pointing to the original ADD and $else(q)$ pointing to the zero terminal. In Fig. 3.3, the function $shift\_col\_var\_labels$, by shifting the column variable number up to facilitate the introduction of missing variables into the ADD, renames nodes representing column variables.

The introduction of zero padding is sufficient to implement the matrix multiplication operation. However, the tensor products of zero-padded PTMs are generally

```
pad_with_zeros(Q)
{
  diff = num_row_vars(Q) − num_col_vars(Q)
  shift_col_var_labels(Q, diff)
  R = Q
  for (i = 0; i < diff; i++)
    R = add_missing_var(R, i)
  return R
}

add_missing_var(Q, i)
{
  c_i = create_new_node(i)
  if (topvar(Q) < c_i && then(Q) > c_i)
    T = ite(c_i, 0, else(Q))
    E = ite(c_i, 1, then(Q))
    R = ite(topvar(Q), T, E)
  else
    T = add_missing_var(then(Q), i)
    E = add_missing_var(else(Q), i)
    R = ite(topvar(Q), T, E)
  return R
}
```

Fig. 3.3 Algorithm to pad matrices with zeros

$$\begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \otimes \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

(a) **(b)** **(c)**

Fig. 3.4 **a** NOT gate ITM; **b** zero-padded NAND gate ITM; **c** their tensor product with incorrect placement of all-zero columns

incorrect. Fig 3.4 shows an example of an ideal NOT gate tensored with an ideal zero-padded NAND gate that yields an incorrect resultant PTM. Columns 3 and 4 of this matrix erroneously consist entirely of zeros carried over from the zero padding of the NAND PTM.

To reconcile tensor products with zero padding, we add dummy outputs to a gate PTM to equalize the number of inputs and outputs. In order to add a dummy output to a gate matrix, we can simply forward one of its input signals to the output, as is done in Fig. 2.8. Dummy outputs can be subsequently removed by eliminating the corresponding column variable. Since *eliminate_variables* removes a variable, it may be necessary to re-pad the matrix with zeros. In such cases, the zero padding is restored using the algorithm given in Fig. 3.3.

```
struct gate
{
  PTM[][]
  node *ADD
  DDmanager M
}

compute circuit ADD(circuit C, errors E)
{
 gatemap = read_into_gates(C)
 sort_topological_levels(gatemap)
 add errors(gatemap, E)
 for(each gate g ∈ gate_map)
  gatemap[g].ADD = convert_to_ADD(gatemap[g].PTM)
  pad with_zeros(gatemap[g])
 for(each level l ∈ gatemap)
  levelgates = order_outputs_by_previous_level_inputs(l)
  for(each gate h ∈ levelgates)
   levelADD = tensor_ADD(levelADD, h.ADD)
   zero track(levelADD)
   redvars = find_redundant_inputs(levelADD)
    eliminate_redundant_variables(levelADD, redvars)
  circuitADD = multiply_ADD(circuitADD, levelADD)
  delete(levelADD)
 return circuitADD
}
```

Fig. 3.5 Algorithm to compute the ADD representation of a circuit PTM. The *gate* structure stores a gate's functional information, including its PTM, input names, output names, and ADD

3.1.1 Computing Circuit PTMs

We present an algorithm for computing the ADD of a circuit PTM in Fig. 3.5, following the method illustrated in Example 2.1 of Chap. 2. First, a gate library is specified in the form of a set of gate PTMs. The circuit (in BLIF format) is read into a data structure that stores its individual gates and wiring structure. The gates are reverse-topologically sorted, from primary outputs to primary inputs, and the subsequent computation proceeds by topological level. Next, the gate PTMs are converted into ADDs. The ADDs for gates at each level are tensored together, zero padding is performed, and finally, the *eliminate_redundant_variables* operation is applied to eliminate dummy outputs. The ADD representing each level, called *levelADD* in Fig. 3.5, is multiplied with the accumulated circuit ADD computed thus far, which is called *circuitADD*. After all levels are multiplied together, the computation of the *circuitADD* is complete. An added subtlety is that the signals of adjacent levels have to be properly aligned, i.e., the outputs of current level have to match with the inputs of the previous level in order for the multiplication to be performed correctly. This can be done by appropriately permuting the row and column variables of the ADDs.

```
eliminate_redundant_variables(Q, v1, v2)
{
 if(isconstant(Q)) return Q
 if(topvar(Q) == v1)
   T = comp_remove_branch(then(Q), v2, 1)
   E = comp_remove_branch(else(Q), v2, 0)
 else
   if(topvar(then(Q)) ≤ v1)
     T = comp_remove_branch(then(Q), v2, 1)
 else
   T = eliminate_redundant_variables(then(Q), v1, v2)
 if(topvar(else(Q)) ≤ v1)
   E = comp_remove_branch(else(Q), v2, 0)
 else
   E = eliminate_redundant_variables(else(Q), v1, v2)
 R = ite(topvar(Q), T, E)
 return R
}

comp_remove_branch(Q, v2, const)
{
 if(table_lookup(Q, v2, const) != NULL)
   return table_lookup(Q, v2, const)
 if(topvar(Q) == v2)
   if(const == 1) return then(Q)
   else return else(Q)
 else if(topvar(Q) ≤ v2)
   return Q
 else
   T = comp_remove_branch(then(Q), v2, const)
   E = comp_remove_branch(else(Q), v2, const)
   R = ite(topvar(Q), T, E)
   table_insert(R, Q, v2, const)
 return R
}
```

Fig. 3.6 Algorithm to eliminate redundant variables

A detail not shown in Fig. 3.5 is that when a circuit has fanout branches to multiple levels, the gate is placed at the first level at which it is needed, and its output is forwarded to other levels using the method shown in Fig. 2.8. The intermediate-level ADDs are discarded after they are multiplied with the *circuitADD*. This is important for the scalability of the implementation because *levelADDs* are the tensor products of several gate ADDs and can have large memory complexity.

In place of fanout gates, we use the *eliminate_redundant_variables* operation (Definition 2.7), whose ADD implementation is given in Fig. 3.6. By removing each duplicated (due to fanout) input signal, the number of levels decreases and multiplications are saved. Previously computed partial results of the *eliminate_redundant_variables* operation are stored in a common hash table, which is searched first to avoid traversing common paths or recomputing existing results.

```
compute_fidelity(Q1, Q2, v)
{
  Q3 = apply(Q1, Q2, *)
  R = multiply_ADD(v, Q3)
  P = sum_probs(R, 1)
  return P
}

sum_probs(Q, mult)
{
  if(table_lookup(Q, mult) != NULL)
    return table_lookup(Q, mult)
  sum = 0
  mult* = 2
  if(isconstant(Q))
    return value(Q) * mult
  if(topvar(Q) + 1! = topvar(then(Q)))
    multt = multt + 2
    sum+ = sum_probs(then(Q), multt)
  if(topvar(Q) + 1! = topvar(else(Q)))
    multe = mult + 2
    sum+ = sum_probs(else(Q), multe)
  table_insert(sum, Q, mult)
  return sum
}
```

Fig. 3.7 Algorithm to compute fidelity

In Fig. 3.6, capitalized variables refer to ADDs, and lower case variables refer to nodes. This algorithm searches the ADD, starting from the root, for the first of two redundant variables v_1, v_2 with $v_1 < v_2$ in the ADD node ordering. Whenever v_1 is found on a path, it traverses down $then(v_1)$ until v_2 is found. It eliminates the v_2 node and points the preceding node to $then(v_2)$. Next, it traverses down $else(v_1)$ and searches for v_2; this time it eliminates v_2 and points the preceding node to $else(v_2)$. This process can be repeated in cases where there are more redundant variables. Both *eliminate_variables* and *eliminate_redundant_variables* are operations that can disturb the equality between row and column variables, since they both remove variables. Therefore, it may be necessary to introduce zero padding (Fig. 3.3).

Once the ADDs for the PTM and ITM of a circuit are known, we can compute the $fidelity$ of the circuit to extract reliability information; see Fig. 3.7. The $fidelity$ algorithm first takes the element-wise product of the ADD for the ITM with the ADD for the PTM, and then performs a depth-first traversal to sum probabilities of correctness. The traversal of the ADD sums the terminal values while keeping track of skipped nodes. A skipped node in an ADD is an indication that the terminal value is repeated for 2^x times, where x depends on the skipped variable's ordering. Note that the ADD implementations of *eliminate_redundant_variables*, $fidelity$, and *eliminate_variables* run in linear time in the size of the ADD arguments.

Table 3.1 Statistics on various small benchmarks

Circuit	Size		Reliability, $p = 0.05$		Number of	$p = 0$		$p = 0.05$	
	Number of gates	Circuit width	Two-way	One-way	ADD nodes	Memory (MB)	Time (s)	Memory (MB)	Time (s)
C17	6	5	0.846	0.880	2.00E3	1.090	0.002	0.071	0.313
mux	6	23	0.907	0.939	1.35E4	26.13	3.109	8.341	2.113
z4ml	8	20	0.670	0.817	7.01E3	6.594	1.113	3.030	0.8400
x2	12	23	0.150	0.099	2.85E4	11.015	2.344	237.9	10.52
parity	15	23	0.602	0.731	1.96E3	1.060	0.113	0.337	0.262
pcle	16	16	0.573	0.657	5.46E5	28.59	6.160	4.196E1	4.300
decod	18	13	0.000	0.000	2.76E4	30.15	1.020	5.690E2	11.80
cu	23	23	0.461	0.579	1.06E5	13.39	2.176	2.155E1	3.430
pm1	24	27	0.375	0.596	4.55E5	77.66	5.031	2.155E2	13.34
9symml	44	37	0.327	0.534	1.05E7	4445	552.7	5.341E3	696.2
xor5	47	19	0.067	0.071	4.67E4	46.72	3.539	10.556E3	19.58

Results from the calculation of circuit ITMs, PTMs, and $fidelity$ are listed in Table 3.1. We used the smaller LGSynth 91 and LGSynth 93 benchmarks with uniform input distributions. These simulations were conducted on a Linux workstation with a 2 GHz Pentium 4 processor. In these experiments, CPU time was limited to 24 h. The runtimes and memory requirements are sensitive to the width of a circuit, i.e., the largest number of signals at any level. Empirically, circuits with widths of around 40 signals can be evaluated. In these experiments, we calculate entire circuit PTMs, i.e., output probabilities for all input combinations. If we separated output cones and calculated individual output probabilities, the results would scale much further. However, as discussed before, individual output probabilities cannot always be accurately combined to obtain the overall error probability of a circuit. The number of ADD nodes required for the $fidelity$ computation is also listed in Table 3.1, including intermediate computations.

Table 3.1 gives the overall probability of correctness for circuits with gate error probabilities of 0.05 and also for one-way gate errors with probability 0.05. In CMOS gates, an erroneous output value 0 is more likely than an erroneous value 1 because SEUs typically short-circuit power to ground. PTMs can easily encode this bias since error probabilities can be different for different input combinations. Relevant empirical results are given in the "one-way" column of Table 3.1. Circuits with a high output-to-input ratio, such as sdecod, tend to magnify gate errors at fanout stems and, therefore, have higher error probabilities.

PTM computation for $p = 0.05$ requires more memory and longer runtime because less compression is possible. Ideal transfer matrices have large blocks of 0 s, which lend themselves to greater compression. When gate PTMs with error probabilities are composed in various ways, PTMs with a greater number of distinct entries are created, thus yielding less compression. Compare the values in the ITM and PTM shown in Example 2.2. Our results indicate that, while exact and complete

circuit-PTM computation cannot be used to evaluate industry-sized circuits, it can be used to calibrate or validate other reliability evaluation tools.

3.2 Improving Scalability

We have presented ADD algorithms for PTM-based computation, but their scalability appears limited due to the possibility of combinatorial explosion in PTM size. Scalability can be improved in a variety of different ways. In this section, we cover several techniques, starting from methods of speeding up exact PTM computation and moving to heuristic methods that approximate the circuit's error probability.

In Sect. 3.2.1, we propose to improve the efficiency of PTM computation by prescheduling an evaluation order for combining gate PTMs into circuit PTMs. While the evaluation order does not affect the final result, it can decrease the sizes of intermediate matrices. Good evaluation orders compute PTMs for clusters of gates with a small number of inputs and outputs in between the clusters. In effect, such an ordering would enclose fanout branches and reconvergences within the clusters.

In Sect. 3.2.2, we use exact PTM computations for partitions in a circuit (instead of the whole circuit) and propagate signal and error probabilities between the partitions. This method allows exact computations to be maximally used while approximating the overall error rate. In Sect. 3.2.3, we briefly discuss methods of sampling, i.e., computing the average error probability of a randomly generated set of input vectors to estimate the true error probability.

3.2.1 Dynamic Ordering of Evaluation

The ADD-based multiplication algorithm used in our PTM algebra implementation [1] has a major impact on the efficiency of PTM computations. The worst-case time and memory complexity of the multiplication operation is $O((|A||B|)^2)$, for two ADDs A and B. The PTM evaluation algorithm described in Fig. 3.5 first tensors gates for each level to form level PTMs and then multiplies the level PTMs, thereby creating relatively large multiplication instances. Smaller instances can be created by rescheduling the order of evaluation and delaying the tensor product as long as possible.

Example 3.2 Consider the tree of AND gates in Fig. 3.8. Suppose we wish to compute its circuit PTM. The algorithm of Fig. 3.5 requires topologically sorting the gates, calculating the PTM for each level, and multiplying together the levels in order. The levels are $L_4 = \{G15\}$, $L_3 = \{G14, G13\}$, $L_2 = \{G12, G11, G10, G9\}$, $L_1 = \{G8, G7, G6, G5, G4, G3, G2, G1\}$. The corresponding level PTMs have dimensions $2^2 \times 2$, $2^4 \times 2^2$, $2^8 \times 2^4$, and $2^{16} \times 2^8$, respectively. We denote the L_i PTM as M_i. First, we compute $M_3 \times M_4$, which is of dimension $2^4 \times 2$; next, M_2 is multiplied

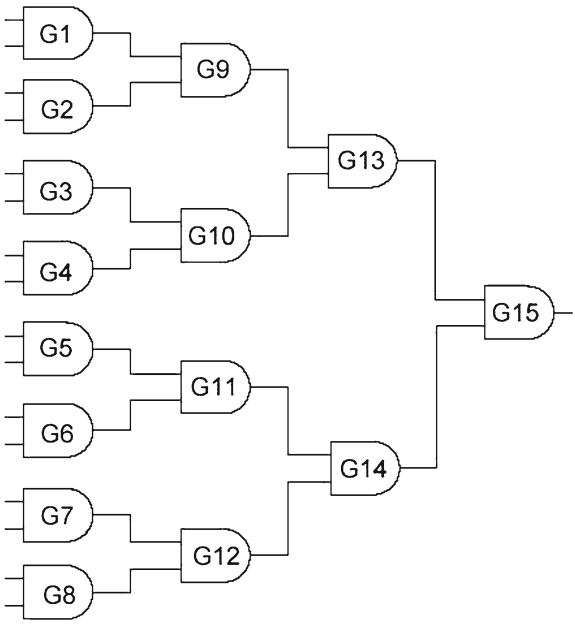


Fig. 3.8 Tree of AND gates used in Example 3.2 to illustrate the effect of evaluation ordering on computational efficiency

by $M_3 \times M_4$, yielding a matrix of size $2^8 \times 2$; and so on. The dimensions of the matrix product instances are as follows: $(2^2 \times 2, 2^4 \times 2^2)$, $(2^4 \times 2, 2^8 \times 2^4)$, $(2^8 \times 2, 2^{16} \times 2^8)$. In the worst case, when ADD sizes are close to matrix sizes (in general, they are smaller, as ADDs provide compression), the total memory complexity of the multiplications is $2^{50} + 2^{34} + 2^{18}$. On the other hand, separating the gates (not tensoring) for as long as possible, starting from the primary inputs, yields the matrix multiplication instances of the following sizes: $4(2^4 \times 2^2, 2^2 \times 2)$, $2(2^4 \times 2, 2^2 \times 2)$, and $(2^{8\times 2, 2^2 \times 2)}$. Here, the total memory complexity is only $2^{20} + 2^{27} + 2^{42}$. Therefore, carefully scheduling matrix multiplication leads to a more efficient PTM computation algorithm.

If the output of a source gate is connected to more than one sink gate, there are two possibilities for evaluation ordering: the first is to tensor gates PTMs and eliminate the redundant variables; the second possibility is to process gates and logic cones separately until they need to be tensored at a different level to facilitate a multiplication. We choose the latter approach, which exchanges multiplications for tensor products. This is advantageous, as the tensor product has lower complexity than multiplication. Determining the optimal order to multiply levels is similar to solving the matrix chain multiplication problem [9], which can be solved by a dynamic programming algorithm in $O(n^3)$ time. Our application can use the same algorithm; the cost of multiplying two matrices is estimated based on their dimensions, without taking ADD compression into account.

The results of applying the improved ordering for multiplication of levels are given in Table 3.2. The data in this table were produced on a Pentium 4 processor

Table 3.2 Runtimes and memory usage for levelized and dynamic evaluation orderings

Circuit	Improved ordering		Levelized ordering	
	Time (s)	Memory (MB)	Time (s)	Memory (MB)
C17	0.212	0.000	1.090	0.004
mux	18.052	2.051	26.314	3.109
z4ml	3.849	1.004	6.594	1.113
x2	11.015	2.344	193.115	12.078
parity	1.060	0.113	1.07	0.133
pcle	28.810	3.309	98.586	6.160
decod	5.132	1.020	30.147	24.969
cu	23.700	2.215	13.385	2.176
pm1	72.384	3.734	77.661	5.031
cc	57.400	4.839	1434.370	155.660
9symml	89.145	6.668	4445.670	552.668
xor5	3.589	0.227	46.721	3.539
b9	9259.680	165.617	23164.900	295.984
c8	35559.500	930.023	mem-out	mem-out

running at 2 GHz. In general, this ordering method uses less memory, with only a modest increase in runtime. The runtime increase seen in Table 3.2 is partially due to the overhead of the dynamic programming. However, this tradeoff is acceptable since memory was the main bottleneck.

3.2.2 Hierarchical Reliability Estimation

In this section, we extend PTM analysis hierarchically to estimate the reliability of larger circuits partitioned into subcircuits. This allows for the use of exact PTM computation for smaller partitions, and provides a way of estimating the error on the entire circuit. The techniques in this section are similar to belief propagation techniques used in Bayesian inference.

First, the ITMs and PTMs of all subcircuits are calculated. Then, in topological order, we calculate the fidelities and output probabilities on each subcircuit output individually. We call the individual fidelity of an output bit its *bit-fidelity*. Since evaluation proceeds in topological order, input *bit-fidelities* are already calculated for the previously processed subcircuits.

In order to formally define bit-fidelity, we introduce the *abstract operation* for notational convenience.

Definition 3.1 For a PTM M and an output variable o_k, $M' = abstract(M, k)$ is the matrix that results from the elimination of all variables *except* o_k from M. Therefore, $M' = eliminate\_variables(M, 0, 1, 2, \ldots, k-1, k+1, \ldots, m)$.

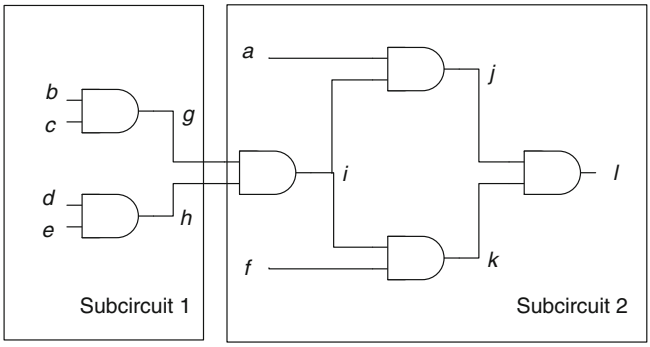


Fig. 3.9 Circuit used in Example 3.3 to illustrate hierarchical reliability estimation

Definition 3.2 The *bit-fidelity* of output o_k of circuit C, with ITM J, PTM M, and input distribution v, is the probability of error of the kth output bit. It is given by $bit\_fidelity(k, v, J, M) = fidelity(v_k, J_k, M_k)$, where $J_k = abstract(J, k)$, $M_k = abstract(M, k)$, and $v_k = abstract(v, k)$

Suppose the input bit-fidelities for the inputs of a particular subcircuit are $p_1, p_2, p_3, \ldots, p_n$. Then, in order to account for input error, the subcircuit PTM is multiplied by $I_{p_1} \otimes I_{p_2}, \ldots, I_{p_n}$, where I_p has the form $\begin{bmatrix} p & 1-p \\ 1-p & p \end{bmatrix}$. The probability distribution of each signal is also calculated by multiplying the input distribution of each subcircuit by its ITM and then abstracting each of the output probabilities. The algorithm details are given in Fig. 3.10, where *SubCircArray* is the topologically sorted array of subcircuits, *PIs* is the list of primary inputs, *POs* is the list of primary outputs, *Distro* stores the separated probability distribution of intermediate variables, and the *Bfid* array contains the bit-fidelities of previously processed signals. At each iteration, *Bfid* is updated with output bit-fidelities of the current subcircuit. At the termination of the algorithm, *Bfid* contains the bit-fidelities of the primary outputs.

This algorithm has several interesting features. First, it only calculates PTMs of subcircuits and thus avoids the state-space explosion associated with directly computing the entire circuit's PTM. For instance, if a circuit with n inputs and m outputs is partitioned into two subcircuits each with $n/2$ inputs and $m/2$ outputs, the PTMs of the two subcircuits together are of size $2(2^{(n+m)/2})$, which is significantly smaller than the circuit PTM, which has size 2^{n+m}. Second, the algorithm approximates joint probability distributions, using marginal probability distributions, and averages local error probabilities at each subcircuit. Any loss of accuracy is a result of the *abstract* operation and the averaging effect which occurs in *bit-fidelity* calculations. Therefore, the estimation technique will be very accurate in cases where there is no reconvergent fanout between the subcircuits. In fact, the error probabilities are exact when each output bit has the same error on all input combinations because, in such cases, averaging does not cause a loss of information. In other cases, the accuracy

Fig. 3.10 The $Bit\_fidelity$ estimation algorithm

```
estimate_bit_fidelity(input V, circuit C)
{
 topological_sort(C)
 for(each primary input in∈C)
  Bfid[in] = 0
  Distro[in] = abstract(V,in)
 partition_circuit(C)
 for(each partition S∈C)
  J = calc_itm(S)
  M = calc_ptm(S)
  for(each input in∈S)
   Vin[S] = tensor_ADD(Vin[S],Distro[in])
   Vout[S] = multiply_ADD(Vin[S],J)
  for(each output out∈S)
   Distro[out] = abstract(Vout[S],j)
  for(each input in∈S)
   I' = tensor_ADD(M',create_I_matrix(Bfid[in]))
  M' = multiply_ADD(I',M)
  for(each output out∈S)
   Bfid[out] = bit_fidelity(out,Distro[out],J,M')
}
```

will depend on the amount of correlation between signals and the variation in signal errors.

Example 3.3 We apply the algorithm of Fig. 3.10 to the circuit in Fig. 3.9. Assume that each of the AND gates in Fig. 3.9 has the following PTM and ITM:

$$\text{AND2}_{0.1} = \begin{bmatrix} 0.9000 & 0.1000 \\ 0.9000 & 0.1000 \\ 0.9000 & 0.1000 \\ 0.1000 & 0.9000 \end{bmatrix} \quad \text{AND2} = \begin{bmatrix} 1 & 0 \\ 1 & 0 \\ 1 & 0 \\ 0 & 1 \end{bmatrix}$$

Suppose that primary inputs are uniformly distributed and have no errors. Initialize $Bfid[a] = Bfid[b] = Bfid[c] = Bfid[d] = BFid[e] = Bfid[f] = 1$ and $Distro[a] = Distro[b] = Distro[c] = Distro[d] = Distro[e] = Distro[f] = [0.5 \;\; 0.5]$. The input vector for subcircuit 1 is given by:

$$vin_1 = [0.0625 \;\; 0.0625 \;\; .0625 \;\; 0.0625 \ldots 0.0625]$$

The PTM and ITM for subcircuit 1 are calculated as follows:

$$\text{ITM1} = \text{AND2} \otimes \text{AND2}$$

$$\text{PTM1} = \text{AND2}_{0.1} \otimes \text{AND2}_{0.1}$$

$$\text{ITM1} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad \text{PTM1} = \begin{bmatrix} 0.81 & 0.09 & 0.09 & 0.01 \\ 0.81 & 0.09 & 0.09 & 0.01 \\ 0.81 & 0.09 & 0.09 & 0.01 \\ 0.09 & 0.81 & 0.01 & 0.09 \\ 0.81 & 0.09 & 0.09 & 0.01 \\ 0.81 & 0.09 & 0.09 & 0.01 \\ 0.81 & 0.09 & 0.09 & 0.01 \\ 0.09 & 0.81 & 0.01 & 0.09 \\ 0.81 & 0.09 & 0.09 & 0.01 \\ 0.81 & 0.09 & 0.09 & 0.01 \\ 0.81 & 0.09 & 0.09 & 0.01 \\ 0.09 & 0.81 & 0.01 & 0.09 \\ 0.09 & 0.01 & 0.81 & 0.09 \\ 0.09 & 0.01 & 0.81 & 0.09 \\ 0.09 & 0.01 & 0.81 & 0.09 \\ 0.01 & 0.09 & 0.09 & 0.81 \end{bmatrix}$$

The fidelity and probability distribution for each output of subcircuit 1 are calculated as follows:

$$vout_1 = vin_1 * \text{ITM1} = [0.5625 \;\; 0.1875 \;\; 0.1875 \;\; 0.0625]$$

$$Distro[g] = abstract(vout_1, g) = [0.75 \;\; 0.25]$$

$$Distro[h] = abstract(vout_1, h) = [0.75 \;\; 0.25]$$

$$\text{PTM1}' = (I(1) \otimes I(1) \otimes I(1) \otimes I) * \text{PTM1} = \text{PTM1}$$

$$Bfid[g] = bit\_fidelity(g, Distro[g], \text{PTM1}', \text{ITM1}) = 0.9$$

$$Bfid[h] = 0.9$$

Similarly for subcircuit 2:

$$\text{ITM2} = (I \otimes \text{AND2} \otimes I)(I \otimes F_2 \otimes I)(\text{AND2} \otimes \text{AND2})(\text{AND2})$$

$$\text{PTM2} = (I \otimes \text{AND2}_{0.1} \otimes I)(I \otimes F_2 \otimes I)(\text{AND2}_{0.1} \otimes \text{AND2}_{0.1})(\text{AND2}_{0.1})$$

$$\text{PTM2}' = (I \otimes I_{0.9} \otimes I_{0.9} \otimes I)(\text{PTM2})$$

$$vin_2 = [0.5 \;\; 0.5] \otimes [0.75 \;\; 0.25] \otimes [0.75 \;\; 0.25] \otimes [0.5 \;\; 0.5]$$

$$\text{ITM2} = \begin{bmatrix} 1\ 0 \\ 1\ 0 \\ 1\ 0 \\ 1\ 0 \\ 1\ 0 \\ 1\ 0 \\ 1\ 0 \\ 1\ 0 \\ 1\ 0 \\ 1\ 0 \\ 1\ 0 \\ 1\ 0 \\ 1\ 0 \\ 1\ 0 \\ 1\ 0 \\ 0\ 1 \end{bmatrix} \quad \text{PTM2} = \begin{bmatrix} 0.8920\ 0.1080 \\ 0.8856\ 0.1144 \\ 0.8920\ 0.1080 \\ 0.8856\ 0.1144 \\ 0.8920\ 0.1080 \\ 0.8856\ 0.1144 \\ 0.8920\ \ 0.108 \\ 0.8344\ 0.1656 \\ 0.8856\ 0.1144 \\ 0.8280\ 0.1720 \\ 0.8856\ 0.1144 \\ 0.8280\ 0.1720 \\ 0.8856\ 0.1144 \\ 0.8280\ 0.1720 \\ 0.8344\ 0.1656 \\ 0.3160\ 0.6840 \end{bmatrix} \quad \text{PTM2}' = \begin{bmatrix} 0.8920\ 0.1080 \\ 0.8851\ 0.1149 \\ 0.8920\ 0.1080 \\ 0.8810\ 0.1190 \\ 0.8920\ 0.1080 \\ 0.8810\ 0.1190 \\ 0.8920\ 0.1080 \\ 0.8441\ 0.1559 \\ 0.8851\ 0.1149 \\ 0.8229\ 0.1771 \\ 0.8810\ 0.1190 \\ 0.7819\ 0.2181 \\ 0.8810\ 0.1190 \\ 0.7819\ 0.2181 \\ 0.8441\ 0.1559 \\ 0.4133\ 0.5867 \end{bmatrix}$$

$$vout_2 = [0.9922\ \ 0.0078]$$

$$Distro[l] = [0.9922\ \ 0.0078]$$

$$BFid[l] = bit\_fidelity(l, Distro[l], \text{PTM2}', \text{ITM2}) = 0.869$$

Alternatively, using the circuit PTM to calculate the fidelity gives $fidelity = 0.862$. This has an error of only 0.003 for gate errors in the range 0.1.

The *fidelity* of the entire circuit (rather than just its output bits) can be further estimated by using the binomial probability distribution to calculate the probability that any output signal has an error. This once again assumes that output signals are independent.

3.2.3 Approximation by Sampling

Reliability estimation requires computing the circuit error associated with each input combination. Like SER analysis, reliability analysis can also be approximated by sampling input vectors. This sampling can be done in several ways. We briefly discuss two methods of sampling which yield estimates of the overall circuit error probability.

The first method is to sample input and output vectors without computing circuit or component PTMs. This method is akin to replacing the TSA/TMSA fault models of Chap. 5 with a more general PTM-based fault model (where the error probabilities depend on the input values) and then using the same methods of bit-parallel sampling. This provides a link between signature-based approximate analysis and PTM-based exact analysis. The algorithm from Fig. 5.6 can be used to compute the error proba-

bility, with a slight modification. Instead of flipping bits of signatures with constant probabilities at each gate, we can flip signatures with probabilities conditioned on the input values (as indicated by the appropriate row of the gate PTM).

A second method of sampling involves computing the exact output distribution for each input vector, but generating the set of input vectors to sample randomly. Computing the exact output vector for an input vector is fairly complicated in and of itself. However, it does scale to much larger circuits than circuit PTM computation. The algorithm for computing the output vector for a given input vector using vector-PTM multiplication and tensoring is described in Chap. 4.

References

1. Bahar RI et al (1997) Algebraic decision diagrams and their applications. J Formal Methods Syst Des 10(2/3):171–206
2. Viamontes GF, Markov IL, Hayes JP (2003) Improving gate-level simulation of quantum circuits. Quantum Inf Process 2(5):347–380
3. Krishnaswamy S, Viamontes GF, Markov IL, Hayes JP (2005) Accurate reliability evaluation and enhancement via probabilistic transfer matrices. In: Proceedings of DATE, pp 282–287
4. Krishnaswamy S, Viamontes GF, Morkov IL, Hayes JP (2008) Probabilistic transfer matrices in symbolic reliability analysis of logic circuits. ACM Trans Des Autom Electron Syst 13(1, article 8): 35
5. Bryant RE, Chen YA (1995) Verification of arithmetic functions with binary moment diagrams. In: Proceedings of 32nd ACM/IEEE design automation conference(DAC), pp 535–541
6. Hachtel G, Somenzi F (1996) Logic synthesis and verification algorithms. Kluwer Academic Publishers, Boston
7. Somenzi F (2004) CUDD: Colorado University Decision Diagram Package. http://vlsi.colorado.edu/fabio/CUDD. Accessed 31 May 2004
8. Clarke E et al (1996) Multi-terminal binary decision diagrams and hybrid decision diagrams. In: Sasao T, Fujita M (eds) Representations of discrete functions. Kluwer Academic Publishers, Norwell pp 93–108
9. Cormen T, Lieserson C, Rivest R, Stein C (2001) Introduction to algorithms, 2nd edn. MIT Press, Cambridge

Chapter 4
Testing Logic Circuits for Probabilistic Faults

After integrated circuits are manufactured, it is important to ensure that they do not have unusually high sensitivity to soft errors. While the SER of a circuit can be analyzed during or after design, the error rate can be significantly increased because of variability in the manufacturing process. Therefore, circuits have to be tested in order to ensure that their soft error rates do not exceed an acceptable threshold. To estimate the expected soft error rate in the field, chips are typically tested while being exposed to intense beams of protons or neutrons, and the resulting error rate is measured. However, these types of tests often take a long time to conduct because random patterns may not be sensitive to vulnerabilities in the circuit. In this chapter, we develop methods of selecting test vectors to minimize test application time.

Generating tests for probabilistic faults is fundamentally different from established testing techniques. Traditionally, the goal of testing has been to detect the presence of defects. A set of test vectors is normally applied to the inputs of a circuit and the resultant outputs are compared to correct pre-computed outputs to determine whether a fault is present. In contrast, the goal of probabilistic testing is the estimation of error probability. Probabilistic testing requires a *multiset* (a set with repetitions) of test patterns, since a given fault is only present for a fraction of the computational cycles. Another difference is that some test vectors detect transient faults with higher probability than others due to path-dependent effects like electrical masking. Therefore, one can consider the likelihood of detection, or the *sensitivity*, of a test vector to a fault. Table 4.1 summarizes these differences.

In Sect. 4.1, we define and provide computation methods for determining test-vector sensitivity to faults. Section 4.2 provides integer linear programming (ILP) formulations for generating a compact set of test vectors for probabilistic faults. Most of the concepts and results in this chapter also appear in [1, 2].

S. Krishnaswamy et al., *Design, Analysis and Test of Logic Circuits Under Uncertainty*, Lecture Notes in Electrical Engineering 115, DOI: 10.1007/978-90-481-9644-9_4,

Table 4.1 Key differences between deterministic and probabilistic testing

Attribute	Deterministic testing	Probabilistic testing
Fault type	Deterministic	Transient or intermittent
Fault model	Stuck-at, bridging, etc.	Probabilistic generalization
Test inputs	Set of input vectors	Multiset of input vectors
Coverage	Faults detected with certainty	Faults detected with varying probabilities
Goal	Detect fault presence	Estimate fault probability

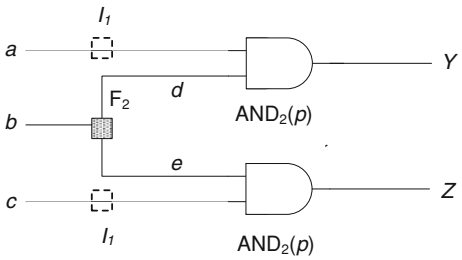


Fig. 4.1 Circuit to illustrate test-vector sensitivity computation

4.1 Test-Vector Sensitivity

In this section, we discuss the sensitivity of test vectors to transient faults. We begin with an example and then present a PTM-based algorithm for test-vector sensitivity computation.

Example 4.1 Consider the circuit in Fig. 4.1. If an SEU occurs with a certain probability at input b, then the test vectors that propagate the induced error to the outputs are: $t_Y = 100$ (to output Y), $t_Z = 001$ (to output Z), and $t_{YZ} = 101$ (to both Y and Z).

In deterministic testing, any test vector that detects a fault can be chosen. However, error attenuation along sensitized paths affects the propagation of probabilistic faults. If the propagation probability is p_{prop} at each gate, then t_1 has probability $p_{t_1} = p_{\text{prop}}$, t_2 has probability $p_{t_2} = p_{\text{prop}}$, and t_3 has probability $p_{t_3} = 2p_{\text{prop}} - p_{\text{prop}}^2$. For a fault that occurs with probability p_f, a vector t_i has to be repeated $\lceil 1/(p_{t_i} * p_f) \rceil$ times for one expected detection. Therefore, test application time will be shortest for test vector t_1. Vector t_3 is said to be the most sensitive to the transient fault in question.

Since PTMs can encode a wide variety of faults, including faults with path-based effects, test-vector sensitivity computed with PTMs can take these effects into account. There are two ways to compute the sensitivity of a test vector. The first is by circuit-PTM computation, which is explained in Chap. 2. Here, the gate PTMs are combined to form a circuit PTM, and the most sensitive test vectors correspond to the PTM rows with the lowest probability of correctness.

We can formally define the sensitivity of a test vector to faults in the circuit using PTMs. The *sensitivity* of a test vector t to a multi-fault set $F = \{f_1, f_2, \ldots, f_n\}$

Fig. 4.2 Sensitivity computation on the circuit of Fig. 4.1

$$v_t \times M_F = v_t \times \begin{bmatrix} 0.81 & 0.09 & 0.09 & 0.01 \\ 0.81 & 0.09 & 0.09 & 0.01 \\ 0.81 & 0.09 & 0.09 & 0.01 \\ 0.09 & 0.81 & 0.01 & 0.09 \\ 0.81 & 0.09 & 0.09 & 0.01 \\ 0.81 & 0.09 & 0.09 & 0.01 \\ 0.09 & 0.01 & 0.81 & 0.09 \\ 0.01 & 0.09 & 0.09 & 0.81 \end{bmatrix} = \begin{bmatrix} 0.81 \\ 0.0900 \\ 0.09 \\ 0.01 \end{bmatrix}^T$$

$$v_t \times M = v_t \times \begin{bmatrix} 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \end{bmatrix}$$

$$sens(F,t) = 1 - ||[1\ 0\ 0\ 0].*[0.81\ 0.09\ 0.09\ 0.01]||$$

$$= 1 - ||[0.81\ 0\ 0\ 0]|| = 1 - (.81) = 0.9$$

which occurs with probability $P = \{p_1, p_2, \ldots, p_n\}$ in circuit C with PTM M_F and ITM M is defined as the total probability that the output under t is erroneous, given that faults F exist with probability P. A test vector t can be represented by the vector v_t, with 0's in all but the index corresponding to t's input assignments. For instance, if a test vector t assigns 0's to all input signals and C has 3 inputs, then $v_t = [1\ 0\ 0\ 0\ 0\ 0\ 0\ 0]$. The sensitivity of t is the probability that the ideal and faulty outputs are different, and this is computed by taking the norm of the element-wise product (Definition 2.4) of the correct and faulty output vectors. This operation is similar to the *fidelity* operation of Chap. 2, defined for vectors rather than matrices.

$$\text{sens}(F, t) = 1 - ||(v_t M_f) . * (v_t M)||_{l_1} \tag{4.1}$$

The sensitivity of test vector

$$v_t = [1\ 0\ 0\ 0\ 0\ 0\ 0\ 0]$$

for the circuit in Fig. 4.1, with the probability of error $p = 0.1$, can be computed from the circuit's ITM M, and PTM M_F, as shown in Fig. 4.2. Here, both the correct and faulty output vectors are computed and the results are compared to obtain the sensitivity.

Note that $v_t M_f$ and $v_t M$ need to be marginalized if there is a multi-bit signal representation. For instance, in the case of the SEU model of Chap. 2, the second bit needs to be summed out for both of the vectors to obtain the correct sensitivity.

The second method of sensitivity computation is through *output vector* computation for a particular test vector. Here, we begin with a pre-selected complete set of

test vectors for the permanent stuck-at faults corresponding to those in F. For each test vector in this set, we compute the faulty output at each gate using vector-PTM multiplication through intermediate gates. We also compute the ideal output at each gate. The ideal output is $v_t M$, and the faulty output vector is $v_t M_f$. The advantage of this method is that we do not have to explicitly compute the circuit PTM and ITM, processes which are computationally expensive. We then use Eq. 4.1 to compute the sensitivity.

A caveat in output vector computation is that fanout branches result in inseparable probability distribution of the branch signals. If these signals are marginalized or treated as separate, then inaccuracies can occur in the output probabilities. A simple method of handling this problem is to jointly store the probabilities of these signals and then enlarge any gate PTM the signals encounter. We accomplish gate enlarging by adding inputs to the gate that pass through unchanged, i.e., tensoring the gate matrix with an identity matrix I. The example below shows the processing, in topological order, of input vectors through the circuit to obtain intermediate and output vectors. At each step, we compute the appropriate joint input probability distribution for the next gate in topological order. However, due to inseparable signal distributions, gates often have to be enlarged with identities.

Example 4.2 Consider the circuit in Fig. 4.1. Suppose the primary input vectors are v_a, v_b, v_c, and the 2-input AND gates have PTM $\text{AND}_2(p)$. The faulty output vector is obtained as follows.

$$
\begin{aligned}
&v_{d,e} = v_b \times F_2 \\
&\text{enlarged AND}_2(p) = (\text{AND}_2(p) \otimes I_1) \\
&v_{a,d,e} = (v_a \otimes v_{d,e}) \\
&v_{Y,e} = v_{a,d,e} \times \text{enlarged AND}_2(p) \\
&v_{d,e,c} = (v_{d,e} \otimes v_c) \\
&v_{d,Z} = v_{d,e,c} \times \text{enlarged AND}_2(p)
\end{aligned}
$$

Additionally, note that signal probabilities incorporate the effects of fanouts. However, storing joint probability distributions can be computationally expensive; therefore, in some cases a loss of accuracy may be traded in favor of memory complexity reduction.

An algorithm for computing the output of a test vector under probabilistic faults (encoded in gate PTMs) is shown in Fig. 4.3. The primary input values, determined by the given test vectors, are converted into input vectors. Then, in topological order, the inputs for each gate are tensored together to form the input vector for the gate. If any of the input signals are stored jointly with other signals, the gate in question is enlarged by the number of additional signals. The gate PTM is multiplied by the input vector to obtain the output vector. In the case of a multiple-output gate such as a fanout gate, the output vector stays as a joint probability distribution. In practice, output distributions can become very large, through the accumulation of correlated signals. However, the

Table 4.2 Runtime and memory usage for sensitivity computation for benchmark circuits; faulty gates have error probability 0.05 for all inputs

Circuit	Characteristics			Ideal circuits		Erroneous circuits	
	Inputs	Outputs	Gates	Time (s)	Memory (MB)	Time (s)	Memory (MB)
C432	36	7	160	0.28	0.7	0.73	0.8
C499	41	32	202	0.30	0.2	0.36	1.2
C880	60	26	383	0.47	0.4	52.50	124.0
C1355	41	32	546	1.44	0.1	0.22	0.6
C1908	33	25	880	0.76	1.1	11.70	42.2
C3540	50	22	1669	1.48	2.2	131.50	547.1
C6288	32	32	2416	2.12	3.3	50.90	44.8

```
compute_faulty_output(Circuit C, testvector T)
{
 for(all inputs i∈C)
  vector[i] = create_row_vector(T[i])
 insert_fanout_gates(C)
 sort_topological(C)
 for(each node g∈C)
  for(each input j∈inputs(g))
   inputvector[g] = inputvector[g] ⊗ PTM[j]
   enlarge(g,size(j)−1)
  outputvector[g] = inputvector[g] × PTM[g]
  for(each outputs o∈outputs(g))
   vector[o] = outputvector[g]
}
```

Fig. 4.3 Algorithm for output-vector computation

joint signals can be separated by using the *eliminate_variables* operation, which may entail some loss of accuracy.

This process can be repeated with gate ITMs (or functional simulation) to obtain the ideal output vector. Finally, test vector sensitivity is computed according to Eq. 4.1, using the *fidelity* algorithm of Fig. 3.7 applied to the ideal and faulty primary-output vectors. Table 4.2 shows average runtime and memory usage statistics for test vector sensitivity computation for various input vectors on standard benchmark circuits from the ISCAS-85 suite. These simulations were conducted on a Linux workstation with an Intel Xeon CPU 2.0 GHz processor and cache size 512 KB. Here, the faulty circuit consisted of gates with output bit-flip probabilities of 0.05.

4.2 Test Generation

Next, we use the test-vector sensitivity information computed in the previous section to generate compact multisets of test vectors for transient-fault detection. Test-set compaction is closely related to the standard SET COVER problem [3]. The goal of

SET COVER is to find a minimal set of subsets $\{t_1, t_2, \ldots, t_n\}$ of a given set S such that every member of S belongs to at least one of the chosen subsets. In the context of test generation, S is the set of all possible faults and each test vector t_i represents a subset of faults, namely the subset of faults that it detects. When testing for transient faults (soft errors), tests may have to be repeated to increase the probability of fault detection, therefore multisets of tests are selected.

This connection between SET COVER and test compaction allows us to modify algorithms designed for SET COVER and introduce related ILP formulations whose LP relaxations can be solved in polynomial time. Modifying the test-multiset objective simply amounts to altering the ILP objective function. We first describe the simgle-fault case and then extend our arguments to incorporate two multiple-fault assumptions often used in the literature. Then, we give algorithms for multiset test generation that achieve high probabilities of fault detection and resolution.

Suppose a single fault f in a circuit C has an estimated probability p of occurrence. We confirm its probability as follows:

1. Derive a test vector t with high sensitivity $\text{sens}(f, t)$.
2. Apply t to C $k = \lfloor 1/\text{sens}(f, t) \rfloor$ times for one expected detection.
3. If we have $d(f) \gg 1$ detections, we can conclude that the actual probability of f is higher and reject the estimated probability. We can estimate the probability that there are $d(f)$ detections in k trials using the binomial theorem. If the probability of $d(f)$ detections is low, then it is likely that the actual sensitivity $\text{sens}(f, t)$ is higher than the estimate.
4. If $\text{sens}(f, t)$ exceeds estimates, we update the estimates and repeat this process.

To extend the above method to multiple faults, we distinguish two cases:

- Assumption 1: There are several probabilistic faults, yet the circuit experiences only a single fault in any given clock cycle. This is the standard single-fault assumption, which is justified by the rarity of particle strikes.
- Assumption 2: Each circuit component (gate) has an independent fault probability, i.e., multiple faults at different locations can occur in the same clock cycle. This assumption is applicable to nanotechnologies where random device behavior can lead to multiple faults in different locations of the circuit. Here, the probability of two faults is given by the product of individual fault probabilities.

Our goal in either case is to pick a multiset of vectors T' taken from $T = \{t_1, t_2, \ldots t_m\}$ such that $|T'|$ is minimal. Recall that each test vector t_i represents a subset of F, i.e., each test vector detects a subset of faults. Under Assumption 1, we minimize the size of the multiset by using test vectors that are either especially sensitive to one fault or somewhat sensitive to many faults. Therefore, to obtain a detection probability of p_{th} we need n tests, where n satisfies $(1 - p)^n \leq 1 - p_{\text{th}}$. Fig. 4.4 gives a greedy algorithm for generating such a multiset of test vectors, starting from a compacted set of test vectors.

Intuitively, compacted test sets are likely to contain many sensitive test vectors since each test vector must detect multiple faults. However, better results can be obtained if we start with a larger set of test vectors, such as the union of different

```
select_test_multiset(faults F, tests T, prob p_th)
{
 UF = F
 while(!isempty(UF))
  Tmax = find_maximal_test(UF, T, p_th)
  add_selected_test(ST, Tmax)
  UF = remove_new_covered(UF, ST, p_th)
 return ST
}

remove_new_covered(faults UF, test ST, prob p_th)
{
 for(each fault f ∈ UF)
  for(each test t ∈ ST)
  Pdet+ = ∏_i(1 − sens(f,t))
  if(1 − Pdet = p_th)
   remove_fault(UF, f)
 return UF
}
```

Fig. 4.4 Greedy minimization of the number of test vectors (with repetition)

compact test sets. The set UF used in the algorithm stores the uncovered faults in any iteration, i.e., faults not detected with a probability of p_{th}. As before, T is the set of tests $\{t_1, t_2 \ldots t_n\}$, and F is the set of faults.

Kleinberg and Tardos [3] prove that a similar algorithm for SET COVER produces covers that have size within $O(\log(|testmultiset|))$ of the minimum size. Note that the runtime is lower bounded by the size of the multiset, as this is the number of iterations through the **while** loop.

Our ILP formulation for generating a minimal test multiset is shown in Fig. 4.5a. The challenge in adapting the ILP solution algorithms for SET COVER or SET MULTICOVER is that there is no notion of a probabilistic cover. In our case, each test detects each fault with a different probability $\text{sens}(f_j, t_i)$. If we were to require a minimum detection probability p_{th}, as in Fig. 4.4, the constraint that for all f_j, $\prod_j(1 - \text{sens}(f_i, t_j)) < 1 - p_{\text{th}}$ would not be linear. We therefore alter this constraint and linearize it by observing that the number of repetitions of each test t_i is an independent identically distributed binomial random variable for each fault f_j. Therefore, if a test is repeated x_i times, the expected number of detections for a fault f_j is $x_i \times \text{sens}(f_j, t_i)$, i.e., the expected value of a binomial random variable with parameters $(x_i, \text{sens}(f_j, t_i))$. Since the expectation is linear, we can add the contributions of all test vectors for each fault f_j as $\sum_i(x_i \times \text{sens}(f_j, t_i))$, leading to the constraint in Line 3 of Fig. 4.5a. It can be shown that this ILP formulation reduces to MULTISET- MULTICOVER, a variant of the set-cover problem previously discussed. The LP-relaxation, along with randomized rounding, gives a solution of this problem, which is within a log factor of optimal [4]. In randomized rounding, each x_i is rounded up, with a probability equal to the fractional part of x_i.

(a)

$$\begin{array}{c}\texttt{Minimize } \sum_{i=0}^{m} x_i \\ \texttt{subject to:} \\ \forall j, (\sum_{i=1}^{0} x_i \times sens(f_j, t_i)) \geq n \\ \forall i, x_i \geq 0, \ x_i \texttt{ is an integer}\end{array}$$

(b)

$$\begin{array}{c}\texttt{Minimize } \sum_{j=0}^{n} \sum_{i=0}^{m} x_i \times sens(f_j, t_i) \\ \texttt{subject to:} \\ \forall j, (\sum_{i=1}^{m} x_i \times sens(f_j, t_i)) \geq n \\ \forall i, x_i \geq 0, \ x_i \texttt{ is an integer}\end{array}$$

Fig. 4.5 ILP formulations for test-set generation with a fixed number of expected detections: **a** to minimize the number of test vectors, and **b** to maximize fault resolution (minimize overlap)

Table 4.3 Number of repetitions required for random vectors versus maximally sensitive test vectors

Circuit	Random vectors		Best vectors		% improvement
	Average sensitivity	Number repetitions	Maximum sensitivity	Number repetitions	
9symml	3.99E−5	2.51E+4	7.99E−5	1.25E+4	50.0
alu4	5.78E−4	1.73E+03	1.82E−3	549	68.2
i1	6.65E−5	1.50E+04	9.99E−5	1.00E+04	33.4
b9	7.70E−5	1.30E+04	1.10E−4	9.09E+03	30.0
C880	5.38E−4	1.86E+03	9.39E−4	1.07E+03	42.7
C1355	1.03E−3	970	1.27E−2	78	91.8
C499	2.76E−4	3.62E+03	1.27E−3	787	78.27
x2	3.39E−5	2.95E+04	4.99E−5	2.00E+04	32.1
Average					53.3

Assumption 2 generalizes the single-fault case described above. We can treat the set of faults as a single fault with multiple locations and introduce fault probabilities into all gate PTMs simultaneously; we denote this fault by F'. Then, we can simply pick the test vector t that is most sensitive to the combination of simultaneous faults, using methods from the previous section. We can repeat t a total of $k/\text{sens}(F', t)$ times for k expected detections. In Table 4.3, we consider a situation in which each gate in the circuit has a small probability of error $p = 10^{-5}$. The most sensitive test vector requires 53.3 % fewer repetitions than a random vector, on average. This implies a proportional decrease in test-application time.

We can additionally diagnose the probabilistic faults under Assumption 1. In other words, we can select test vectors such that ambiguity about which fault is detected is minimized. For this purpose, we modify the objective to that of Fig. 4.5b. Intuitively, once the required detection probability is achieved, we minimize the total number of extra detections. This is equivalent to minimizing the overlap in the subsets represented by the test vectors. In contrast to the previous formulation, this problem is related to MULTISET EXACT MULTICOVER, and the approximation is also within a log factor of optimal.

In practice, the number of test vectors needed is often quite small because testers are likely to be primarily concerned with the faults that occur the most. The number of repetitions of a test vector for n expected detections is n/p_f, where p_f is the fault probability. Therefore, the multiset decreases with the expected fault probability.

Table 4.4 Number of test vectors required to detect input signal faults with various threshold probabilities p_{th}

Circuit	$p_{\text{th}} = 0.05$		$p_{\text{th}} = 0.75$		$p_{\text{th}} = 0.85$		$p_{\text{th}} = 0.95$		$p_{\text{th}} = 0.99$	
	Rand	Our	Rand	Our	Rand	Our	Rand	Our	Rand	Our
c6288	377	56	782	112	1034	148	1266	236	1998	360
c432	731	462	1415	924	1771	1221	2696	1947	3797	2970
c499	1643	518	2723	1036	3085	1369	4448	2183	8157	3330
c3540	907	411	1665	817	2256	1078	3589	1716	4975	2615
c5315	2669	854	4691	1708	6531	2557	8961	3599	13359	5490
c7552	3729	1680	6824	3364	8352	4445	12210	7082	18314	10805
c2670	3650	884	5699	1770	7755	2339	11104	3729	15961	5682
% improvement		64.5		59.7		57.26		53.71		53.05

Rand is the average number of test vectors selected during random test generation

Additionally, if application time is limited, we can select test vectors to maximize the expected detection rate. Here, we use a binary search for the largest value of n which can be achieved with m test vectors. Since the program in Fig. 4.5a attempts to minimize the number of test sets selected, it also maximizes the number of faults covered by each test.

In summary, test generation for probabilistic faults requires the following steps:

- Generate a set of tests T for the corresponding deterministic faults in F.
- Evaluate the sensitivity of each test in T with respect to each fault in F.
- Execute the greedy algorithm in Fig. 4.4 or solve the ILP shown in Fig. 4.5.

Table 4.4 shows the number of test vectors required to detect probabilistic stuck-at faults using the method of Fig. 4.4, and assuming probability $p_f = 0.05$. These results show that our algorithm requires 53–64 % fewer test vectors than random selection, even with a small complete test vector set (generated by ATALANTA) used as a base set.

Once a multiset of test vectors is generated, the actual probability of error can be estimated using Bayesian learning. This well-established AI technique uses observation (data) and prior domain knowledge to predict future events [5]. In our case, the prior domain knowledge is the expected or modeled fault probabilities in a circuit, and the data come from testing.

References

1. Krishnaswamy S, Markov IL, Hayes JP (2005) Testing logic circuits for transient faults. In: Proceedings of ETS, pp 102–107
2. Krishnaswamy S, Markov IL, Hayes JP (2007) Tracking uncertainty with probabilistic logic circuit testing. IEEE Des. Test 24(4):312–321
3. Kleinberg J, Tardos E (2005) Algorithm design. Addison-Wesley, Boston
4. Vazirani VV (2001) Approximation algorithms. Springer, New York
5. DeGroot M (1970) Optimal statistical decisions. McGraw-Hill, New York

Chapter 5
Signature-Based Reliability Analysis

Soft errors are increasingly common in logic circuits, making soft-error rate (SER) prediction important in all phases of design. As discussed in Sect. 1.2, the SER depends not only on noise effects, but also on the logical, electrical, and timing-masking characteristics of the circuit. Each of these types of masking can be predicted with a fair amount of accuracy after certain phases of the design process—logic masking after logic design, electrical masking after technology mapping, and timing masking after physical design—and generally stays in effect through the rest of the design flow. However, reliability analysis should not become a bottleneck in the design process due to modern compressed design schedules. Therefore, it is important to efficiently and accurately analyze the SER during the actual design process.

This chapter describes an SER analysis tool called AnSER. It employs a functional simulation technique based on bit-parallel signal representations called signatures. These provide an efficient way of computing testability measures like signal probability and observability, which are closely connected to the probability of error propagation. More specifically, the probability of logic-fault propagation is the same as the testability of the fault. The testability of a fault is the likelihood that a random input vector sensitizes a fault. In other words, it is a measure of how easy it is to test the fault. Enumerating test vectors for a particular fault is known to be a problem with ♯P-hard complexity. In other words, it has the same complexity as counting the number of solutions to a SAT instance. Since exact analysis is impractical for all but the smallest of circuits, we estimate testability using a new, high-performance signature-based algorithm.

The remainder of this chapter is organized as follows. Section 5.1 develops our method for computing the SER of logic circuits by accounting for logic masking. Section 5.2 extends this methodology to sequential circuits. Finally, Sect. 5.3 incorporates timing and electrical masking into SER estimates. Most of the techniques and results presented in this chapter also appear in [1–4].

S. Krishnaswamy et al., *Design, Analysis and Test of Logic Circuits Under Uncertainty*, Lecture Notes in Electrical Engineering 115, DOI: 10.1007/978-90-481-9644-9_5,

5.1 SER in Combinational Logic

This section presents an SER analysis method for combinational logic. We first develop fault models for soft errors, and provide background on functional-simulation signatures, which are used extensively in AnSER. We then derive SER algorithms under both single- and multiple-fault assumptions using signal probability and observability measures that are computed from signatures. Finally, we show how to account for electrical and timing masking.

5.1.1 Fault Models for Soft Errors

In earlier chapters, we encoded all probabilistic behavior in logic gates, including probabilistic faults, as a probabilistic transfer matrix (PTM). In this section, we restrict our analysis to particular types of faults that model single-event upsets as probabilistic flips of logic values in circuits. Such simpler fault models enable more efficient computation of soft-error rates for the purposes of evaluation during design.

In general, fault models are abstract, logic-level representations of defects and are usually employed in automatic test-pattern generation (ATPG) algorithms. Fault models for soft errors, as we showed in Chap. 4, can be useful for testing. However, here their primary use is in SER analysis; the close connections between testability and SER facilitate this use.

Recall that we model external noise (such as an SEU) by transient faults. The main difference between a permanent fault and a transient fault is its persistence, which we model as a probability of error per clock cycle. Each circuit node g can potentially experience a transient single stuck-at-1 (TSA-1) fault with probability $Perr_1(g)$, and a transient single stuck-at-0 (TSA-0) fault with probability $Perr_0(g)$.

Definition 5.1 A *transient stuck-at fault (TSA)* is a triple, $(g, v, Perr(g))$ where g is a node in the circuit, $v \in \{0, 1\}$ indicates a stuck-at value, and $Perr(g)$ is the probability of a stuck-at fault when the node has correct value v.

The advantage of basing a fault model on the stuck-at model is that test vectors for TSA faults can be derived in the same way as for SA faults. Therefore, the same ATPG tools can be used for TSA faults as well. The TSA fault model, in particular, assumes that at most one fault will occur in any clock cycle. This assumption is common in much of SER research because for most technologies, the intrinsic error rate (due to neutron flux, for instance) is fairly low. Using the single-error assumption, SER can be computed as the sum of gate/component contributions. The contribution of each gate to the SER depends on the SEU rate of the particular gate, as captured by $Perr(g)$, and on the observability of the error.

In the case of multiple faults, we have to consider the possible sets of gates that experience faults in the same cycle and the possibility that these faults interfere with

each other. The TSA model can be extended to two types of multiple faults called transient multiple correlated stuck-at faults, and transient multiple stuck-at faults.

Faults, considered as random events, can be *independent* or *correlated*. Multiple-bit upsets, where a single particle strike causes multiple upsets in nearby gates, are an example of spatially correlated faults.

Definition 5.2 A *transient multiple-correlated stuck-at fault* (TMCSA) is the triple $(G, V, Perr)$ where G is a set of nodes $\{g_1, g_2, \ldots g_n\}$, V is a set of binary values $\{v_1, v_2, v_3 \ldots v_n\}$ that correspond to the stuck-at values of nodes in G, and $Perr$ is the joint-fault probability of nodes in G.

Transient multiple stuck-at faults apply to circuits with independent probabilities of gate or node failure.

Definition 5.3 A *transient multiple stuck-at fault (TMSA)* is represented by (G, V, P) where G is a set of nodes $\{g_1, g_2, \ldots g_n\}$, V is the set of corresponding binary stuck-at values $\{v_1, v_2, \ldots v_n\}$ and P is the corresponding vector of independent error probabilities $\{p_1, p_2, \ldots p_n\}$.

Unlike TSA and TMCSA faults, a circuit may contain only one TMSA fault of interest—the fault with G containing all the nodes in the circuit. TMSA faults may be used to model independent device failure probabilities rather than SEU effects.

In the next two sections, we utilize the TSA fault model to compute the SER of logic circuits. It is sometimes convenient to measure the SER in terms of the probability of error per cycle. The results can easily be converted into units of FIT, or failures per 10^9 s. If the soft-error probability per cycle is p, then the expected number of failures per 10^9 s is simply $p \times \text{freq} \times 10^9$, where freq is the clock frequency. Assuming only one error occurs in each cycle, $Perr_0(g)$ is the probability that only gate g experiences an error. Therefore, gate SER in units of FITs can also be used in a similar fashion.

5.1.2 Signatures and Observability Don't-Cares

In this work, we systematically use functional-simulation signatures for three purposes: (1) to compute the SER, (2) to identify error-sensitive areas of a circuit, and (3) to identify redundant nodes for resynthesis. A circuit node g can be labeled by a signature as defined below.

Definition 5.4 A *signature* $sig(g) = F_g(X_1)F_g(X_2)\ldots F_g(X_K)$ is the sequence of logic values observed at circuit node g in response to applying a sequence of K input vectors $X_1, X_2, \ldots, X_K$ to the circuit.

Here, $F_g(X_i) \in \{0, 1\}$ indicates the value appearing at g in response to X_i. The signature $sig(g)$ thus partially specifies the Boolean function F_g realized by g. Applying all possible input vectors (exhaustive simulation) generates a signature

```
compute_sigs(Circuit C, size K)
{
 for(all inputs i∈C)
  sig(i) = gen_random_sig(K)
 sort_topological(C)
 for(all nodes g∈C)
  sig(g) = Op<g>(inputsigs(g))
}
```

Fig. 5.1 Basic algorithm for computing functional-simulation signatures

that corresponds to a full truth table. In general, $sig(g)$ can be seen as a kind of "supersignal" appearing on g. It is composed of individual binary signals that are defined by some current set of vectors. Like the individual signals, $sig(g)$ can be processed by EDA tools such as simulators and synthesizers as a single entity. It can be propagated through a sequence of logic gates and combined with other signatures via Boolean operations. This processing can take advantage of bitwise operations available in CPUs to speed up the overall computation compared to processing the signals that compose $sig(g)$ one at a time.

Signatures with thousands of bits can be useful in pruning non-equivalent nodes during equivalence checking [5, 6]. A related speedup technique is also the basis for "parallel" fault simulation [7]. The basic algorithm for computing signatures is shown for reference in Fig. 5.1. Here, $Op\langle g\rangle$ refers to the logic operation performed by gate g. This operation is applied to the signatures of the input nodes of gate g, denoted $inputsigs(g)$.

Figure 5.2 shows a 5-input circuit where each of the 10 nodes is labeled by an 8-bit signature computed with eight input vectors. These vectors are randomly generated, and conventional functional simulation propagates signatures to the internal and output nodes. In a typical implementation such as ours, signatures are stored as logical words and manipulated with 64-bit logical operations, ensuring high simulation throughput. Therefore, 64 vectors are simulated in parallel, with each signature processed. Generating K-bit signatures in an N-node circuit takes $O(NK)$ time.

The observability don't cares (ODCs) for a node g are the input vectors which produce a logic value at g that does not affect the primary outputs. For example, in the circuit AND$(a,$ OR$(a, b))$, the output of the OR gate is inconsequential when $a = 0$. Hence, input vectors 00 and 01 are ODCs for b.

Definition 5.5 Corresponding to the K-bit signature $sig(g)$, the *ODC mask* of g, denoted $ODCmask(g)$, is the K-bit sequence whose ith bit is 0 if input vector X_i is in the don't-care set of g; otherwise the ith bit is 1, i.e., $ODCmask(g) = X_1 \notin ODC(F_g)X_2 \notin ODC(F_g) \ldots X_K \notin ODC(F_g)$.

The ODC mask is computed by bitwise inverting $sig(g)$ and resimulating through the fan-out cone of g to check if the changes are propagated to any of the primary outputs. This algorithm is shown as *compute_odc_exact* in Fig. 5.3a and has complexity $O(N^2)$ for a circuit with N gates. It can be sped up by recomputing signatures only as long as changes propagate.

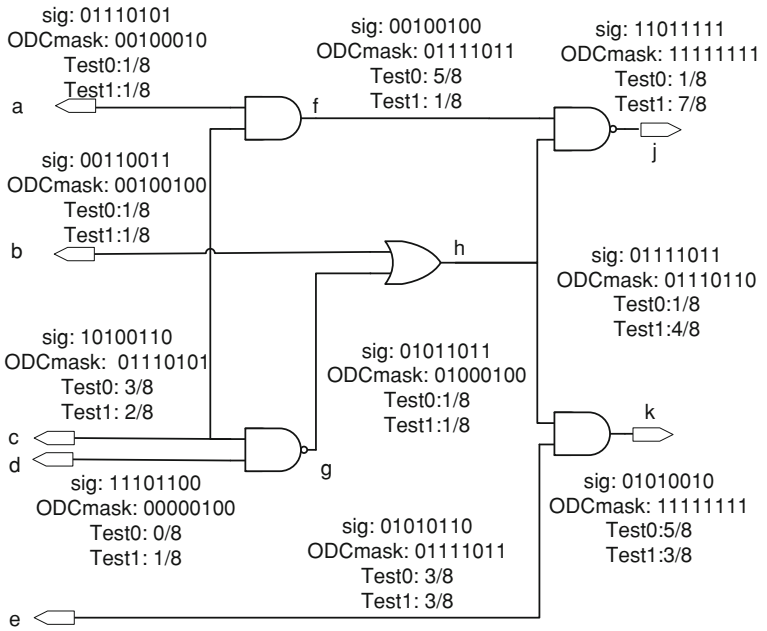


Fig. 5.2 Signatures, ODC masks, and testability information associated with circuit nodes

We found the heuristic algorithm for ODC mask computation presented in [6], which has only $O(N)$ complexity, particularly convenient to use. This algorithm, shown in Fig. 5.3b, traverses the circuit in reverse topological order and, for each node, computes a local ODC mask for its immediate downstream gates. The local ODC mask is derived by inverting the signature in question and checking if the signature at the gate output changes. The local ODC mask is then bitwise-ANDed with the respective global ODC mask at the output of the gate to produce the ODC mask of the gate for a particular fan-out branch. The ODC masks for all fan-out branches are then ORed to produce the final ODC mask for the node. The ORing takes into account the fact that a node is observable for an input vector if it is observable along any of its fan-out branches. Reconvergent fan-out can eventually lead to incorrect values. The masks can then be corrected by performing exact simulation downstream from the converging nodes. This step is not strictly necessary for SER evaluation, as we show later.

Example 5.1 Fig. 5.2 shows a sample 8-bit signature and the accompanying ODC mask for each node of a 10-node circuit. The ODC mask at c, for instance, is derived by computing ODC masks for paths through nodes f and g, respectively, and then ORing the two. The local ODC mask of c for the gate through f is 01110101. When this is ANDed with the ODC mask of f, we find the global ODC mask 01110001 of c on paths through f. Similarly, the local ODC mask of c for the gate with output g is 11101100, and the global ODC mask for paths through g is 01000100. We get the

(a)

```
compute_odc_exact(Circuit C, size K)
{
 compute_sigs(C,K)
 sort_reverse_topological(C)
 for(all nodes g ∈ C)
  newsig(g) =~ sig(g)
  recompute_sigs(C,K,g)
  for(each output o ∈ C)
   ODCmask(g)| = newsig(o) ⊕ sig(o)
  restore_computed_sigs(C)

}
```

(b)

```
compute_odc_approx(Circuit C, size K)
{
 compute_sigs(C,K)
 sort_reverse_topological(C)
 for(all nodes g ∈ C)
  newsig(g) =~ sig(g)
  for(each fan-out branch f ∈ fanout(g))
   sig(f) = Op < f > (inputsigs(f))
   localodc(g, f) = newsig(f) ⊕ sig(f)
   globalodc(g, f) = localodc(g, f)&ODCmask(f)
   ODCmask(g)| = globalodc(g, f)
}
```

Fig. 5.3 **a** Exact and **b** approximate ODC mask computation algorithms

ODC mask of c by ORing the ODC masks for paths through f and g, which yields 01110101.

5.1.3 SER Evaluation

We compute the SER by counting the number of test vectors that propagate the effects of a transient fault to the circuit outputs. Test-vector counting was also used in [8] to compute SER, although the algorithm there uses BDD-based techniques. Intuitively, if many test vectors are applied at the inputs, then faults are propagated to the outputs often. SER computation is inherently more difficult than test generation. Testing involves generating vectors that sensitize the error signal on a node and propagate the signal's value to the output. SER evaluation involves counting the number of vectors that detect faulty signals.

Next, we describe how to compute signatures and ODC masks to derive several metrics that are necessary for our SER computation algorithm. These metrics are based on the signal probability (controllability), observability and testability parameters commonly used in ATPG [7].

Fig. 5.4 Computing SER under the TSA fault model

```
compute_TSA_SER(Circuit C, int K)
{
  compute_sigs(C,K)
  compute_odc_approx(C,K)
  for(all nodes g ∈ C)
    test0(g) = zeros(sig(g)&ODCmask(g))/K
    test1(g) = ones(∼ sig(g)&ODCmask(g))/K
    Perr(C)+ = Perr0(g)test1(g)
    Perr(C)+ = Perr1(g)test0(g)
  return Perr(C)
}
```

Figure 5.4 summarizes our algorithm for SER computation. It involves two topological traversals of the target circuit: one to propagate signatures forward and another to propagate ODC masks backwards. The fraction of 1s in a node's signature, known as its *ones count*, provides an estimate of its signal probability, while the relative proportion of 1s in an ODC mask indicates observability. These two measures are combined to obtain a testability figure-of-merit for each node of interest, which is then multiplied by the probability of the associated TSA to obtain the SER for the node. This SER for the node captures the probability that an error occurs at the node, combined with the probability that the error is logically propagated to the output. Our estimate can be contrasted with technology-dependent SER estimates, which include timing and electrical masking.

We estimate the probability of signal g having logic value 1, denoted $P[g = 1]$, by the fraction of 1s in the signature $sig(g)$. This is sometimes called the controllability of the signal.

Definition 5.6 The *controllability* of a signal g, denoted $P[g = 1]$, is the probability that g has logic value 1.

$$P[g = 1] = ones\big(sig(g)\big)/K \tag{5.1}$$

Definition 5.7 The *observability* of a signal g, denoted $P[obs(g)]$ is the probability that a change in the signals value changes the value of a primary output.

The observability of a node is approximated by the number of 1s in its ODC mask, i.e., the ODC mask's ones count.

$$P[obs(g)] = ones\big(ODCmask(g)\big)/K \tag{5.2}$$

This observability metric is an estimate of the probability that g's value is propagated to a primary output. The 1-testability of g, denoted $P[test_1(g)] = P[obs(g), g = 1]$, is the number of bit positions where g's ODC mask and signature both are 1.

Definition 5.8 The 1-*testability* of a node g is the probability that the node's correct value is 1 and that it is observable.

$$P[test_1(g)] = ones\big(sig(g)\&ODCmask(g)\big)/K \tag{5.3}$$

Similarly, if the number of positions where the ODC mask is 1 and the signature is 0. In other words, 0-testability is an estimate of the number of vectors that test for stuck-at-0 faults.

Example 5.2 Consider again the circuit in Fig. 5.2. The signature for node g is given by $sig(g) = 01011011$ and ODC mask $ODCmask(g) = 01000100$. Hence, $P[g = 1] = \text{ones}(sig(g)) = 5/8$, $P[g = 0] = 3/8$, $P[obs(g)] = 2/8$, $P[test_0(g)] = 1/8$ and $P[test_1(g)] = 1/8$.

Suppose each node g in a circuit C has fault probabilities $Perr_0(g)$ and $Perr_1(g)$ for TSA-0 and TSA-1 faults, respectively, then the SER of C is the sum of SER contributions from each gate g in the circuit. Here, we weight gate error probabilities by the testability of the gate for the particular TSA.

$$Perr(C) = \sum_{g \in C} P[test_1(g)]Perr_0(g) + P[test_0(g)]Perr_1(g) \tag{5.4}$$

Example 5.3 The $test_0$ and $test_1$ measures for each gate in the circuit are given in Fig. 5.2. If each gate has TSA-1 probability $Perr_0 = p$ and TSA-0 probability $Perr_1 = q$, then the SER is given by $Perr(C) = 2p + (13/8)q$, this is obtained by summing the testabilities of all of the gates.

The metrics $test_0$ and $test_1$ implicitly incorporate error sensitization and propagation conditions. Hence, Eq. 5.4 accounts for the possibility of an error being logically masked. Note that $Perr_1(g)$ refers to the 1-controllability of g and so is weighted by the 0-testability, similarly for $Perr_1(g)$.

5.1.4 Multiple-Fault Analysis

In this section, we discuss SER computation for the two multiple-fault models introduced previously: the TMSCA model for multiple correlated faults, and the TMSA model for multiple independent faults.

The SER for TSA faults requires the computation of signatures and ODC masks for each node in the circuit. Each node represents the location of a potential TSA fault, and its ODC mask contains information about the probability of the corresponding fault being observed. The same process can be generally followed for TMSCA faults.

However, ODC masks must then be applied to a set of nodes rather than a single node.

Recall that the exact ODC computation of Fig. 5.3a requires resimulation of the entire fan-out cone of the fault location, while the algorithm from [6], shown in Fig. 5.3b, only resimulates through the immediate successor gate(s). These techniques represent two extremes in error-propagation analysis. Between these extremes, it is possible to resimulate the fan-out cone partially. In fact, the farther through the fan-out cone we resimulate, the more accurate our ODC masks become. For TMSCA faults, we resimulate through the subcircuit consisting of gates in the set of interest G. We then bitwise invert the signatures of all the nodes in this subcircuit and resimulate to either the boundary of the subcircuit (for approximate ODC computation) or through the entire fan-out cone of the subcircuit (for exact ODC computation). This algorithm is shown in Fig. 5.5. In this algorithm, the nodes in the set G are topologically sorted and resimulated by flipping the signature of each node $sig(g)$, to the value $V[g]$. This requires the use of a bit mask called $valsig(g)$ that contains $V[g]$ in every bit position. After the resimulation is completed through G, we check for differences that are propagated to the immediate outputs of G (locally) and combine them with the global ODCs computed at the outputs of G using the bitwise AND operation.

For TMSA faults, each of the gates in G has an independent probability of error. Thus, the difference between computing SER for TMSCA faults and TMSA faults is that the signatures of nodes within G are flipped with independent probabilities. In order to represent this situation, we only flip a fraction of the bits in each signature randomly. The rest of the algorithm remains the same. Since usually a single TMSA fault is of interest, we can compute the exact error propagation probability of a TMSA fault by resimulating through the entire circuit in only linear time. The algorithm for SER computation using a TMSA fault is given in Fig. 5.6. Here, the circuit is resimulated by selectively flipping the bits in the signatures of gates in G. The bits are flipped randomly based on the probability of error $P[g]$ and the value $V[g]$ for each node g. The resimulation is done all the way to the primary outputs, then the primary outputs are checked for any differences that have been propagated.

In practice, random bits of the signature can be bitwise XORed by a mask with p/K ones where p is the probability of error. Such a mask can be created by forming a bit vector with p/K ones that are permuted randomly. Then, when the signature is bitwise XORed with the mask, p/K of the bits are flipped, corresponding to a fault that occurs with probability p.

5.2 SER Analysis in Sequential Logic

Next, we extend our SER analysis to handle sequential circuits, which have memory elements (D flip-flops) in addition to primary inputs and outputs. Recall that the values stored in the flip-flops collectively form the state of the circuit. The combinational logic computes state information and primary outputs as a function of the current

```
compute_TMCSA_SER(Circuit C, nodes G, values V, Perr P)
{
  T = sort_topological(G)
  T' = find_output_nodes(T
  for(each node g ∈ T')
    compute_sig(g)
    if(g ∈ T)
      valsig(g) = create_sig(V[g])
  for(each node g ∈ output(T'))
    diff(g) = sig(g) ⊕ newsig(g)
    ODCmask(G)| = (diff(g)&ODCmask(g))
  return Perr × ones(ODCmask(G))/K
}
```

Fig. 5.5 Computing SER under the TMCSA fault model

```
compute_TMSA_SER(Circuit C, nodes G, values V, errors P)
{
  sort_topological(C)
  for(each node g ∈ C)
    compute_sig(g)
    if(g ∈ G)
      flip_sig_bits(g,V[g],P[g])
  for(each node o ∈ output(C))
    diff(o) = sig(o) ⊕ newsig(o)
    ODCmask(G)| = diff(g)
  return Perr × ones(ODCmask(G))/K
}
```

Fig. 5.6 Computing SER under the TMSA fault model

state and primary inputs. Below, we list three factors to consider while analyzing sequential-circuit reliability.

1. Steady-state probability distribution: It has been shown that under normal operation most sequential circuits converge to particular state distributions [9]. Discovering the steady-state probability is useful for accurately computing the SER.
2. State reachability: Some states cannot be reached from a given initial state, therefore only the reachable part of the state space should account for the SER.
3. Sequential observability: Errors in sequential circuits can persist past a single cycle if captured by a flip-flop. A single error may be captured by multiple flip-flops and result in a multiple-bit error in subsequent cycles. Such errors can then be masked by logic.

The following two sections develop a simulation-based framework to address these issues.

5.2.1 Steady-State and Reachability Analysis

Usually, the primary input distribution is assumed to be uniform, or is explicitly given by the user, while the state distribution has to be derived. In [10], it is shown that most ISCAS-89 and other benchmark circuits reach steady-state because they are synchronizable; in other words, they can be taken to a reset state starting from any state, using a specific fixed-length input sequence. This indicates that the circuits are aperiodic (otherwise, different-length sequences would have to be used from each state) and strongly connected (otherwise some states could not be taken to the reset state).

In order to approximate the steady-state distribution, we perform sequential simulation, using signatures. Assume that a circuit with m flip-flops $L = \{l_1, l_2 \ldots l_m\}$ is in state $S_L = \{s_0, s_1, s_2 \ldots s_m\}$, where each $s_i \in \{0, 1\}$. Our method starts in state S_0 for each simulation run (sets of 64 states are processed in parallel in our implementation). Then, we simulate the circuit for n cycles. Each cycle propagates signatures through the combinational logic and stops when flip-flops are reached. Primary input values are generated randomly from a given fixed probability distribution. At the end of each simulation cycle, flip-flop inputs are transferred to flip-flop outputs, which are, in turn, fed into combinational logic for the subsequent cycle. All other intermediate signatures are erased before the next simulation cycle starts. The K-bit signatures of the flip-flops, at the end of n simulations cycles, define K states. We claim that for a large enough n, these states are sampled from the steady-state probability distribution. Empirical results suggest that most ISCAS-89 benchmarks reach steady-state in 10 cycles or fewer, under the above operating conditions [11].

Additionally, our signature-based SER analysis methods can handle systems that are decomposable. Such systems pass through some transient states and are then confined to a set of strongly connected closed (SCC) states. That is, the system can be partitioned into transient states and sets of SCC states. For such systems, the steady-state distribution strongly depends on the initial states. We address this implicitly by performing reachability analysis starting in a reset state. Thus, each bit of the signature corresponds to a simulation that (1) starts from a reset state and propagates through the combinational logic, (2) moves to adjacent reachable states, and (3) for a large enough n, reaches steady-state within the partition.

Figure 5.7 summarizes our simulation algorithm for sequential circuits. Using this algorithm, simulating a circuit with g gates for n simulation cycles and with K-bit signatures takes time $O(Kng)$. Note that it does not require matrix-based analysis, which is often the bottleneck in other methods [9, 11]. For example, Markov transition matrices encode state-transition probabilities explicitly, and therefore, can be large due to the problem of state-space explosion [9, 11].

Figure 5.8 shows an example of sequential simulation with 3-bit signatures. The flip-flops with outputs x and y are initialized to 000 in cycle 0, T_0. Then the combinational logic is simulated. For cycle T_1, the inputs of x and y are transferred to their outputs, and the process continues. At the conclusion of the simulation, the values

```
simulate_sequential(Circuit C, int K)
{
 for(all flip-flops l ∈ C)
  outputsig(l) = inputsig(l)
 for(all inputs in0 ∈ C)
  in0 = new_random_input()
 compute_sigs(C,K)
}
```

Fig. 5.7 Multi-cycle sequential-circuit simulation

for x and y at T_3 are saved for sequential-error analysis, which is explained in the next section.

Although, we only considered aperiodic systems, we observe that for a periodic system the SER would need to be analyzed for the maximum period D, since the state distribution oscillates over that period. If the FSM is periodic with period D, then we can average over the SER for D or more simulation cycles. Generally, since probabilistic behavior in a sequential circuit can be modeled by a Markov chain, any circuit whose transition matrix is ergodic reaches steady state.

5.2.2 Error Persistence and Sequential Observability

In order to assess the impact of soft faults on sequential circuits, we analyze several cycles through which faults persist, using time-frame expansion. This involves making n copies of the circuit, $C_0, C_1 \ldots C_{n-1}$, thereby converting a sequential circuit into a pseudo-combinational one. In the expanded circuit, flip-flops are modeled as buffers. The outputs from the flip-flops of the k-th frame are connected to the primary inputs of frame $k+1$ (as appropriate) for $0 < k < n-1$. Flip-flop outputs that feed into the first frame ($k = 0$) are treated as primary inputs, and flip-flop inputs of frame n are treated as primary outputs. Fig. 5.9 shows a three-time-frame circuit that corresponds to Fig. 5.8. Here, the primary inputs and outputs of each frame are marked by their frame numbers. Further, new primary inputs and outputs are created, corresponding to the inputs from flip-flops for frame 0 and outputs of flip-flops for frame 3. Intermediate flip-flops are represented by buffers.

Observability is analyzed by considering all n frames together as a single combinational circuit, thus allowing the single-fault SER analysis described in the previous section to be applied to sequential circuits. Other useful information, such as the average number of cycles during which faults persist, can also be determined using time-frame expansion.

After the sequential simulation described in the previous section, we store the signatures of the flip-flops and use signatures to stimulate the newly created primary inputs (corresponding to frame-0 flip-flops) in the time-frame expanded circuit. For instance, the x_0 and y_0 inputs of the circuit in Fig. 5.9 are simulated with the corresponding signatures, marked T_3 (the final signature after multi-cycle simulation is

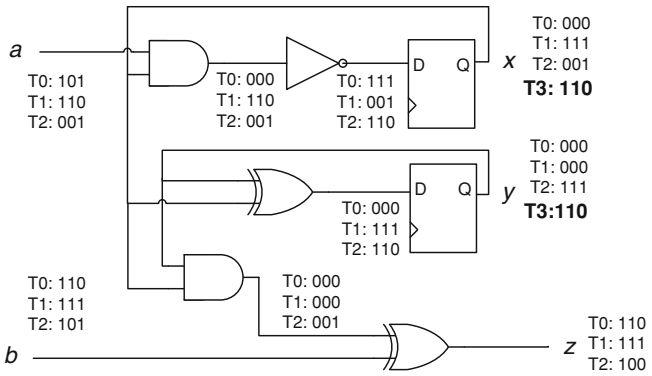


Fig. 5.8 Illustration of bit-parallel sequential simulation

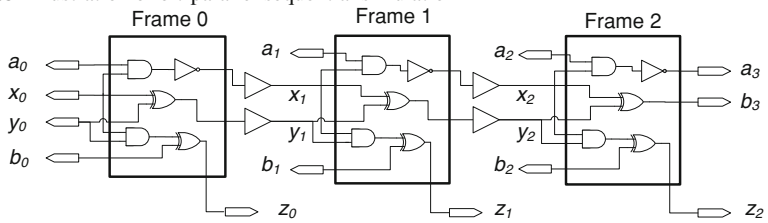


Fig. 5.9 Illustration of time-frame expansion into three frames C_0, C_1 and C_2

finished), from Fig. 5.8. Randomly generated signatures are used for primary inputs not corresponding to flip-flops, such as a_0 and b_0 in Fig. 5.9.

Following simulation, we perform ODC analysis, starting from the primary outputs and flip-flop inputs of the n-th frame and moving all the way to the inputs of the 0-th frame. In other words, errors in primary outputs and flip-flops are considered to be observable. Fig. 5.10 gives our algorithm for sequential SER computation. The value of n can be varied until the SER stabilizes, i.e., it does not change appreciably from an n-frame analysis to an $(n+1)$-frame analysis.

The n-frame ODC-analysis can lead to different gates being seen as critical for SER. For instance, the designer can deem errors that persist longer than n cycles as more critical than errors that are quickly flushed at primary outputs. In this case, the ODC analysis only considers the fan-in cones of the primary outputs of C_n. We denote the ones count of $ODCmask(g, f, n)$ as $seqobs(g, f, n)$. The testability is computed using the signature and ODC mask after n simulations and f frames of unrolling.

$$P[test_0(g, f, n)] = zeros(sig(g, f, n) \& ODCmask(g, f, n))/K \tag{5.5}$$

$$P[seqobs(g, f, n)] = ones\big(ODCmask(g, f, n)\big)/K \tag{5.6}$$

$$Perr(C_0, f, n) = \sum_{g_i \in C_0} P[test_1(g_i, f, n)]Perr_0(g_i) + P[test_0(g_i, f, n)]Perr_1(g_i) \tag{5.7}$$

```
compute_seq_SER(Circuit C, int K, int n, int f)
{
 for (i < n)
  seq_simulate(C,K)
 C' = time_frame_expand(C,f)
 copy_flipflop_inputs(C',C)
 compute_sigs(C',K)
 compute_odc_approx(C',K)
 for (all nodes g ∈ C0)
  test0(g) = zeros(sig(g)&ODCmask(g))/K
  test1(g) = ones(~ sig(g)&ODCmask(g))/K
  Perr(C') += (Perr0(g)test1(g) + Perr1(g)test0(g))
 return Perr(C')
}
```

Fig. 5.10 Computing SER in sequential circuits under TSA faults

The SER algorithm in Fig. 5.10 still runs in linear time with respect to the size of the circuit, since each simulation is linear and ODC analysis (even with n time frames) runs in linear time as well.

5.3 Additional Masking Mechanisms

Recall that researchers have identified three mechanisms by which soft errors are mitigated in combinational circuits: logic masking, electrical masking and timing masking [12]. In previous sections, we estimated logic masking using simulation signatures and ODC masks. In this section, we extend this framework to include timing and electrical masking in a modular, and accurate manner.

We propose linear-time algorithms in the spirit of static timing analysis (STA) for computing the *error latching window* (ELW) and *attenuation factor* of each gate in a circuit. The ELW of a gate in comparison to the clock cycle is used to derive the probability of errors that strike the gate and are capable of latching. The electrical derating factor is an indicator of the probability of propagation of the error without attenuation to a primary output or latch. We also show how to incorporate input-vector dependency into our estimates using logic simulation. This essentially renders our method statistical rather than static.

5.3.1 Incorporating Timing Masking into SER Estimation

This section presents a method for computing ELWs for a sequential circuit with edge-triggered flip-flops separated by combinational logic blocks. The timing constraints associated with each edge-triggered D flip-flop are as follows:

- The data (D) input has to receive all data before the setup time T_s preceding the latching clock edge.
- The data input must be held steady for the duration of the hold time T_h following the latching clock edge.

Soft errors are usually characterized by a transient glitch of duration d that results from a particle strike. If such a glitch is present at the data or clock inputs of a flip-flop during the interval $[T_s, T_h]$, it can result in an incorrect value being latched. If the glitch is present during the setup or hold time, it can prevent a correct value from being latched. Therefore, the ELW of the D-flip flop is simply $[T_s, T_h]$.

The ELW for a gate is computed by (1) translating the ELWs of each of its fanout gates backwards by appropriate path delays, and (2) taking the union of the resulting ELWs. In contrast, during STA we compute only the minimum required time at each gate even though a similar backwards traversal is used. Fig. 5.12 shows the algorithm that computes the union of such intervals. The union of two intervals can result in two separate intervals if the respective intervals are disjoint, or one if the intervals overlap. In general, the latching window for a gate g is defined by a sequence of intervals ELW$(g)[0]$, ELW$(g)[1]\ldots$, where ELW$(g)[i]$ refers to the ith interval in the latching window. Each interval ELW$(g)[i]$ is itself described by its start and end times $[S_{gi}, E_{gi}]$. Hence, we can write

$$\text{ELW}(g) = \left(\left[S_{g1}, E_{g1}\right], \left[S_{g2}, E_{g2}\right], \ldots \left[S_{gn}, E_{gn}\right]\right)$$

An algorithm to compute the ELWs for each gate in a circuit is shown in Fig. 5.11, while Fig. 5.12 shows how two ELWs can be combined.

Example 5.4 A typical ELW computation is illustrated by the circuit in Fig. 5.13. Each wire is marked with a delay and each gate i is assumed to have delay $d(i)$. The corresponding ELWs are:

$$\begin{aligned}
\text{ELW}(F1) &= ELW(F2) = [T_s, T_h] \\
\text{ELW}(i) &= [T_s - d2, T_h - d2] \\
\text{ELW}(g) &= [T_s - d2 - d(i) - d4, T_h - d2 - d(i) - d4] \\
\text{ELW}(h) &= [T_s - d1, T_h - d1] \\
\text{ELW}(f) &= [T_s - d2 - d(i) - d4 - d(g) - d5, T_h - d1 - d(h) - d3]
\end{aligned}$$

Note that f has a larger ELW than other gates because its two output paths have different delays.

We define the *timing masking factor* as the ratio of the ELW to the clock cycle time C. For a node f, the timing masking factor is computed as follows:

$$Tmask(f) = \sum_{i=1}^{n} (E_{fi} - S_{fi})/C$$

```
compute_ELW(Circuit C)
{  reverse_topological_sort(C);
 for(all latches l ∈ C)
  ELW(l) = [T_s(l), T_h(l)];
 for(all gates g ∈ C)
  for(all fanouts f)
    ELW'(f) = translate(ELW(f), delay(l,f))
   ELW(g) = union(ELW(g), ELW'(f));}
```

Fig. 5.11 Computing the error-latching windows (ELW) of a circuit

```
union(ELW(g), ELW(f))
{  for(all intervals ELW(g)[i])
  insert_interval(ELW(g)[i], ELW(f)[1])
 return ELW(f); }
```

```
insert_interval(ELW(g)[i], ELW(f)[j]))
{  if(E_gi < S_fj)
  return insert_before(ELW(f)[j], ELW(g)[i])
 if(S_gi > E_fj)
  if(j == size(ELW(f)))
  return insert_after(ELW(f)[j], ELW(g)[i]);
  else
  return insert_interval(ELW(g)[i], ELW(f)[j+1]);
 S_gi = max(S_gi, S_fj);)
 E_gi = min(E_gi, E_fj);)
 delete ELW(f)[j];
 return insert_interval(ELW(g)[i], ELW(f)[j]); }
```

Fig. 5.12 Computing the union of two ELWs

Taking timing masking into account, the SER contribution of each gate is computed by scaling the testability and error probability by *Tmask*.

$$SER(C) = \sum_{g \in C} \big(test1(g)gerr0(g) + test0(g)gerr1(g)\big)Tmask(g) \tag{5.8}$$

The latching-windows computation method described above can be seen a type of static analysis. Therefore, some intervals (or portions of intervals) correspond to paths that are not traversed frequently. Our aim is to weight each interval in the ELW by the probability that an error occurring within the interval gets latched. In order to compute such a probability, we use bit-parallel logic simulation. Recall that the ones count of the signature of a signal is used as a measure of signal probability and the ones count of the ODC mask is used as a measure of signal observability. Together these measures give an estimate of the testability of the associated stuck-at fault.

We extend this test-vector counting method to account for path faults, i.e., those in which an entire path rather than a single stuck-at signal is sensitized. For our purpose, we consider sets of paths associated with each ELW interval, rather than a single path. Therefore, we associate an *interval ODC-mask* $ODC(f, i)$ with each interval i in an $ELW(f)$. Note that the ODC mask of a particular gate f, (not associated with an interval) is denoted $ODC(f)$. The fraction of 1s in the interval ODC-mask,

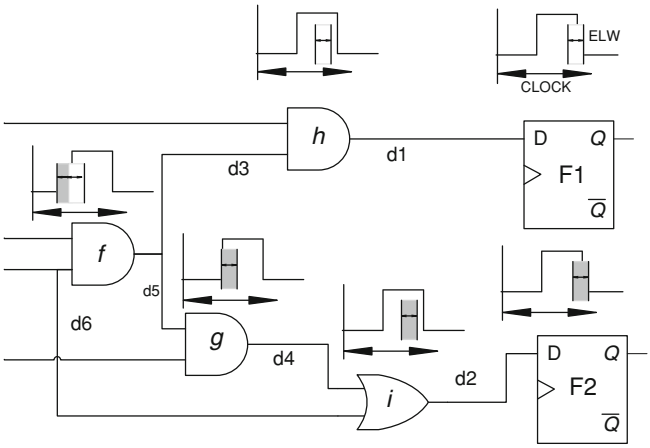


Fig. 5.13 Illustration of error-latching window computation

Fig. 5.14 Computing interval ODCs and electrical derating factors along with ELWs

```
compute_interval_odcs_derating(Circuit C)
{  reverse_topological_sort(C);
 for(all latches l ∈ C)
  ELW(l)[0] = [T_s(l), T_h(l)]
  interval_0DC(l) = 111...'
 for(all gates g ∈ C)
  for(all fanouts f)
   for(all intervals ELW(f)[i])
    ELW(f') = translate(ELW(f)[i], delay(g,f))
    ODC((f',i)) = ODC(f,i)&ODC(g)
    derate(f',i) = derate(f,i) × derate(g)
   ELW(l) = union(ELW(g), ELW(f'))
}
```

denoted $ones(ODC(f, i))$, is the interval weight. As shown in Fig. 5.14, we proceed in reverse topological order by initially considering gates that feed primary outputs. For such gates, all ODC-masks for intervals are simply equal to their ODC-masks (all 1s). For subsequent gates, ELWs are computed by translating and merging the ELWs of sink gates. Here, each interval ODC associated with a sink gate is masked by the ODC of the gate in question to form the interval ODC for the current gate. Intuitively, the interval ODC mask keeps track of the observability of a signal through specific paths. Therefore, masking the ODC corresponding to a path by the ODC of the additional gate simply adds that gate to the path. Note that the runtime of the algorithm remains linear in the size of the circuit even when computing interval weights.

When intervals from two fanout cones are merged, the interval ODC masks are combined using the bitwise OR operation. This ORing results in some lack of accuracy for the weighting algorithm because it averages the weight for both intervals in

```
union_odcs_derating(ELW(g), ELW(f))
{  for(all intervals ELW(g)[i])
   insert_interval(ELW(g)[i],ELW(f)[1])
 }
insert_interval(ELW(g)[i],ELW(f)[j]))
{   if(E_gi < S_fj)
   insert_before(ELW(f)[j],ELW(g)[i])
 if(S_gi > E_fj)
   if(j == size(ELW(f)))
   return insert_after(ELW(f)[j],ELW(g)[i])
   else
   return insert_interval(ELW(g)[i],ELW(f)[j+1])
 S_gi = max(S_gi,S_fj)
 E_gi = min(E_gi,E_fj)
 ODC(g,i) = ODC(g,i) + ODC(f,j)
 derate(g,i) = derate(g,i) + ones(ODC(f,j)) × derate(f,j)
 delete ELW(f)[j]
 return insert_interval(ELW(g)[i],ELW(f)[j])  }
```

Fig. 5.15 Computing the union of two ELWs along with interval ODCs and derating factors

the merged interval. However, this operation is necessary for scalability, since each gate can be subject to exponentially many intervals and the loss of accuracy is small.

Consider a gate f with fanouts g and h. Assume that during ELW computation, intervals $ELW(g)[i]$ and $ELW(h)[j]$ are merged to form $ELW(f)[k]$. In this case,

$$ODC(f,k) = (ODC(g,i) + ODC(h,j))\&ODC(f)$$

The SER computation from interval weights is simply the sum of observabilities corresponding to each interval, weighted by the length of the interval in question.

$$Tmask(f,i) = (E_{fi} - S_{fi})/C$$

$$SER(C) = \sum_{f \in C} \sum_{i \in ELW(f)} \big(ODC(f,i) \times gerr1(f)\big) Tmask(f,i) \tag{5.9}$$

5.3.2 Electrical Attenuation

Recall from Sect. 2.2.2 that we use the glitch-propagation model of [13] to determine which signal characteristics to capture; a different model might require different characteristics to be represented. Glitches are classified into three types depending on their duration D and amplitude A relative to the gate propagation delay T_p and the logic threshold voltage V_s.

- Type 1: If $D > 2T_p$, the glitch passes through un-attenuated because there is sufficient energy to propagate it. The output amplitude A' in this case is V_{dd}.
- Type 2: If $2T_p > D > T_p$, the glitch propagates with its amplitude A diminished by attenuation to $A' = V_{dd}/(VT_1 - VT_2) * A(V_{dd}/2 - VT_1)$, where VT_1 and VT_2 are threshold functions defined in [13]. If $A' < V_s$, then this glitch no longer has the energy to cause a logical error; such type-2 glitches are effectively electrically masked.
- Type 3: If $T_p > D$, the glitch will not propagate at all. Hence, in this case $A' = 0$.

Equivalently, according to [13], the minimum output voltage V_{min} of a gate can be expressed as a linear function of the input glitch amplitude A. Then, when the output voltage has a high logic value V_{dd} and a glitch affects the input, the minimum output voltage can be written as:

$$V_{\min} = \begin{cases} V_{dd} & \text{if } \frac{A}{V_s} < VT_1 \\ \frac{-V_{dd}}{VT_2 - VT_2}\frac{A}{V_s} + \frac{V_{dd}VT_1VT_1}{VT_2 - VT_2} & \text{if } VT_1 < \frac{A}{V_s} < VT_2 \\ 0 & \text{if } VT_1 < \frac{A}{V_s} \end{cases}$$

Again, V_s is the fan-out gate logic threshold which is generally close to $V_{dd}/2$. The thresholds VT_1 and VT_2 are entirely functions of glitch duration and propagation delay.

In order to map the foregoing equations into a probabilistic model, we need to derive the probabilities of glitches being of type 1, 2 or 3 for particular gates. Researchers have measured the spectrum of cosmic ray energies in the atmosphere; see Fig. 5.16. The data imply an exponential flux distribution of the form $Flux = 1.5F(\ln(E))$, where E is the energy of the particle in MeV, and $F(\ln(E))$ is a third degree polynomial in E [14]. The amount of energy that is transferred from a particle strike is known as the *linear energy transfer (LET)*, and is proportional to the particle energy scaled by a factor that depends on the density of the material being struck. It has been estimated that an ionizing particle has a charge of about $10.8fc$ per MeV of LET, for every micron of its track in silicon. Tracks are normally about 2 microns long. Therefore, the charge collected and current produced can be estimated as follows [15]:

$$Q_{col} = 2 \times 10.8fC \times \text{LET}$$

$$I(t) = (\exp^{-t/\tau_a} - \exp^{-t/\tau_b})\frac{Q_{col}}{\tau_a - \tau_b}$$

Here, τ_a and τ_b are charge-collection time constants that depend on process parameters. Therefore, we believe that it is possible to obtain duration-probability ranges directly from the cosmic-ray energy spectrum. From now on, we will assume that each gate g can be characterized to obtain the probability that it experiences glitches of types 1, 2 or 3, denoted by $p_{R1}(g)$, $p_{R2}(g)$, $p_{R3}(g)$, respectively.

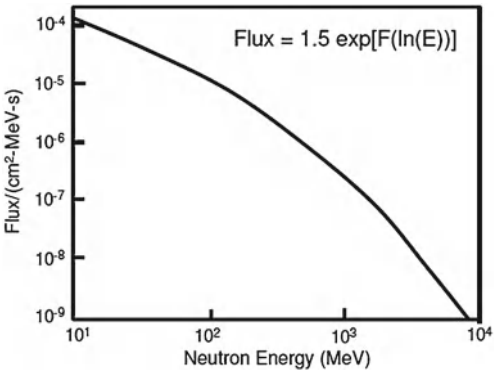


Fig. 5.16 Cosmic ray energy spectrum, as shown in [14]

For a particular gate g, the ELW and the associated interval ODCs can provide an estimate of the probability of a glitch at g being latched. This estimate, however, does not directly take into account the possibility of the error being attenuated on its way to the latch. Therefore, we will explain how to account for that possibility by means of attenuation factors applied to the ELW.

As shown in Fig. 5.15, electrical derating can be computed in a manner similar to the ELW, because it too is a factor that accumulates as the signal passes through a sensitized path. In the ELW case, the window is moved according to the delays it encounters along its path to a latch. Similarly, a derating factor scales down as it passes through gates along a path with the ability to attenuate the errors resulting from particle strikes.

Formally, an *attenuation factor* $att(g, k)$ gives the probability that an SEU occurs at a gate g during ELW interval k and is electrically attenuated out of existence (not propagated). Assuming g has multiple fanouts $f_1, f_2, \ldots f_m$, the attenuation factor is computed recursively based on the following assumptions:

- Glitches of type 1 which occur with probability p_{R1} are immediately eliminated.
- Glitches of type 3 occur with probability p_{R3} and are never attenuated and always propagated.
- Glitches of type 2 are propagated with some attenuation, depending on their duration. Some fraction $df(g)$of these glitches are small enough that their amplitude goes below the threshold voltage of V_s and becomes a logic 0. The remaining glitches, constituting $1 - df(g)$ of the type-2 glitches, are propagated with some attenuation, but their amplitude remains above the threshold voltage. However, those glitches may be eliminated by downstream gates—this is the recursive part of the computation.

$$att(g,k) = p_{R1}(g) + p_{R2}(g)df(g) + p_{R2}(1 - dg(g))$$
$$(1 - prop_2(g,k))$$
$$prop_2(g,k) = \sum_{1<j<m} prop_2(g, f_m, i_m)$$
$$prop_2(g, f_m, i_m) = trans(f_m, g, 2)prop_2(f_m, i_m)df(g) + trans(f_m, g, 1)$$

The variables used in these equations are as follows:

- $prop_2(g,k)$ is the probability that a glitch of type 2 on g is propagated to a primary output or a latch. This says nothing about the probability with which type-2 glitches 2 actually occur at g.
- $prop_2(g, f, i_m)$ is the probability that a glitch of type 2 on g is propagated *through* f to the primary outputs.
- $df(g)$ is the derating factor, which corresponds to the amount by which glitch amplitudes are reduced by g before being passed onto f.
- $trans(f,g,1)$, $trans(f,g,2)$ and $trans(f,g,3)$ are the fractions of type-2 glitches on g that are type-1, -2 or -3 glitches on f.

In the case of a gate adjacent to a latch or primary output.

$$att(g,k) = p_{R1}(g) + p_{R2}(g)df(g)$$

We have discussed how to derive attenuation factors for each ELW interval and have shown earlier how to compute interval ODCs that estimate logic masking during each interval. We now combine the two. As before, suppose that a gate g has multiple fanouts $f_1, f_2, \ldots f_m$ with $ELW(f_1, i_1), ELW(f_2, i_2) \ldots ELW(f_m, i_m)$ merged to form $ELW(g,k)$. The logically sensitized path goes through $g - f_j$ during a particular interval k only when $ODC(g,k)\&ODC(f_j, i_j) = 1$. Note that this may double-count some vectors. But such vectors can propagate through at least two path, which justifies double-counting. Therefore, the modified factor $prop_2(g,k)$ weights different paths g, f_j by the fraction of sensitized paths that go through them. The algorithm to compute the union of two ELWs and derating factors is shown in Fig. 5.15.

$$att(g,k) = p_{R1}(g) + p_{R2}(g)df(g) + p_{R2}(1 - dg(g))(1 - prop_2(g,k))$$
$$prop_2(g,k) = \sum_{1<j<m} prop_2(g, f_m, i_m) \times \left(\frac{ones(ODC(g,k)\&ODC(f_j, i_j))}{ones(ODC(g,k))} \right)$$
$$prop_2(g, f_m, i_m) = trans(f_m, g, 2)prop_2(f_m, i_m)df(g) + trans(f_m, g, 1)$$

The total probability of logically and electrically propagating an upset which latches at a sequential element is given by:

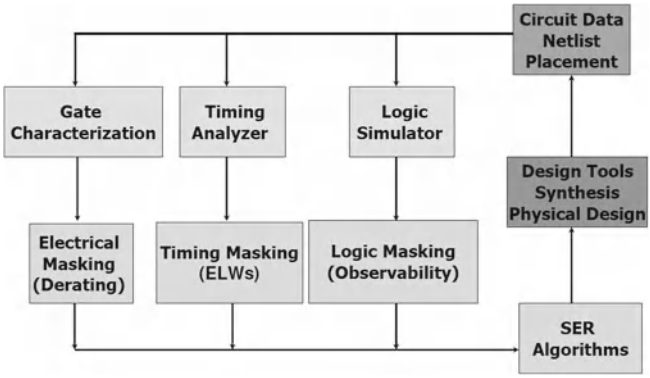


Fig. 5.17 SER evaluation framework including logic, timing and electrical masking

$$SER(C) = \sum_{f \in C} \sum_{i \in ELW(f)} \big(ODC(f,i) \times gerr1(f) \times (1 - att(f,i))\big) Tmask(f,i) \tag{5.10}$$

5.3.3 An Overall SER Evaluation Framework

AnSER is a scalable, lightweight method for guiding logic and physical synthesis flows towards increased reliability. It can also be used to simply check the impact of logic and physical synthesis techniques on reliability, possibly rejecting logic transformations or physical relocations whose impact on reliability is unacceptable. The scalability is in large part due to the efficient linear-time algorithms that are used to compute the impact of logic masking.

Figure 5.17 shows how to incorporate our method into a typical RTL-to-GDSII design flow. After each change to the netlist or placement, AnSER can be invoked incrementally. Physical changes only require the ELWs and signatures of fanin cones to be updated; logic changes can require both input and output cone signatures and ODCs to be updated. Since reliability is expressed as a sum of values in both cases, incremental evaluation involves regenerating signatures and ELWs for each gate in question. Unlike other reliability evaluators which often require a lot of circuit information, the amount of information processed and output by AnSER can be adjusted according to the needs of the user. For instance, if the user wishes to study the impact on reliability of only the timing optimizations steps, then she can use only timing computations in static mode. If a designer is only looking at logic transformations, then the logic-only mode may be used. Therefore, the amount of coupling between the masking mechanisms can also be adjusted. AnSER can be connected to any external timing engine, not necessarily the one used in our work.

Table 5.1 Comparison of SER (FIT) data for AnSER and ATALANTA

Circuit	Number of gates	ATALANTA	AnSER	% Error	AnSER Exact-ODC	% Error
c17	13	6.96E-7	6.96E-7	0.01	6.96E-7	0.01
majority	21	6.25E-6	6.63E-6	6.05	6.57E-6	4.87
decod	25	2.60E-5	2.62E-5	0.83	2.60E-5	0.83
b1	25	1.28E-5	1.31E-5	2.81	1.27E-5	0.78
pm1	68	2.86E-5	3.00E-5	4.70	2.97E-5	3.5
tcon	80	5.30E-5	5.39E-5	1.67	5.35E-5	0.94
×2	86	3.78E-5	3.87E-5	2.38	3.93E-5	3.97
z4ml	92	5.29E-5	5.37E-5	1.50	5.41 E-5	2.20
parity	111	7.60E-5	7.69E-5	1.24	7.71E-5	1.45
pcle	115	5.38E-5	5.34E-5	0.75	5.35E-5	0.56
pcler8	140	7.06E-5	7.24E-5	2.52	7.23E-5	2.41
mux	188	1.58E-5	1.38E-5	12.54	1.63E-5	3.16
Ave.				3.06		2.65

5.4 Empirical Validation

We now report empirical results for SER analysis using AnSER and our two SER-aware synthesis techniques. The experiments were conducted on a 2.4 GHz AMD Athlon 4000+ workstation with 2GB of RAM. The algorithms were implemented in C++.

For validation purposes, we compare AnSER with complete test-vector enumeration using the ATPG tool ATALANTA [16]. We provided ATALANTA with a list all of possible stuck-at (SA) faults in the circuit to generate tests in "diagnostic mode," which calculates all test vectors for each fault. We used an intrinsic gate-fault value of $gerr_0 = gerr_1 = 1 \times 10^6$ on all faults. Since TSA faults are SA faults that last only one cycle, the probability of a TSA fault causing an output error is equal to the number of test vectors for the corresponding SA fault, weighted by their frequency. Assuming a uniform input distribution, the fraction of vectors that detect a fault provides an exact measure of its testability. Then, we computed the SER by weighting the testability with a small gate fault probability, as in Eq. 5.4. While the exact computation can be performed only for small circuits, Table 5.1 suggests that our algorithm is accurate to about 3 % for 2,048 simulation vectors. More test vectors can be used if desired.

We isolate the effects of the two possible sources of inaccuracy: (1) sampling inaccuracy, and (2) inaccuracy due to approximate ODC computation. Sampling inaccuracy is due to the incomplete enumeration of the input space. Approximate ODCs computed using the algorithm from [6] incur inaccuracy due to mutual masking. When an error is propagated through two reconvergent paths, the errors may cancel each other. However, the results in Table 5.1 indicate that most of the inac-

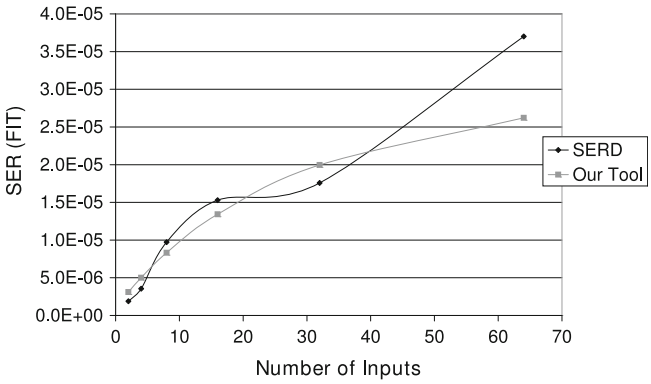


Fig. 5.18 Comparison of SER trends on inverter chains produced by SERD [18] and AnSER

curacy is due to sampling, not approximate ODCs. The last two columns of Table 5.1, corresponding to exact ODC computation, show an average error of 2.65 %. Therefore, only 0.41 % of the error is due to the approximate ODC computation. On the other hand, while enumerating the entire input space is intractable, our use of bit-parallel computation enables significantly more vectors to be sampled than other techniques [17–19] given the same computation time.

To characterize the gates in the circuits accurately, we adapted data from [18], where several gate types are analyzed in a 130 nm, $1.2V_{DD}$ technology via SPICE simulations. We use an average SER value of $gerr_0 = gerr_1 = 8 \times 10^{-5}$ for all gates. However, the SER analyzers from [17, 18, 20] report error rates that differ by orders of magnitude. SERA tends to report error rates on the order of 10^{-3} for 180 nm technology nodes, and FASER reports error rates on the order of 10^{-5} for 100 nm. Furthermore, although our focus is logic masking, we also approximate electrical masking by scaling our fault probabilities at nodes by a small derating factor to obtain trends similar to those of [18]. In Fig. 5.18, we compare AnSER and SERD when computing SER for inverter chains of varying lengths. Since there is only one path that is always sensitized in this circuit, it helps us estimate the derating factor.

Table 5.2 compares AnSER with the previous work on ISCAS 85 benchmarks, using similar or identical host CPUs. While the runtimes in [21] include 50 runs, the runtimes in [18] are reported per input vector. Thus, we multiply data from [18] by the number of vectors (2,048) used there; our runtimes appear better by several orders of magnitude. We believe that this is due to the use of bit-parallel functional simulation to determine logic masking, which has a strong input-vector dependency. Most other work uses fault simulation or symbolic methods.

Table 5.3 shows SER and runtime results for the IWLS 2005 benchmark suite [22], which were evaluated when we implemented AnSER within the OAGear package. Note that our algorithm scales linearly in the size of the circuit, unlike the majority of prior algorithms. We assume a uniform input distribution in these experiments, although AnSER is not limited to any particular input distribution. An input distrib-

Table 5.2 Runtime comparisons of four SER analyzers

Circuit	Number of gates	Time (s)			
		AnSER	SERD [18]	FASER [20]	[21]
c432	246	<0.01	10	22	—
c880	591	< 0.01	10	—	—
c1355	746	0.014	20	40	2.09
c1908	760	0.015	20	66	0.781
c3540	1951	<0.01	60	149	5m42s
c6280	4836	1.00	120	278	—

Table 5.3 SER (in FITs) and runtime for AnSER on the IWLS 2005 benchmarks

Circuit	Number of gates	SER (FIT)	Time (s)
pci_conf_cyc_addr_dec	97	4.89E-3	0.23
steppermotordrive	226	8.00E-3	0.27
ss_pcm	470	1.68E-2	0.3
usb_phy	546	1.53E-2	0.28
sasc	549	2.10E-2	0.26
simple_spi	821	2.50E-2	0.3
i2c	1142	2.7E-2	0.34
pci_spoci_ctrl	1267	0.029	0.342
des_area	3132	0.019	0.782
spi	3227	0.118	0.68
systemcdes	3322	0.127	0.55
tv80	7161	0.104	0.91
systemcaes	7959	0.267	0.97
mem_ctrl	11440	0.494	1.36
ac97_ctrl	11855	0.409	1.38
usb_funct	12808	0.390	1.42
pci_bridge32	16816	0.656	1.78
aes_core	20795	0.550	2.1
wb_conmax	29034	1.030	4.18
ethernet	46771	1.480	5.77
des_perf	98341	3.620	9.34
vga_lcd	124031	4.800	11.7

ution supplied by a user, a sequential gate-level simulator, or a Verilog simulator can be used directly, even if it includes repeated vectors. SER and runtime results with exact and approximate ODCs are shown for the ISCAS-85 benchmarks in Table 5.4. Again, the results show that approximate ODCs are sufficient for most benchmark circuits, since the loss of accuracy due to ODC approximation is negligible.

Table 5.5 compares the multi-cycle simulation runtimes of AnSER with those of MARS-S, the sequential circuit SER analyzer from [11]. MARS-S employs symbolic simulations, using a BDD/ADD-based method to compute steady-state probability

Table 5.4 SER evaluation of various benchmarks with exact and approximate ODCs

	Number of gates	SER (FIT)	Time (s)	SER (s)	Time (s)
alu4	740	1.13E-2	0.227	1.19E-2	0.004
b9	14	4.67E-3	0.007	4.69E-3	0.005
b1	114	6.79E-3	0.050	6.69E-3	0.000
C1355	536	1.93E-2	2.010	1.93E-3	0.034
C3540	1055	3.06E-2	0.409	3.07E-2	0.080
C432	215	5.70E-3	0.056	5.71E-3	0.016
C499	432	1.75E-2	0.291	1.71E-2	0.260
C880	341	1.50E-2	0.54	1.51E-2	0.23
cordic	84	9.43E-2	0.007	9.43E-2	.004
dalu	1387	2.18E-2	0.535	2.17E-2	0.225
des	4252	2.04E-1	5.283	2.03E-1	4.87
frg2	1228	3.61E-1	0.217	3.65E-1	0.169
i10	2824	1.03E-1	1.063	1.04E-1	0.315
i9	952	5.07E-2	2.237	5.06E-2	2.044

Table 5.5 Comparison of multi-cycle simulation runtimes

Circuit	Number of gates	Number of cycles	Time (s)	
			MARS-S	AnSER
s208	112	10	1000	1
s298	133	10	6900	0
s444	181	10	365	4
s526	214	5	551	11
s1196	547	5	68	8
s1238	526	4	70	8

distributions, while we use signature-based bit-parallel functional simulation. The number of cycles needed to reach steady-state is also listed. Table 5.6 shows the results of SER analysis on sequential circuits from the ISCAS-89 benchmark suite under time-frame expansion. The listed runtimes in Table 5.6 are for processing signatures and ODCs on 10 frames. These results indicate that the SER obtained by considering only one time frame is 62 % higher than the 2-frame SER. After this point, increasing the number of frames has little effect on the SER. This indicates that most faults, if at all propagated, are usually observable at primary outputs in the current cycle. This result is supported by observations in [8]. In other words, flip-flops propagate few errors to the outputs in later cycles, due to sequential circuit masking. The latched errors tend to quickly dissipate after a few cycles. This leaves the SER for multiple-cycle analysis close to the error rate of the current cycle's primary outputs.

Table 5.7 shows SER results under the TMSA model, which represents single-event multiple-bit upsets. In this experiment, we included as TMSA faults, sets of

Table 5.6 Change in SER for sequential circuits with increasing number of time frames

Circuit	Number of gates	Time (s)	SER for n time frames					
			n = 0	n = 1	n = 2	n = 3	n = 4	n = 5
s208	112	9	2.40E-3	2.34E-3	2.34E-3	2.33E-3	2.32E-3	2.33E-3
s298	133	9	2.97E-3	2.75E-3	2.69E-3	2.67E-3	2.65E-3	2.62E-3
s400	180	14	4.24E-3	3.00E-3	2.38E-3	2.23E-3	2.23E-3	2.05E-3
s444	181	14	4.69E-3	3.06E-3	2.43E-3	2.18E-3	2.02E-3	1.98E-3
s526	214	9	3.87E-3	2.97E-3	2.65E-3	2.52E-3	2.46E-3	2.44E-3
s1196	547	18	6.35E-3	3.87E-3	3.71E-3	3.68E-3	4.05E-3	3.89E-3
s1238	526	14	6.09E-3	3.54E-3	3.42E-3	3.47E-3	3.72E-3	3.62E-3
s1488	659	5	1.02E-1	1.11E-2	1.03E-2	1.06E-2	1.15E-2	1.07E-2
s1423	731	47	1.43E-2	8.48E-3	5.08E-3	3.47E-3	2.78E-3	2.54E-3
s9234	746	4	1.31E-2	1.24E-2	1.22E-2	1.18E-2	1.07E-2	9.78E-2
s13207	1090	15	3.07E-2	2.66E-2	3.14E-2	3.62E-2	3.61E-2	4.39E-2

Table 5.7 SER under single-event multiple-bit upsets

Circuits	Exact time (s)	Exact SER (FIT)	Approx SER (FIT)
alu4	0.492	2.56e-2	1.00e-2
b1	0.001	2.56e-4	4.62e-4
b9	0.008	2.72e-3	3.30e-3
C1355	0.843	1.82e-2	1.44e-2
C3540	0.992	3.95e-2	2.29e-2
C432	0.129	7.93e-3	6.24e-3
C499	0.589	1.45e-2	1.45e-2
C880	0.087	7.79e-3	7.30e-3
cordic	0.014	2.30e-3	1.25e-3
dalu	0.857	3.89e-3	1.76e-3
des	1.201	0.113e-3	0.139e-3
frg2	0.332	2.75e-2	3.26e-3
i10	0.212	8.07e-2	7.83e-2
i9	0.496	1.87e-3	2.57e-3

topologically adjacent gates 2–3 levels away from a central gate. The results under exact SER are obtained by resimulating the entire fan-out cone. The results under approximate ODC computations, given in Fig. 5.3b, are shown with analysis of 10 levels of logic. The runtime is shown for the exact algorithm.

To evaluate our algorithms involving timing masking, we use the IWLS benchmarks, with design utilization set to 70 % to match recent practice in industry. Our wire and gate characterizations are based on a 65 nm technology library. We perform STA using the D2M delay metric [23] on rectilinear steiner minimal trees (RSMTs) produced by FLUTE [24]. These designs are placed using Capo version 10.2 [25, 26],

Table 5.8 SER evaluation with logic and timing masking

Circuit	Number of gates	Clock period (s)	Logic SER (FIT)	Time (s)	Timing SER (FIT)	Time (s)	Potential improvement
aes_core	20265	5.68E-07	0.1654	6	9.33E-05	3	37.57
spi	2998	3.19E-07	0.05722	1	4.23E-05	1	15.28
s35932	5545	6.18E-07	0.1363	2	6.03E-05	1	26.73
s38417	6714	3.56E-07	0.1360	2	1.22E-04	1	37.83
tv80	6802	6.79E-07	0.05602	2	2.64E-05	1	37.50
mem_ctrl	11062	6.44E-07	0.2185	2	8.45E-05	3	19.64
ethernet	36227	1.46E-06	0.7010	9	1.31E-04	9	91.68
usb_funct	10357	5.06E-07	0.1852	3	8.79E-4	3	36.59

and relocations are legalized, i.e., gates are moved to the nearest empty or legal locations, using the legalizer provided by GSRC Bookshelf [26].

Table 5.8 shows changes in SER when timing masking is considered. Incorporating timing masking into SER can be useful in guiding physical synthesis operations, while only considering logic masking is sufficient for technology-independent logic synthesis steps in the design flow. Table 5.8 also shows the potential for improvement in timing masking, i.e., the improvement in reliability when the ELW of each gate is made as small as possible (equal to the ELW of a latch). This shows that SER can be significantly decreased by manipulating timing masking.

References

1. Krishnaswamy S, Plaza SM, Markov IL, Hayes JP (2007) Enhancing design robustness with reliability-aware resynthesis and logic simulation. In: Proceedings of ICCAD, pp 149–154
2. Krishnaswamy S, Markov IL, Hayes JP (2008) On the role of timing masking in reliable logic circuit design. In: Proceedings of DAC, pp 924–929
3. Krishnaswamy S, Plaza SM, Markov IL, Hayes JP (2009) Signature-based SER analysis and design of logic circuits. IEEE Trans CAD 28:74–86
4. Krishnaswamy S, Plaza SM, Markov IL, Hayes JP (2007) AnSER: a lightweight reliability evaluator for use in logic synthesis. Digest IWLS, In, pp 171–173
5. Zhu Q, Kitchen N, Kuehlmann A, Sangiovanni-Vincentelli A (2006) SAT sweeping with local observability don't-cares. In: Proceedings of DAC, pp 229–234
6. Plaza S, Chang K-H, Markov I (2007) Bertacco V (2007) Node mergers in the presence of don't cares. In: Proceedings of ASP-DAC, pp 414–419
7. Bushnell M, Agrawal V (2000) Essentials of electronic testing. Kluwer, Boston
8. Hayes JP, Polian I, Becker B (2007) An analysis framework for transient-error tolerance. In: Proceedings of VTS, pp 249–255
9. Hachtel GD, Macii E, Pardo A, Somenzi F (1996) Markovian analysis of large finite state machines. IEEE Trans CAD 15:1479–1493
10. Cho H, Jeong Somenzi F, Pixley C (1993) Synchronizing sequences and symbolic traversal techniques in test generation. J Elect Test 4:19–31

11. Miskov-Zivanov N, Marculescu D (2007) Soft error rate analysis for sequential circuits. In: Proceedings of DATE, pp 1436–1441
12. Shivakumar P et al (2002) Modeling the effect of technology trends on soft error rate of combinational logic. In: Proceedings of ICDSN, pp 389–398
13. Omana M et al (2003) A model for transient fault propagation in combinatorial logic. In: Proceedings of OLTS, pp 11–115
14. Ziegler JF et al (1996) IBM experiments in soft fails in computer electronics (1978–1994). IBM J Res Dev 40(1):3–18
15. Wang F, Agrawal V (2008) Single event upset: an embedded tutorial. In: Proceedings of VLSI, pp 429–434
16. Lee HK, Ha DS, On the generation of test patterns for combinational circuits, TR No. 12–93, EE Depertment., Virginia Polytechnic Institute.
17. Zhang M, Shanbhag NR (2004) A soft error rate analysis (SERA) methodology. In: Proceedings of ICCAD, pp 111–118
18. Rao R, Chopra K, Blaauw D, Sylvester D (2006) An efficient static algorithm for computing the soft error rates of combinational circuits. In: Proceedings of DATE, pp 164–169
19. Almukhaizim S et al (2006) Seamless integration of SER in rewiring-based design space exploration. In: Proceedings of ITC, pp 1–9
20. Zhang B, Wang WS, Orshansky M (2006) FASER: fast analysis of soft error susceptibility for cell-based designs. In: Proceedings of ISQED, pp 755–760
21. Choudhury M, Mohanram K (2007) Accurate and scalable reliability analysis of logic circuits. In: Proceedings of DATE, pp 1454–1459
22. IWLS-05 Benchmarks. http://iwls.org/iwls2005/benchmarks.html
23. Alpert CJ, Devgan A, Kashyap C (2000) A two moment RC delay metric for performance optimization. In: Proceedings of ISPD, pp 69–74
24. Chu C, Wong Y-C (2005) Fast and accurate rectilinear Steiner minimal tree algorithm for VLSI design. In: Proceedings of ISPD, pp 28–35
25. Caldwell A, Kahng A, Markov I (2000) Can recursive bisection alone produce routable placements?. In: Proceedings of DAC, pp 693–698
26. UMICH Physical Design Tools. http://vlsicad.eecs.umich.edu/BK/PDtools/

Chapter 6
Design for Robustness

At the gate level, soft errors have been traditionally eliminated from logic circuits through the use of time or space redundancy. The cascaded-TMR scheme illustrated in Fig. 1.10 [1] is an example. Here, the circuit is replicated three times, and the majority value is taken as the result. To protect against faults in the majority voter, this entire circuit ensemble is replicated three times, a process which can be recursively applied until a desired level of fault tolerance is reached. In this chapter, we show how to improve reliability without resorting to explicit or massive redundancy.

Section 6.1 is concerned with increasing the reliability of combinational circuits. It presents a signature-based method to identify partial redundancy and a metric for selecting error-sensitive gates. It also describes a gate relocation technique to improve timing masking. Section 6.2 presents a sequential optimization technique to retime circuits by moving registers, so that fewer latched errors are propagated. Linear programming (LP) is applied to the sequential retiming problem in Sect. 6.3. Finally, Sect. 6.4 offers empirical validation of these techniques. Most of the methods and results described in this chapter also appear in [2–4].

6.1 Improving the Reliability of Combinational Logic

In combinational circuits, an SEU only affects the primary outputs if it is propagated through the intermediate gates. Recall that this phenomenon is known as logic masking. A basic way that designers can improve a circuit's reliability is to ensure that faults are logically masked with high probability. We target logic and timing masking to obtain soft-error-resistant circuits in the following ways: (1) by identifying and using partial redundancy already present within the circuit, to mask errors; (2) by selecting error-sensitive areas of the circuit for replication or hardening; (3) by generating a large number of candidate rewrites for each subcircuit and selecting among them for improvements in area and SER; and (4) by increasing timing masking during physical design.

S. Krishnaswamy et al., *Design, Analysis and Test of Logic Circuits Under Uncertainty*, Lecture Notes in Electrical Engineering 115, DOI: 10.1007/978-90-481-9644-9_6,

6.1.1 Signature-Based Design

We next present a technique called signature-based design for reliability (SiDeR). Using functional simulation, SiDeR identifies redundancy already present in the circuit and utilizes it to increase logic masking. As discussed in Chap. 5, signatures provide partial information about the Boolean function of a node. Therefore, candidate nodes with similar functionality can be identified by matching signatures.

SiDeR takes advantage of the fact that nodes need not implement identical Boolean functions to bolster reliability. Any node that provides predictable information about another can be used to mask errors. For instance, if two internal nodes x and y satisfy the property $(y = 1) \Rightarrow (x = 1)$, where $\Rightarrow$ denotes "implies", then y gives information about x whenever $y = 1$. More generally, if $f(x_0, x_1, x_2, \ldots, x_n) = x$, then x can be replaced by f to logically mask errors that are propagated through x. However, errors at x are only masked in cases where x does not control f. The probability that x controls f can be determined by reevaluating the SER, with the modified node in place.

Additionally, we can increase the number of potential candidates that can replicate x by taking ODCs into account. Instead of searching for candidates where $f(x_0, x_1, x_2, \ldots, x_n) = x$, we search for those such that $f(x_0, x_1, x_2, \ldots, x_n)$ & care $(x) = x$&care(x). Here, care(x) is the function representing the care-set of x, i.e., the set of all input vectors that generate a required 0 or 1 value for x. In terms of signatures, this corresponds to bitwise ANDing sig(f) and sig(x) by ODCmask(x) to check for the following relation:

$$\text{sig}(f)\&\text{ODCmask}(x) = \text{sig}(x)\&\text{ODCmask}(x)$$

Figure 6.1a shows an example of replicated logic for node a, derived by utilizing don't-care values and signatures.

In order to limit area overhead, the function f must be efficiently constructed from $x_0, x_1, \ldots, x_n$. Therefore, we only consider cases where f is implemented by a single AND or OR gate. We add redundant logic by transforming node x into OR(x, y). This means that either $(y = 1) \Rightarrow (x = 1)$ or $(x = 1) \Rightarrow (y = 1)$, which makes candidate pairs x and y easy to identify.

When OR$(x, y) = x$, it follows that sig$(x) >$ sig(y) in lexicographic order; otherwise, sig(y) is 1 in a position where sig(x) is not. Therefore, lexicographically sorting the signatures can narrow the search for candidate signals y. Also, sig(x) must contain more 1s than sig(y), i.e., $|\text{sig}(x)| > |\text{sig}(y)|$, where $|\text{sig}(x)|$ is the size of the signature. Thus, maintaining an additional list of size-sorted signatures and intersecting the two lists can prune the search. Multiple lexicographical sorts and multiple size sorts of signatures starting from different bit positions can further narrow the search. For instance, if we sort the signatures lexicographically from the ith bit, sig(x) must still occur before sig(y), for the same reason. As a result of these properties, signature-based redundancy identification can efficiently perform logic-implication analysis.

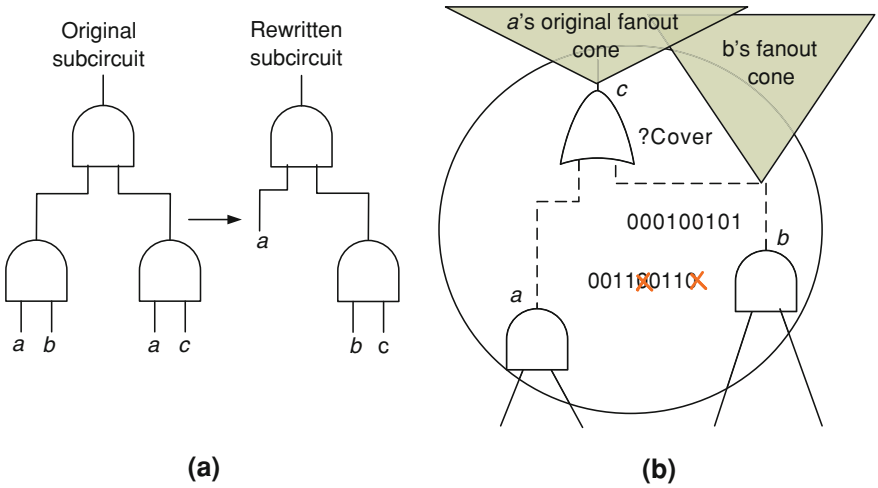


Fig. 6.1 **a** Rewriting a subcircuit to improve area. **b** Finding a candidate cover for node a

Generally, several candidates satisfy implication relations for each node x. Among the candidates, we choose a node y that (1) most often controls the output of the added OR/AND gate, and (2) whose fanin cone is maximally disjoint from that of x. Errors in the mutual fanin cone can be propagated both through x and y. Hence, the additional OR or AND gate would not stop propagation in these cases. However, in order to decide exactly between candidates, it is necessary to evaluate the SER for each potential modification. The high speed of our linear-time SER computation algorithm allows for this, in most cases.

Once we find candidates for resynthesis, a SAT solver can be used to verify the implication relation. The basic process of verifying circuit optimizations with SAT is as follows [5]. Two copies of the circuit are constructed, the original C and the modified version C'. To check if $C = C'$, each output of C and corresponding output of C' are connected to a so-called miter (actually an XOR gate). The outputs of all the miters are fed into an OR gate. This entire ensemble (containing C, C', the miters, and the OR gate) is converted into a SAT instance. A SAT engine then checks if the output of the OR gate is ever 1 (satisfied). If it is, the two circuits cannot be equivalent. In our case, the modified circuit contains $f(x, y)$ in place of x. ODCs are taken into account by feeding the primary outputs (rather than earlier signals) into the miters. In this case, only those differences between C and C' that are observable from the primary outputs result in a 1 at the output of the miters. However, it is possible to decrease the size of the SAT instance by using cuts that are closer to f and x as the inputs of the miters. In [6], verification is done incrementally, starting from f and x, and moving closer to the primary outputs if the relation is unverified.

Fig. 6.2 Algorithm to approximate *impact*

```
approx_impact(Circuit C)
{
 sort_topological(f,C)
 for(all gates f∈C)
  for(all g∈inputs(f))
   impactsig(f)| = impactsig(g)&ODCmask(f)
  impactsig(f)| = relODCmask(g,f)
  impact(f) = p∗ones(impactsig(f))/K
}
```

6.1.2 Impact Analysis and Gate Selection

Gate selection is important in many optimization methods to improve SER. For instance, gate selection is used by SiDeR to limit the area overhead, and the same is true of techniques that harden gates [7]. Gate hardening refers to the use of larger gates with higher critical charge Q_{crit} in order to electrically mask more errors. When a gate is hardened, it does not just change the SER contribution of that particular gate but can also mask errors propagated from its fanin cone. For instance, if a gate f is hardened, and a gate $f' \in$ fan-in(f) is smaller than f, then errors occurring in f' can also be stopped by f. Therefore, in deciding which gates to harden, it is important also to account for the error probability of gates in the fanin cone. In the case of multiple faults, hardening a gate can affect the whole circuit. For instance, if f masks certain errors, they can alter the propagation of other errors in the fanin of the fanout cone of f.

We define the improvement in SER, when a subcircuit c (possibly consisting of a single gate) is changed to a subcircuit c', as the *impact* of c with respect to the change (c, c'). Formally, impact$(c, (c, c')) = \text{Perr}(C'_c) - \text{Perr}(C_c)$, i.e., the difference in the SER of the entire circuit C when c is replaced by c'. However, this is not an efficient method for practically identifying high-impact gates. For instance, evaluating the impact of each gate with respect to replication takes time $O(n^2)$ for a circuit with n gates. Therefore, we provide an approximate algorithm to assess the impact of gates. Our algorithm, given in Fig. 6.2, runs in linear time and employs a notion of the observability of one node g relative to another node f.

$$\text{relODCmask}(g, f) = \text{ODCmask}(g)\&\text{ODCmask}(f) \tag{6.1}$$

The algorithm works by keeping a running signature called impactsig(f) at each node f, which is an indication of the faults propagated to f through paths from its fanout cone.

In general, nodes closer to the primary outputs are more observable than those closer to the primary inputs. However, a node g in the fanin cone F of node f may be more observable than f, due to fanout in F. For the circuit in Fig. 5.2, relODCmask$(g, h) = 01000100\&01110110 = 01000100$. If Perr $= p$, then when including faults on h itself, the impact of h is $5p/8 + 2p/8 = 7p/8$. In cases where some gates have higher intrinsic-error probabilities than others, an average value of

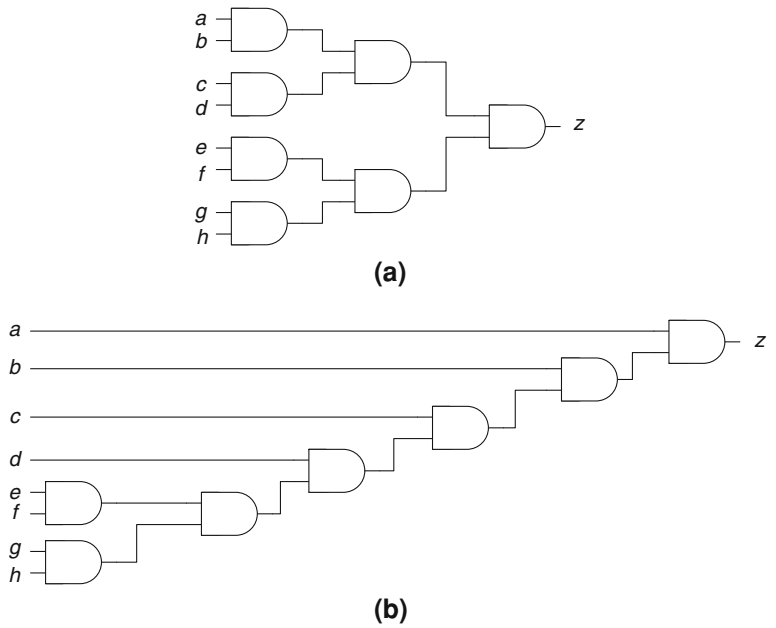


Fig. 6.3 Two different realizations of an 8-input AND

p can be used. For gate hardening, this measure can also be modified by weighting each node f with the width of its ELW, as in Eq. 5.9.

6.1.3 Local Logic Rewriting

Rewriting is a general technique that optimizes small subcircuits to obtain overall area improvements [8]. We optimize circuits for SER and area simultaneously by using AnSER to accept or reject rewrites. This technique relies on the fact that different irredundant circuits corresponding to the same Boolean function can exhibit different SER characteristics. For instance, the balanced AND tree in Fig. 6.3a is more error-tolerant than the imbalanced one of Fig. 6.3b, if the input vectors are distributed uniformly. However, when $P[a = 0] = 0.8$, the imbalanced tree actually has lower SER. Due to this dependence on signal probability, choosing such cases is difficult—this is precisely where AnSER's speed can aid in deciding between certain optimizations for a particular subcircuit.

We use the implementation of rewriting reported in [8, 9], which first derives a 4-input cut for a selected node, defining a one-output subcircuit. Next, replacement candidates are looked-up in hash tables that store several alternative implementations of each function. We rewrite 4-input subcircuits to both improve area and reliability.

To ensure global reliability improvement, we resimulate the circuit and update SER estimates. Computational efficiency is achieved through fast incremental updates by AnSER. As shown in Fig. 6.1a, the original subcircuit with three gates can be rewritten with two gates. New nodal equivalences for the rewritten circuit can quickly be identified using structural hashing to further reduce area.

6.1.4 Gate Relocation

Here, we consider ways to enhance timing masking, rather than logic or electrical masking. The timing-masking characteristics of a circuit can be improved by reducing the width of gate ELWs. Gates with many different-length paths to outputs have the largest latching windows, due to uneven path delay. Therefore, timing masking can be improved if some fanout paths are eliminated or if the paths are modified such that the ELWs from the paths have greater overlap.

Embedding an SER analyzer within a placement tool or closely coupling a placement algorithm with reliability goals is one way of tackling this problem. However, in order to be compatible with all placement algorithms, we take a less intrusive approach, by making local changes to pre-placed designs. Specifically, we relocate nodes within the bounding box defined by their adjacent gates; global characteristics of the placement are maintained in this way.

If a gate f has two fanout branches g and h, then ELW(f) can be translated by adding or subtracting delay from the g-to-f path and the g-to-h path in such a way that the overlap is maximized when ELW(g) and ELW(h) are merged to form ELW(f). The problem of computing a locally optimal position for a gate f, i.e., an (x, y) position such that the ELWs of its successor gates maximally overlap, is a nonlinear constrained optimization problem that can be difficult to solve for even one gate. We conjecture that the best location is likely to be near the center of gravity of the sources and sinks of the gate; neighboring locations should be tried as well. We move in reverse-topological order because the latching windows of gates near primary outputs affect the latching windows of earlier gates, but not vice versa. Our results suggest that these gate relocations can improve reliability while maintaining delay. When interconnect delay forms a large portion of circuit delay, we expect this technique to decrease SER even more.

Figure 6.4 illustrates gate relocation. Here, gate h is moved from the position shown in Fig. 6.4a to the position in Fig. 6.4b to make ELW(f) smaller. Recall that ELW(f) is computed by translating and merging ELW(g) and ELW(h). The relocation results in the ELW(h) being translated by the new path delay between f and h, which has greater overlap with ELW(h) when translated and merged.

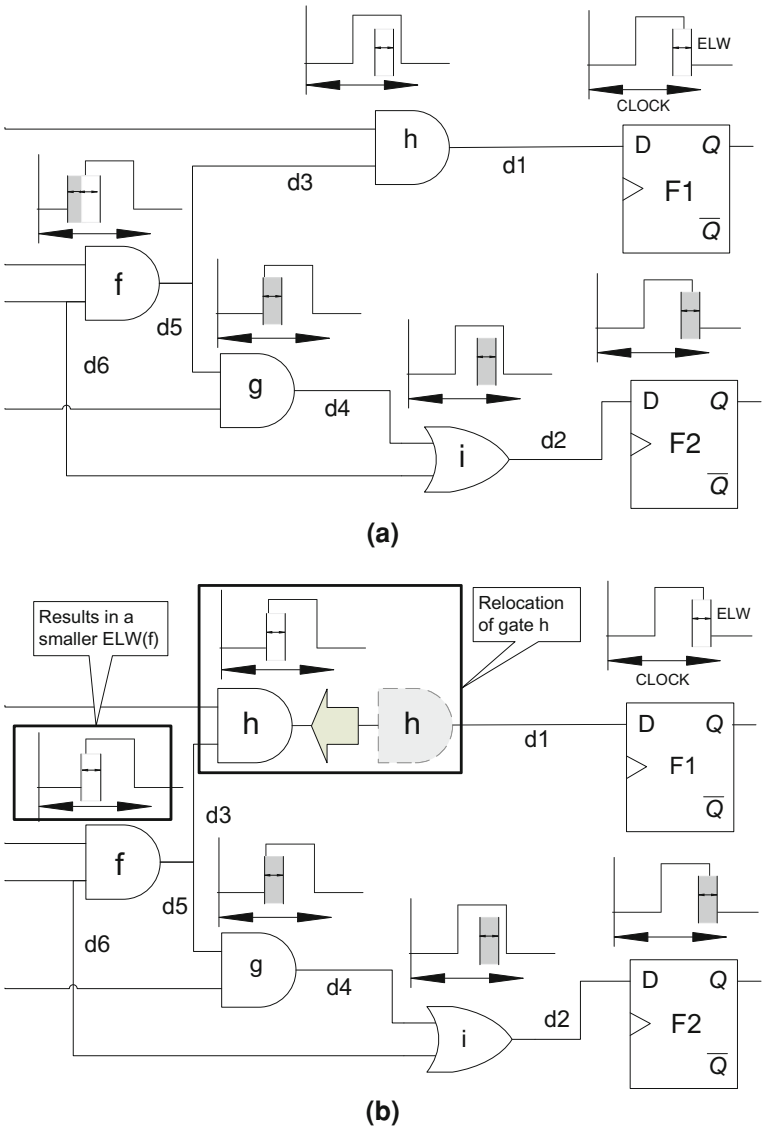


Fig. 6.4 **a** Original circuit with ELWs; **b** modified circuit with gate *h* relocated to decrease the size of ELW(f)

6.2 Improving Sequential Circuit Reliability and Testability

Thus far, we have focused on methods to improve circuit reliabiliity by analyzing combinational logic masking mechanisms. In this section, we utilize sequential SER analysis to reduce the error vulnerability of sequential elements such as flip-flops

and registers. These elements have a key role in circuit testing, since they are often configured as scan chains in testing mode to read out responses to test vectors. Circuit test is becoming more important because fabrication process parameters are harder to control in sub-wavelength lithography. Fluctuations in dopant concentrations, transistor gate length, wire shapes, and via alignments can lead to hard errors in chips [10]. Therefore, methods that decrease SER without compromising testability are desirable to identify incorrectly manufactured chips.

Heidel et al. [11] observe that the large number of registers in high-performance ICs (required for higher operating frequency) is a major contributor to failure rates, since latched errors are often not subject to electrical or timing masking. Errors in combinational logic are only becoming problematic now, while errors in registers are already a problem for critical applications [11]. We aim to reduce the soft-error susceptibility of registers, while simultaneously improving circuit testability. The main idea is to design circuits such that even if a register experiences an SEU, its chances of propagating to a primary output are small. We account for logic masking in both combinational logic and state elements during sequential operation and use this information to improve both the overall SER and testability of the design through retiming. Since we focus on logic masking, our solution is applicable to the various sources of errors mentioned above.

Retiming is the process of relocating registers to improve some design objective (usually area or clock period) such that the functionality of the circuit remains unchanged. An example is shown in Fig. 6.5, where retiming is used to minimize sequential observability. In this example, the register that is at the output is pushed in through the AND gate. Although this increases the number of registers in the design, the observability of these registers is only $2/8$ each, as compared to full observability at a primary output. If the probability of a soft fault on any register is p, then the probability of obtaining an erroneous output is reduced from p to $p/2$.

Our approach to retiming exploits the close relations between signal observability, soft-error propagation, and random-pattern testability. We also construct linear programs (LPs) to solve the problem of relocating registers in order to minimize their sequential observability.

6.2.1 Retiming and Sequential SER

Leiserson and Saxe [12] first developed algorithms for minimum-period and minimum-area retiming of edge-triggered circuits. For the minimum-area retiming problem, a sequential circuit is usually represented by a graph $G(V, E)$, where each vertex $v \in V$ denotes a combinational gate or node, and each edge $(u, v) \in E$ denotes a wire between a driver u and sink v. An edge is labeled by a weight $w(u, v)$, indicating the number of registers (flip-flops) between u and v. The objective of minimum-area retiming is to determine labels $r(v)$ for each vertex v such that the total sum of edge weights is minimized. Here, $r(v)$ denotes the number of registers that are moved from the outputs to the inputs of v. The weight of an edge after

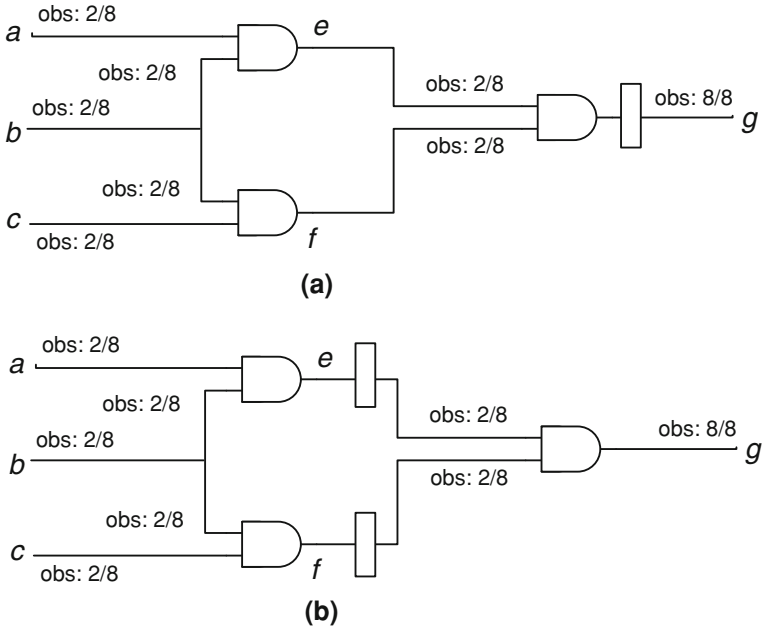


Fig. 6.5 Illustration of observability reduction through retiming

Fig. 6.6 LP for minimum-area retiming

```
Minimize
  Σ_(u,v)∈E w(u,v) − r(u) + r(v)

subject to
  ∀(u,v) ∈ E, r(u) − r(v) ≤ w(u,v)
```

retiming is given by:

$$w_r(u, v) = w(u, v) - r(u) + r(v)$$

Therefore, the total number of registers in the retimed circuit can be minimized by finding a minimum value of the following expression.

$$\sum_{(u,v)\in E} w(u, v) - r(u) + r(v)$$

Additionally, the retiming labels have to meet *legality constraints*, $w(u, v) \geq r(u) - r(v)$, for each edge to enforce the fact that edges cannot have negative weights. A linear program (LP) formulation of the minimum-area retiming problem is given in Fig. 6.6. Leiserson and Saxe [12] observe that this program is the dual of a min-cost network flow problem and can therefore be solved in polynomial time.

It is also possible to constrain the period in minimum-area retiming by ensuring that every path between two vertices with delay more than the target period P has weight ≥ 1. In minimum-period retiming, a binary search is conducted for the target clock period P and the feasibility of each period according to the legality constraints is checked using the Bellman–Ford algorithm[12].

Aspects of circuit testing have also been improved using retiming. Dey and Chakradhar [13] aim to reduce the lengths of partial scan chains in order to decrease testing time. Das and Bhattacharya [14] observe that combinational redundancies can be converted into sequential redundancies (unobservable changes in the state diagram) to improve the scan-based testability of circuits. In [15], the authors use the reverse process to convert sequential redundancies to combinational ones and then remove this redundancy using combinational optimization techniques. We note that these works are significantly different in their focus from ours. We aim to reduce the average observability—in effect the random-pattern observability of registers during normal operation. However, as most registers are scanned, this retiming improves their observability during testing.

Retiming can improve circuit reliability by relocating registers so that soft errors are more likely to be masked. In order to account for error propagation through multiple stages, we modify the circuit using time-frame expansion. In an n-frame expansion, n copies of the circuit are made and each register is simply replaced by a wire. The outputs of the kth frame are fed back into the inputs of the $(k+1)$st frame, as appropriate. Register-inputs in frame 0 are treated as primary inputs, while register-outputs in frame n are treated as primary outputs.

Recall that sequential observability considers multiple cycles of operation through time-frame expansion. The primary output of each frame as well as registers of the n-th frame are considered completely observable. This accounts for errors that propagate past a single cycle before appearing at a primary output, and for errors which stay latent in the system past n cycles. It is true that some errors in registers at the n-th frame may never appear at primary outputs, but this overestimate is slight since the probability of errors in the n-th frame are small. Experiments reported in [16, 17]. indicate that the majority of errors are flushed out in 2 or 3 cycles.

Additionally, in order to compute observability from a reasonable set of start states we obtain a sample of reachable states by simulating the sequential circuit for 20 cycles starting from a reset state. Experiments have shown that 10–15 cycles of simulation suffice to reach steady state on most ISCAS-89 benchmarks [17, 18]. We denote this measure by $\text{seqobs}(f, n)$, where f is the name of the signal, n is the number of frames of expansion. The sequential observability is given by the fraction of ones in the ODC mask of f in the n-frame expanded circuit.

$$\text{seqobs}(f, n) = \text{ones}\big(\text{ODC}(f, n)\big)/K$$

The SER of a sequential circuit is therefore:

$$\text{SER}(C, n) = \sum_{f \in C} \text{gerr0}(f)\text{seqobs}(f, n)$$

If we separate the contributions of the registers from the contributions of the combinational logic to the SER of the sequential circuit, then we obtain:

$$\begin{aligned}\text{SER\_Comb}(C,n) &= \sum_{f\in\text{Comb}(C)} \text{gerr0}(f)\text{seqobs}(f,n)\\ \text{SER\_Reg}(C,n) &= \sum_{r\in\text{Reg}(C)} \text{gerr0}(r)\text{seqobs}(r,n)\\ \text{SER}(C) &= \text{SER\_Reg}(C,n) + \text{SER\_Comb}(C,n)\end{aligned}$$

The SER_Comb(C, n) portion of the error does not change under register relocation, since functionally, registers simply transmit the input signal to the output unchanged after some delay. Therefore, registers do not logically mask errors when considering multi-cycle operation and in turn, do not affect the observability of other nodes in the circuit. However, SER_Reg(C, n), does change with the movement of registers. For instance, if registers move from the output of a node f to its inputs then, errors on the registers can be additionally logically masked by f. The sequential observability of the a register r is generally (with some exceptions) the same as that of its driving gate; therefore, if a register is moved, its sequential observability changes. This suggests that registers should be placed in locations such that their sequential observability is low.

6.2.2 Retiming and Random-Pattern Testability

Since the signature-based framework computes the testability of a circuit based on random simulation vectors, $\text{test}_1(f)$ computes the random-pattern testability of node f for 0–1 errors, and $\text{test}_0(f)$ computes the same for 1–0 errors. In the absence of registers, the random-pattern testability of the entire circuit can be computed by

$$\text{Rand\_Test}(C,n) = \frac{1}{|\text{Comb}(C)|}\sum_{f\in\text{Comb}(C)} \text{test}_1(f,n) + \text{test}_0(f,n)$$

In modern logic circuits, most registers are directly scanned out and read in test mode, i.e., registers can be treated as primary outputs when considering testability. Therefore, if registers were added to nodes f with low testability, then Rand_Test(C) would increase. This again leads to the conclusion that registers should be placed in regions of low observability.

The random-pattern testability of a circuit can be improved iteratively. At iteration 0, we assume that there are no fixed registers in the design. Therefore, Rand_Test(C, n) is improved by placing registers in locations of low observability—in our case, through retiming. In iteration 1, the testability is analyzed with respect to the current register locations. Additional test points are placed at locations that

have low observability with respect to the circuit of iteration 0, and so on. Hence, iteration 0 of this process involves the same goal as minimizing SER, i.e., decreasing the total sequential observability of registers.

Example 6.1 For the circuit in Fig. 6.5, node testabilities are as follows:

$$\begin{aligned} \text{test}_0(e,1) &= 1/8, \quad \text{test}_1(e,0) = 1/8 \\ \text{test}_0(f,1) &= 1/8, \quad \text{test}_1(e,0) = 1/8 \\ \text{test}_0(g,1) &= 7/8, \quad \text{test}_1(g,0) = 1/8 \end{aligned}$$

The random-pattern testability of this circuit is

$$\text{Rand\_Test}(C,n) = (2/8 + 2/8 + 1)(1/3) = (1/2)$$

Registers should be placed at nodes e and f due to their low observability.

6.3 Retiming by Linear Programs

We now develop LP formulations for retiming, accounting for the sequential observability of each register location. First, we present the basic retiming formulation assuming no register sharing, i.e., if a latch driven by a node u has fanouts v and w. Then we model this as though there was a latch at both (u, v) and (u, w), i.e., $w(u, v) = w(u, w) = 1$. Then, we account for register sharing at fanout branches.

6.3.1 Minimum-Observability Retiming

The sequential observability of each edge (u, v) is the same as the output of a buffer that is placed on edge (u, v). We denote the observability of (u, v) by seqobs$((u, v), n)$, which is computed using signatures as described in Sect. 5.2.1. In the case where u only has one fanout, seqobs$((u, v), n)$ = seqobs(u, n); in other words, the observability is the same as that of its driver. Since registers logically act as buffers, a register output has the same observability as a register input, and edges with registers still have the same observability after the registers are moved. The objective function accounting for total register observability is given by:

$$\sum_{(u,v)\in E} w_r(u, v)\text{seqobs}((u, v), n)$$

Fig. 6.7 Minimum-observability retiming formulation

```
Minimize
  Σ_{(u,v)∈E} (w(u,v) − r(u) + r(v)) seqobs((u,v), n)

subject to
  ∀(u,v) ∈ E, r(u) − r(v) ≤ w(u,v)
```

Fig. 6.8 Area- and period-constrained retiming for minimum-observability

```
Minimize
  Σ_{(u,v)∈E} (w(u,v) − r(u) + r(v)) seqobs((u,v), n)

subject to
  ∀(u,v) ∈ E, r(u) − r(v) ≤ w(u,v)
  (Σ_{(u,v)∈E} w(u,v) − r(u) + r(v)) < M
  ∀u,v ∈ V  s.t.  D(u,v) > P, r(u) − r(v) ≤ W(u,v) − 1
```

Additionally, if u is a primary input then $r(u)$ is necessarily 0, and similarly for v. This ensures that no peripheral retiming is done, and that the overall period of the circuit does not increase beyond the longest combinational path in the module being optimized. The modified LP algorithm is shown in Fig. 6.7.

Example 6.2 For the circuit shown in Fig. 6.5 the edges include (a, e), (b, e), (b, f), (c, f), (e, g), (f, g) and (g, o). Note that input and output wires are also considered valid edges. However, we only derive retiming labels for the intermediate nodes e, f, g. The objective function is

$$w_r(a, e)(2/8) + (w_r(b, e) + w_r(b, f))(2/8) + w_r(c, f)(1/8) + w_r(e, g)(2/8) + w_r(f, g)(2/8) + w_r(g, o)(8/8)$$

The retimed weight of edge (e, g), for instance, is $w_r(e, g) = w(e, g) - r(e) + r(g)$.

Once the circuit has been retimed, registers can be shared again. For instance, if edges (u, v) and (u, w) both have registers after retiming, then these are simply shared. In general, the number of registers required at the output of u is $\max(w_r(u, f_1), w_r(u, f_2) \ldots w_r(u, f_n))$, where $f_1, f_2, \ldots f_n$ are fanouts of u.

The formulation in Fig. 6.7 can be modified to constrain area and clock period. For area constraints, we can perform a binary search for the smallest feasible area M by including the constraint $\sum_{(u,v)\in E} w(u, v) - r(u) + r(v) < M$. The period can be constrained to a target P by the method of [12]. Here, the D matrix stores the delay of longest path between the vertices (u, v) in $D(u, v)$ and the W matrix stores the weight of the said path. These additional constraints are shown in Fig. 6.8.

6.3.2 Incorporating Register Sharing

The preceding discussion leads to a minimum-observability retiming formulation that does not account for register sharing. Hence, the optimization takes place on a version of the circuit with registers cloned at each fanout branch. The difficulty in incorporating sharing is that observability is a non-linear property of edges.

Example 6.3 Consider again the circuit C of Fig. 6.5. Suppose the retimed weights of edges (b, e) and (b, f) are $w_r(b, e) = 2$ and $w_r(b, f) = 1$. According to Fig. 6.7, the objective function for this portion of the circuit equals $(2/8)w_r(b, e) + (2/8)w_r(e, g) = 1/3$. Now, the two registers at (b, f) and (b, e) can be replaced by a single register with fanouts to both e and f. This register does *not* have observability $\text{seqobs}((b, e), n) + \text{seqobs}((b, e), n) = 4/8$. Instead, its observability is computed by counting the fraction of 1 s in its ODC mask. The ODC mask, in turn, is computed as the bitwise OR of the ODC masks through each fanout, as shown in Fig. 5.3. In this case, $\text{ODC}(b) = 2/8$.

For each register with driver u and fanouts $S = \{s_1, s_2, \ldots, s_m\}$, the correct sequential observability must be computed using the method of Fig. 5.3. This observability is equivalent to the seqobs of a buffer with input u and outputs S, denoted $\text{seqobs}((u, S), n)$. Here, S is a subset of all n fanout branches $F_u = \{f_1, f_2, \ldots, f_n\}$ of u. We can compute the total number of registers that can be shared by any subset of these branches using the edge weights introduced earlier, as follows. The number of registers that can be shared by *all* the fanout branches is given by:

$$w_r(u, F_u) = \min(w_r(u, f_1), w_r(u, f_2) \ldots w_r(u, f_n))$$

The number of registers shared by a subset $S = \{f_1, f_2, \ldots, f_{n-1}\}$ of F_u of size $|F_u| - 1$ is $\min(w_r(u, f_1), w_r(u, f_2) \ldots w_r(u, f_{n-1})) - w_r(u, F_u)$. In general, the number of registers shared by S is the minimum weight of any edge of the form $(u, s_i), s_i \in S$, minus the registers that are shared by any larger subset S' of F_u. Therefore, the total weight of a subset of fanout branches S, using the principle of inclusion and exclusion, is given by:

$$w_r(u, S) = \sum_{S' : S \subset S' \subseteq F} (-1)^{(|S'|-|S|)} \min(w_r(u, s'_1), w_r(u, s'_2), \ldots), s'_i \in S'$$

Then these register counts, $w_r(u, S)$, are weighted by their sequential observability $\text{seqobs}((u, S), n)$. The sum of such quantities over all possible subsets of F_u gives us the correct total observability of registers driven by u, assuming maximal sharing. Maximal sharing is desired because a shared register always has observability less than or equal to that of its cloned registers combined.

$$\text{totobs}(u) = \sum_{S : S \subseteq F_u} w_r(u, S) * \text{seqobs}((u, S), n) \tag{6.2}$$

Fig. 6.9 Minimum-observability retiming formulation with register sharing

```
Minimize
  Σ(u)∈V totobs(u) − (c Σ S⊂Fu MINu,S)

subject to
  ∀u ∈ V, S ∈ Fu, ∀(si ∈ S) Minu,S ≤ wr(u, s1)
  ∀(u, v) ∈ E, r(u) − r(v) ≤ w(u, v)
```

The function totobs(u) has to be linearized to incorporate it into the LP. This requires linearizing the min function—the only non-linear element of totobs. Generally, the function $\min(a_1, a_2, \ldots a_n)$, for any real values a_i can be linearized by introducing a new variable MIN, along with the constraints $\text{MIN} \leq a_1, \text{MIN} \leq a_2, \ldots \text{MIN} \leq a_n$. Then, the objective function has to maximize the value of MIN as LP converges to a solution so that $\text{MIN} = \min(a_1, a_2 \ldots a_n)$.

We introduce a variable $\text{MIN}_{u,S}$ for each function $\min(w_r(u, s_1), w_r(u, s_2), w_r(u, s_3) \ldots)$ in totobs(u, F_u). The associated constraints are of the form $\text{MIN}_{u,S} \leq w_r(u, s_1)$, $\text{MIN}_{u,S} \leq w_r(u, s_2)$, etc. Finally, we append $-c(\text{MIN}_{u,S})$ to the end of the objective function for each new variable $\text{MIN}_{u,S}$. Here, c is any sufficiently large constant, i.e., for all S, $c \gg \sum \text{seqobs}((u, S), n)$. Since the retiming linear program has a minimization objective, the additional terms ensure that the $\text{MIN}_{u,S}$ variables are set to their highest (correct) value when the objective is optimized. The remaining retiming variables will be optimized for low observability as before. This altered LP incorporating register sharing is given in Fig. 6.9.

While the formulation in Fig. 6.9 correctly captures register sharing, it can become intractable for nodes with many fanouts. By collecting the coefficients next to the MIN variables, we can rewrite the totobs function as follows:

$$\text{totobs}(u) = \sum_{S \subseteq F_u} C_{u,S} \text{Min}(u, S),$$

$$C_{u,S} = \sum_{S' : S' \subset S \subseteq F_u} (-1)^{|S|-|S'|} \text{seqobs}(u, S', n)$$

From this formulation, it is clear that the number of additional terms generated in the objective function for each node u is on the order of $2^{|F_u|}$. However, many practical circuits have low maximum fanout due to drive-strength limitations of available standard cells.

6.4 Empirical Validation

We now report empirical results for the various design techniques presented above. The experiments were conducted on a 2.4GHz AMD Athlon 4000+ workstation with 2 GB of RAM, and the algorithms were implemented in C++.

Table 6.1 Improvements in SER obtained by SiDeR

Circuit	Area	With exact covers		With approximate covers	
		% SER decrease	% Area increase	% SER decrease	% Area increase
cordic	84	1.7	1.2	27.3	45.2
b9	114	18.1	14.9	30.7	31.6
C432	215	37.6	14.0	38.7	14.9
C880	341	9.6	0.9	13.1	2.3
C499	432	1.0	3.2	32.2	20.6
C1908	432	5.9	9.0	32.4	24.1
C1355	536	25.3	9.0	30.7	8.6
alu4	740	55.9	0.9	55.9	1.6
i9	952	65.4	6.6	65.4	6.6
C3540	1055	31.1	2.2	49.4	3.6
dalu	1387	74.3	1.2	74.3	1.2
i10	2824	40.4	5.4	40.4	5.6
des	4252	11.4	2.9	26.7	4.4
Average	–	29.1	5.5	39.8	13.1

Table 6.1 shows SER and area overhead improvements obtained by SiDeR. The first set of results is for exact implication relationships, i.e., not considering ODCs. The second column shows the use of ODCs to increase the number of candidates. In both cases, AND/OR gates are added based on the functional relationship to be satisfied. We see an average 29 % improvement in SER, with only 5 % area overhead without ODCs. The improvements for the ODC covers are 40 % with area overhead of 13 %, suggesting a greater gain per additional unit area than the partial TMR techniques in [19], which achieve a 91 % improvement but increase area by 104 % on average.

Table 6.2 illustrates the use of AnSER to guide the local rewriting method implemented in the ABC logic-synthesis package [9]. AnSER calculates the global-SER impact of each local change to decide whether or not to accept the change. After checking hundreds of rewriting possibilities, those that improve SER and have limited area overhead are retained. The data indicate that, on average, SER decreases by 10.7 %, while area decreases by 2.3 %. For instance, for `alu4`, a circuit with 740 gates, we achieve 29 % lower SER, while reducing area by 0.5 %. Although area optimization is often thought to hurt SER, these results show that carefully guided logic transformations can eliminate this problem.

Table 6.3 shows the results of combining SiDeR and local rewriting. In this experiment, we first used SiDeR, followed by two passes of rewriting (in area-unconstrained and area-constrained modes) to improve both area and reliability. This particular combination of the two techniques yields 68 % improvement in SER with 26 % area overhead.

We evaluated our gate relocation and gate hardening techniques on circuits from the IWLS 2005 benchmarks, with design utilization set to 70 % to match recent

Table 6.2 Improvements in SER and area with local rewriting

Circuits	Number of gates	Number of rewrites	% SER decrease	% Area decrease	Time (s)
alu4	740	13	29.3	0.5	24.5
b1	14	0	0.0	0.0	0.2
b9	114	8	6.8	0.9	0.3
C1355	536	97	1.2	9.0	37.6
C3540	1055	23	5.8	0.9	51.5
C432	215	68	5.5	1.4	12.1
C499	432	37	0.0	0.5	13.0
C880	341	7	0.2	0.0	5.4
cordic	84	5	1.2	1.2	0.5
dalu	1387	58	24.0	3.2	35.0
des	4252	282	11.2	0.1	12.3
frg2	1228	96	27.9	2.0	8.9
i10	2824	143	5.0	0.6	16.7
i9	952	83	31.4	11.7	35.3
Average	–	–	10.7	2.3	18.1

Table 6.3 Improvements in SER by a combination of rewriting and SiDeR

Circuit	% SER decrease	% Area increase
alu4	95.33	55.41
b1	8.08	14.29
b9	19.88	25.44
C1355	99.49	19.40
C3540	96.02	39.72
C432	96.81	22.79
C499	86.74	14.58
C880	59.58	24.93
cordic	58.34	33.33
dalu	92.68	41.17
des	40.41	−1.69
frg2	46.42	27.85
i10	80.67	2.16
i9	78.05	49.26
Average	68.46	26.33

practice in industry. Our wire and gate characterizations are based on a 65 nm technology library. We perform static timing analysis using the D2M delay metric [20] on rectilinear steiner minimal trees (RSMTs) produced by FLUTE [21]; these designs are placed using Capo version 10.2 [22, 23], and relocations are legalized (i.e., gates are moved into the nearest empty cells) using the legalizer provided by GSRC Bookshelf [23].

Table 6.4 SER improvements through gate hardening

Circuit	% New critical gates	SER (FIT)	% SER decrease
aes_core	21.86	5.57E-05	40.29
spi	53.51	3.15E-05	25.43
s35932	57.03	3.80E-05	36.92
s38417	87.63	7.34E-05	40.30
tv80	33.67	1.39E-05	47.42
mem_ctrl	64.54	5.80E-05	31.36
ethernet	83.51	8.28E-05	36.67
usb_funct	88.96	8.70E-05	90.11
Average	61.34		43.56

Table 6.4 shows improvements achieved by guiding gate hardening. In particular, hardening the top 10 % of the most susceptible gates leads to an average of 43 % decrease in SER. Gates were selected using the impact measure discussed in Sect. 6.1.2. The first column of this table shows the percentage of most-susceptible gates that were not identified using logic masking alone. This indicates that guiding hardening with a timing masking model leads to different gates being hardened.

Table 6.5 shows the results of locally relocating gates within the bounding box of adjacent gates. We only accept changes that affect delay and SER positively. However, the process of legalization, which moves gates into valid empty slots in the layout, can later slightly increase delay. Our results indicate a 14 % improvement at the 65 nm technology node, where average intrinsic gate delay is approximately a 100 times larger than (unit) interconnect RC delay. The second two columns project results to smaller technology nodes where wire delay is expected to become comparable to gate delay. Such trends are indicated in the *international roadmap for semiconductors* which reports that at 32 nm and below, wiring contributes >90 % of the circuit delay.

The first set of results indicates a 14 % decrease in SER, while the second set shows a 41.59 % decrease. Therefore, as technology scales, timing masking can offer greater potential for improvement in SER.

We now describe experiments to validate our proposed retiming formulation. The linear programs are solved using the *lp* problem type on the LP solver program CPLEX v.10.1 [24]. Since these formulations are duals of min-cost network flow problems, and because the problem is unimodular with the right hand sides of constraints being integer, the linear program is guaranteed to have an integer optimal solution [12] without explicitly enforcing integer constraints.

Figure 6.10 summarizes the propagation of errors through sequential circuits, as estimated by bit-parallel functional simulation extended to sequential circuits. The figure indicates that most errors are apparent at the outputs in the cycles immediately after their occurrence. The probability of errors at later cycles diminishes rapidly. Therefore, we only compute observability from the primary outputs over the 10 cycles after an error occurs.

Table 6.5 Improvements in SER, through gate relocation

Circuit	65 nm		<45 nm	
	% SER decrease	% Delay increase	% SER decrease	% Delay increase
aes_core	11.83	3.00	21.15	−3.10
spi	18.87	4.8	41.62	−2.90
s35932	10.74	−0.13	44.02	3.40
s38417	10.10	1.38	14.35	−11.57
tv80	4.89	1.45	43.62	17.50
mem_ctrl	7.75	1.14	78.43	−1.70
ethernet	19.07	0.43	75.17	6.04
usb_funct	28.50	−5.26	14.29	−9.09
Average	13.97	0.55	41.59	2.10

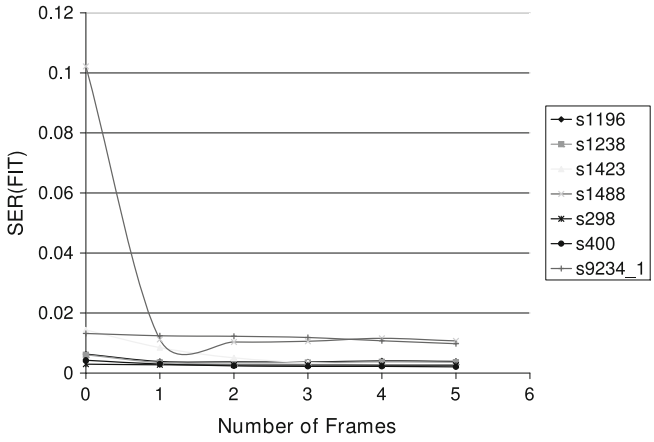


Fig. 6.10 Error propagation in sequential circuits through multiple cycles of operation

Table 6.6 shows the results from the ISCAS-89 benchmark circuits with the minimum-observability retiming formulation. Each edge (u, v) in these is weighted by a 10-frame sequential observability measure. Using the formulation shown in Fig. 6.7, the LP was solved in all cases in less than 0.1 s by CPLEX. The observability, which is proportional to the SER susceptibility, is decreased by an average of 42 % with only a 7 % overall area increase. Table 6.7 shows an average improvement of 31 % in the random-pattern testability of nodes in the combinational portion of the circuit.

Our results indicate an interesting feature of the fault tolerance of logic circuits—that it is possible to decrease the overall sequential SER while increasing what is traditionally computed as the SER of a combinational logic circuit, i.e., with registers treated as primary outputs.This apparent paradox is resolved by the analysis in the previous section, showing that the combinational SER remains unchanged

Table 6.6 Decrease in register observability through retiming

Circuit	Before		After		Change	
	Number of FFs	Total Obs	Number of FFs	Total Obs	% Area Increase	% Obs Decrease
s208	8	9.570	10	9.173	1.785	4.147
s298	14	15.579	34	5.059	15.037	67.527
s344	15	12.770	11	8.539	−2.29	33.126
s444	21	14.752	34	4.146	6.435	71.895
s386	6	4.206	4	3.8256	−1.121	9.035
s526	21	15.906	94	3.567	34.112	77.568
s832	5	25.311	30	2.669	8.561	89.3365
s1238	18	2.848	27	2.147	1.711	24.631
s1196	18	2.911	29	2.115	2.010	27.373
s1423	73	25.273	115	20.115	5.753	20.110
s1488	6	5.744	8	3.8256	0.303	42.787
s1494	6	5.746	7	3.248	0.153	43.47
Avgerage	–	–	–	–	6.53	42.61

Table 6.7 Improvements in random-pattern scan-testability

Circuit	Number of gates	Scan testability		
		Before	After	% Improved
s208	104	0.494	0.499	1.03
s298	119	0.523	0.699	33.72
s386	159	0.360	0.389	7.77
s444	181	0.474	0.689	45.42
s526	193	0.445	0.614	37.75
s823	287	0.222	0.368	65.28
s1238	508	0.300	0.317	5.78
s1196	529	0.302	0.321	6.45
s1494	647	0.269	0.413	53.69
s1488	653	0.267	0.410	53.57
s1423	657	0.424	0.552	30.08
Avgerage	–	–	–	30.96

through register movements when considering sequential behavior. Therefore, it is insufficient to solely consider combinational behavior for SER analysis. While combinational circuit optimization for SER remains important and generally decreases overall circuit SER, it is necessary to account for error-propagation behavior at the sequential level. For instance, combinational circuits can be designed such that they reduce error propagation to registers with high observability.

References

1. von Neumann J (1956) Probabilistic logics and synthesis of reliable organisms from unreliable components. In: Shannon CE, McCarthy J (eds) Automata studies. Princeton University Press, Princeton, pp 43–98
2. Krishnaswamy S, Plaza SM, Markov IL, Hayes JP (2007) Enhancing design robustness with reliability-aware resynthesis and logic simulation. In: Proceedings of ICCAD, pp 149–154
3. Krishnaswamy S, Markov IL, Hayes JP (2008) On the role of timing masking in reliable logic circuit design. In: Proceedings of DAC, pp 924–929
4. Krishnaswamy S, Plaza SM, Markov IL, Hayes JP (2009) Signature-based SER analysis and design of logic circuits, IEEE Trans Comput Aided Des 28(1):74–86
5. Brand D (1993) Verification of large synthesized designs. In: Proceedings of ICCAD, pp 534–537
6. Plaza S, Chang KH, Markov I, Bertacco V (2007) Node mergers in the presence of don't cares. In: Proceedings of ASP-DAC, pp 414–419
7. Mohanram K (2005) Simulation of transients caused by single-event upsets in combinational logic. In: Proceedings of ITC, pp 973–982
8. Mishchenko A, Chatterjee S, Brayton R (2006) DAG-aware AIG rewriting: a fresh look at combinational logic synthesis. In: Proceedings of DAC, pp 532–535
9. Berkeley logic synthesis and verification group. ABC: a system for sequential synthesis and verification. http://www.eecs.berkeley.edu/alanmi/abc/
10. Borkar S et al (2003) Parameter variations and impact on circuits and microarchitecture. In: Proceedings of DAC, pp 328–342
11. Heidel DF et al (2008) Alpha-particle induced upsets in advanced CMOS circuits and technology. IBM J Res Dev 52(3):225–232
12. Leiserson CE, Saxe JB (1991) Retiming synchronous circuitry. Algorithmica 6:5–35
13. Dey S, Chakradhar ST (1994) Retiming sequential circuits to enhance testability. In: Proceedings of VTS, pp 25–28
14. Das DK, Bhattacharya BB (1996) Does retiming affect redundancy in sequential circuits?. In: Proceedings of VLSID, pp 260–263
15. Yotsuyanagi H, Kajihara S, Kinoshita K (1995) Synthesis for testability by sequential redundancy removal using retiming. In: Proceedings of FTCS, pp 33–40
16. Hayes JP, Polian I, Becker B (2007) An analysis framework for transient-error tolerance. In: Proceedings of VTS, pp 249–255
17. Miskov-Zivanov N, Marculescu D (2006) MARS-C: modeling and reduction of soft errors in combinational circuits. In: Proceedings of DAC, pp 767–772
18. Hachtel GD, Macii E, Pardo A, Somenzi F (1996) Markovian analysis of large finite state machines. IEEE Trans Comput Aided Des 15:1479–1493
19. Mohanram K, Touba NA (2003) Partial error masking to reduce soft error failure rate in logic circuits. In: Proceedings of DFT, pp 433–440
20. Alpert CJ, Devgan A, Kashyap C (2000) A two moment RC delay metric for performance optimization. In: Proceedings of ISPD, pp 69–74
21. Chu C, Wong YC (2005) Fast and accurate rectilinear steiner minimal tree algorithm for VLSI design. In: Proceedings of ISPD, pp 28–35
22. Caldwell A, Kahng A, Markov I (2000) Can recursive bisection alone produce routable placements?. In: Proceedings of DAC, pp 693–698
23. UMICH physical design tools. http://vlsicad.eecs.umich.edu/BK/PDtools/
24. ILOG Cplex: high-performance software for mathematical programming and optimization. http://www.ilog.com/products/cplex

Chapter 7
Summary and Extensions

This chapter summarizes the research presented in the preceding chapters, and suggests some possible ways in which it might be extended further.

7.1 Summary

We have attempted to provide an in-depth analysis of methods for representing and reasoning about logic circuits subject to uncertainty. We have proposed both exact and heuristic methods for error probability analysis and applied them to various design and testing problems. The main ideas presented may be summarized as follows:

- The development of the probabilistic-transfer matrix (PTM) algebra into a practical computation technique for computer-aided design.
- A reliability analyzer that uses a PTM-based framework to model logic circuits.
- Test-generation methods for probabilistic faults with the goal of estimating fault probabilities.
- A soft-error rate (SER) analyzer, AnSER, based on functional simulation with signatures.
- Logic synthesis techniques that improve SER with low area and performance overhead.

Due to changing device technologies, CMOS scaling affects, and other sources of noise vulnerability, a deterministic paradigm no longer fully captures circuit behavior. To this end, we introduced the PTM concept to provide a general framework for representing uncertainty. We showed that the PTMs can capture a wide variety of probabilistic effects in logic circuits. We then developed algorithms for efficiently compressing PTMs by means of arithmetic decision diagrams (ADDs) and operating directly on the compressed forms. The derivation of the circuit PTMs from gate PTMs is a form of exact probabilistic inference, which is known to be a very difficult problem in most settings, and our algorithms scale farther than other methods known in the literature.

S. Krishnaswamy et al., *Design, Analysis and Test of Logic Circuits Under Uncertainty*, Lecture Notes in Electrical Engineering 115, DOI: 10.1007/978-90-481-9644-9_7,

We applied the PTM-based formulation to automatic test pattern generation for probabilistic faults. The generalization of testing algorithms from a deterministic setting required a semantic change where the purpose of the testing is to determine a probability of error under certain conditions, rather than to directly reveal a defect. In particular, we proposed linear programming formulations to generate test multisets of minimal cardinality to reduce testing time. This is akin to test acceleration by pattern selection rather than by artificially induced radiation.

In addition to the exact PTM algorithms that we described, we also explored heuristic methods that support error-rate analysis in practical settings. An example is the AnSER method, which is based on the use of bit-parallel sampling methods and signatures. The latter are snapshots of truth tables from which behavioral properties like observability and signal probability can be computed. The efficiency and fast convergence of signature-based methods make it possible to incorporate error analysis into the design process.

Our development AnSER allowed us to design techniques that improve SER by enhancing error masking. An accurate SER analyzer can simply be used as a black box to approve or reject circuit optimizations. However, AnSER can do more than black-box analysis. Observability and node functionality computations performed during SER analysis make it possible to identify circuit flexibility and redundancy, which can be exploited to bolster circuit reliability. Further, error-latching window computations can guide physical design changes to improve both timing masking and logic masking.

7.2 Future Directions

There are many ways to extend our work to accommodate emerging issues in modeling stochastic behavior. Two of them are discussed next.

7.2.1 Process Variations and Aging Effects

There are three main sources of variation in circuit behavior that result from the difficulty of controlling transistor parameters during manufacture. These are: (1) dopant fluctuations, (2) gate length and wire width variations, and (3) varying heat flux across the chip [1]. First, the number of dopant atoms in the channel of a transistor decreases quadratically with transistor size. For instance, at the 1 μm technology node, transistors have thousands of dopant atoms [2]. At the 32 nm technology node, however, they only have tens of dopant atoms. Since the dopant level determines the threshold voltage V_t, even a few extra or missing dopant atoms can cause V_t to vary widely. Second, sub-wavelength lithography causes variations in gate length and wire width. Transistors at the 65 nm technology node are being manufactured with a 157 nm lithography wavelength, causing variations of up to 30 % [1].

Variations in wire widths normally increase delay, but variations in channel width can also change V_t. Third, changes in the switching activity of a chip can cause varying heat flux, and the resulting "hot spots" can lead to sudden changes in power-supply voltage and circuit performance.

While process variations have been analyzed for statistical timing analysis, there has been little effort in analyzing the impact of these variations on SER. In particular, changes in V_t alter SEU propagation in several ways. If V_t is increased, fewer SEUs will propagate, regardless of the cause [3]. If a lower dopant level causes higher V_t, it can sometimes lead to a higher rate of SEU occurrence (though lower rates of propagation), because of increased charge-collection efficiency for particle strikes. If the gate length is increased, then more chip area is exposed to particle strikes, causing more SEUs [4]. These effects often counterbalance each other—although not perfectly. Therefore, it becomes important to model them probabilistically in the context of SER. Also, delay variability, especially due to dynamic changes in heat flux, can cause differences in timing masking.

Interestingly, it may be possible to use accelerated SER tests to actually determine a circuit's static process parameters such as V_t and gate length. Since variations can directly affect the SER, irradiated chips can be used to induce changes in SER that reflect the process parameters. Then, a fault resolution method like that in Chap. 4, may be able to pinpoint locations with anomalous behavior.

After a device is deployed, wearout effects, such as oxide breakdown, negative temperature-bias instability, and electromigration can affect device reliability as well. These effects can eventually lead to hard failures in which a system outputs the wrong result with probability 1. In this book, we have analyzed the effects of probabilistic faults as transient phenomenon that can be modeled with static underlying discrete probability distributions. However, aging and wearout effects can change these distributions. Therefore, it is desirable to take lifetime reliability effects into account in forming dynamic models of uncertainty in logic circuits.

One advantage of incorporating both process variation and aging into models of system behavior is to be able to reason about error-tolerance mechanisms and how they impact the various sources of unreliable behavior. For instance, hard errors can often be corrected by reconfiguration using spare parts which are initially disabled. However, soft errors, as we have seen, may require online redundancy to correct errors during operation. Further, variability in manufacturing can lead to different specific errors being prevalent in a system. It may be necessary for future systems to introspectively decide and take measures to bolster reliability during their operational lifetime.

7.2.2 Analysis of Biological Systems

Currently, vast research efforts are being directed at modeling and designing biological systems using genes, transcription factors, and other types of proteins as computational elements. Biological systems are naturally uncertain and necessitate

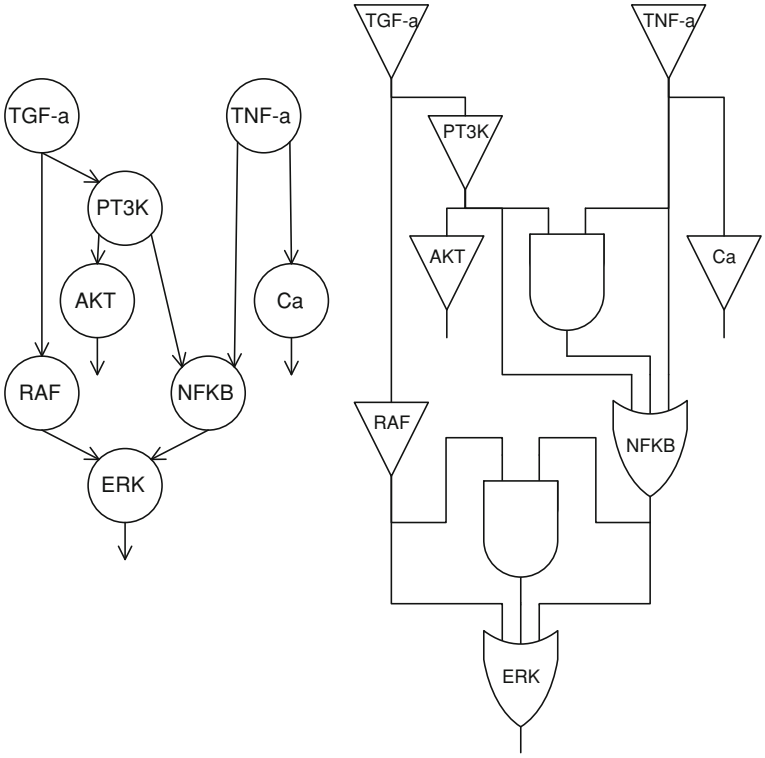


Fig. 7.1 Sample signaling network depicted as a protein interaction network (*left*), and a logic circuit (*right*) [5]

computational models that incorporate probabilistic behavior. We believe that in order to analyze biological systems as computational systems, and eventually to re-design and manipulate such systems, it will be necessary to model them using a probabilistic logic paradigm.

Biological systems are being modeled computationally in the emerging field of systems biology. An example of a biological system is a cell signal transduction network, in which signals are transmitted by particular biological molecules, such as hormones or cytokines which are sensed by surface receptors on cells. These signals are then passed through cascades of internal cell proteins that process and integrate the signals to eventually stimulate or inhibit the expression of specific genes. Alterations in gene expression amount to a reconfiguration of the system. The signals are generally passed from protein-to-protein through a specific chemical change known as *phosphorylation*. For our purposes, it is enough to think of a phosphorylated protein as "on" and a de-phosphorylated protein as "off".

A biological signal is inherently probabilistic in that it is carried by thousands of protein molecules, whose variations in concentration affect stochastic decisions made by cells. Since signals are carried by diffusing proteins, these proteins collide

with their targets with some probability dependent on their concentration. This is analogs to a relay race in sports where a baton (the signal) is passed between runners. Therefore, the signaling is both inherently probabilistic and time-varying. Moreover, it has been found useful to discretize the signals so they can be represented by Bayesian networks or even by Boolean logic [5, 6].

Figure 7.1 illustrates a simple signal transduction network of the kind found in many types of tissues. The nodes marked TGFB and TNF are receptors for growth factor signals, which are carried down to other proteins by means of successive protein phosphorylations. The topmost nodes marked TGF-a and TNF-a are cell-surface receptors, and act as primary inputs which sense the external cytokines. The nodes marked AKT, ERK, and NFKB are transcription factors, which act as primary outputs and directly affect gene transcription and cell reconfiguration. Since the actual underlying mechanisms by which a protein A encounters a protein B is random, the signal transduction itself is inherently probabilistic. Researchers have used Bayesian networks [6] and approximate logic models to represent this circuitry, which is internal to cells [5]. Clinical researchers are actively inventing drugs that alter this circuitry. Cancerous cells, for example, often have an altered signal transduction network which is repaired or disabled by drugs.

We believe that PTM models are well-suited for this type of representation because they capture many types of probabilistic behavior and avoid the rigid determinism of Boolean logic models. PTMs can thus form an intermediate representation between entirely free-form Bayesian networks and deterministic Boolean logic models. This can reduce the number of parameters to be learned in cases where gate behavior is inferred from large datasets, while at the same time faithfully representing the non-determinism of such systems.

7.3 Concluding Remarks

As we enter a new era of post-CMOS circuit design, computational paradigms must increasingly incorporate the uncertain behaviors that are displayed by emerging nano-, quantum-, and biological-device fabrics. A key part of this paradigm shift will be foregoing the illusion of perfect reliability and learning to tolerate and mitigate probabilistic errors.

This work has provided a broad framework for dealing with probabilistic behavior in logic design. It has also extensively explored ways to construct robust designs and testing methods in an uncertain world. We believe that the concepts presented here are extendable to many kinds of systems with vastly different modes of operation and sources of uncertainty. We hope that the book will inspire further research in the area of reliable design, and help to enable successful transitions to circuit technologies that are yet to come.

References

1. Borkar S (2005) Designing reliable systems from unreliable components: the challenges of transistor variability and degradation. IEEE Micro 25(6):10–16
2. Borkar S et al. (2003) Parameter variations and impact on circuits and microarchitecture. In: Proceedings of DAC, pp 328–342
3. Degalahal V et al (2004) The effect of threshold voltages on the soft error rate. In: Proceedings of ISQED, pp 503–508
4. Seifert N, Zhu X, Massengill LW (2002) Impact of scaling on soft-error rates in commercial microprocessors. IEEE Trans Nuclear Sci 49(6, part 1):3100–3106
5. Saez-Rodriguez J et al (2009) Discrete logic modelling as a means to link protein signalling networks with functional analysis of mammalian signal transduction. Mol Syst Biol 5:331
6. Sachs K et al (2005) Causal protein-signaling networks derived from multiparameter single-cell data. Science 308:523–529

Index

S. Krishnaswamy et al., *Design, Analysis and Test of Logic Circuits Under Uncertainty*, Lecture Notes in Electrical Engineering 115, DOI: 10.1007/978-90-481-9644-9,