Design, Analysis and Test of Logic Circuits under Uncertainty

by

Smita Krishnaswamy

A dissertation submitted in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
(Computer Science and Engineering)
in The University of Michigan
2008

Doctoral Committee:

Professor John P. Hayes, Co-Chair
Associate Professor Igor L. Markov, Co-Chair
Associate Professor Todd Austin
Associate Professor Dennis M. Sylvester
Assistant Professor Martin J. Strauss

To my family and friends

ACKNOWLEDGEMENTS

I have been blessed with numerous wise and caring people who have supported me through my Ph.D. and earlier education. I would like to thank my advisors John Hayes and Igor Markov for their advice in all aspects of research including conceptualization, sanity checking, technical writing, and presentation. Their feedback and unending patience were essential in helping me develop as a researcher. Financial support from the Air Force Research Laboratory and the National Science Foundation is also gratefully acknowledged.

I wish to thank all the professors I have worked with, either as a GSI or a student, including Marios Papaefthymiou, Bill Rounds, Kevin Compton, Alfred Hero, Alyce Brady, Tom Askew, and Eric Barth. I would also like to thank my secondary and primary school teachers, especially my math teachers Peggy Becker, Rosemary Brown, and Marcia Weinhold who encouraged me to keep thinking differently.

I wish to thank George Viamontes and Stephen Plaza with whom I collaborated extensively. Their insights and tireless efforts enriched our work. I also would like to thank all of the intelligent students I befriended at Michigan including Shaili Jain, David Papa, Jarrod Roy, Kai-Hui Chang, Ramashis Das, Jin Hu, Hector Garcia, Chien-Chih Yu, Sungsoon Cho, Dae Young Lee, and Ganesh Dasika. I gained a lot of ”megaknowledge”, technical insights, and practical skills from each of them.

I wish to thank my entire family who taught me to believe that I have the ability to achieve anything as long as I put my mind to it. I thank my grandmother Nagalakshmi

for all her love and support. I wish she were still here with us. I thank my parents Jyothi and Tipa who did everything possible to support me including paying for parking tickets, listening to complaints, and cooking for me weekly. I would not be able to do anything without them. Most people have two parents, I have about sixteen. I wish to thank these extra parents; my uncles Satyamama, Ramu, Papu, Raghu, Somu, Shekhar and Visweswar, and my aunts for always being there for me and telling me the right things to do. Way back in kindergarten, when I hated going to school, my uncle Shekhar would personally drop me off and say, "when you get your Ph.D. you'd better thank me for this." Twenty four years later, here it is. I thank my sister Esha who lived with me and brought joy and humor to my life. Finally, I thank my husband Ben for being supportive, rational, and distracting me at just the right moments with earth-shattering football news.

PREFACE

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

SER analysis must account for the three main error-masking mechanisms in ICs: logic, timing, and electrical masking. We observe that logic masking is closely related to node testability of the circuit. We use functional-simulation signatures, i.e., partial truth tables, to efficiently compute the 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 our 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.

In order to analyze probabilistic effects beyond soft errors, we develop a reliability analysis method that can evaluate circuits under a variety of fault assumptions. This method represents faulty gate behavior by means of stochastic matrices called probabilistic transfer matrices (PTMs). To improve computational efficiency, PTMs are, in turn, compressed into algebraic decision diagrams (ADDs). Several 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, any 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 integer linear programming (ILP) formulations to optimize our test sets for various objectives.

TABLE OF CONTENTS

LIST OF FIGURES

Figure

LIST OF TABLES

Table

CHAPTER I

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) [113] 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. These errors 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 [70] 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 mate-

rial, other sources of soft error soon became apparent. Later that year, Ziegler et al. [130] at IBM, showed that cosmic rays, consisting primarily of neutrons produced by cosmic rays from outer space, could also cause errors. The neutrons could strike 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. Additionally, new device technologies such as carbon nanotubes (CNTs), resonant tunneling diodes (RTDs), and quantum computers exhibit inherently probabilistic behavior due to 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 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 describe soft errors and 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. Finally, we state the goals of our research and outline the remaining chapters.

1.1 Background and Motivation

Soft errors are one of the main causes of uncertainty and failure in logic circuits [114]. Current trends in circuit technology are exacerbating 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 and molecular computing.

1.1.1 Soft Errors

A soft error is a signal that has an incorrect logic value but does not imply a permanent defect. Soft errors can be caused by cosmic rays, α-particles, and even thermal noise. Cosmic rays are particles that originate in space, usually 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 [129]. When primary cosmic particles enter the atmosphere, they can create a shower of secondary and tertiary particles, as shown in Figure 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 32nm, the particle energy required to upset the state of registers and memory circuits becomes smaller [41]. Figure 1.2 shows 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

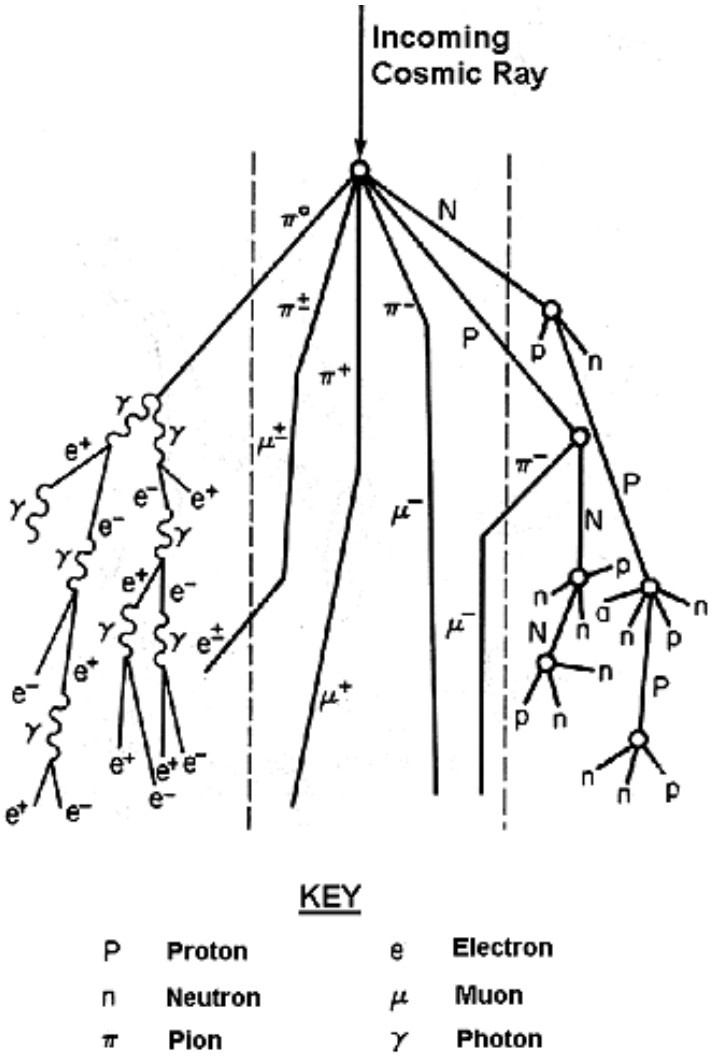


Figure 1.1: Shower of error-inducing particles caused by a primary particle in the atmosphere [129].

of particles with sufficient energy to cause errors increases rapidly [106]. For instance, even the lead in solder balls or trace amounts of radioactive contaminants in tin can affect CMOS circuits at lower energies [40].

When a particle actually strikes a 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 Figure 1.3. An SEU is a transient, or soft, fault as opposed to a permanent fault. The effects of an SEU do not propagate if the charge deposited is below the critical charge Q_{crit} required to switch the corresponding transistor on or off [114]. 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

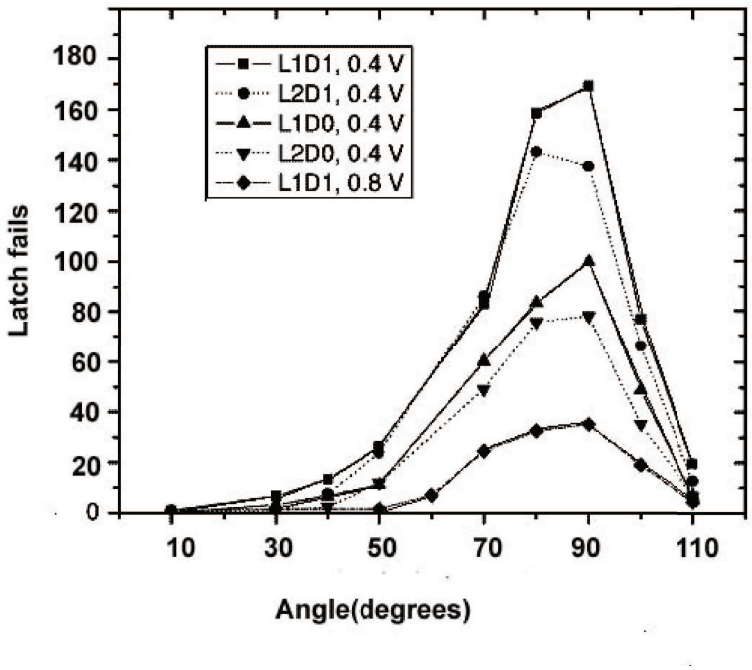


Figure 1.2: Latch failure for 1.25MeV proton strikes, as a function of the angle of incidence [106].

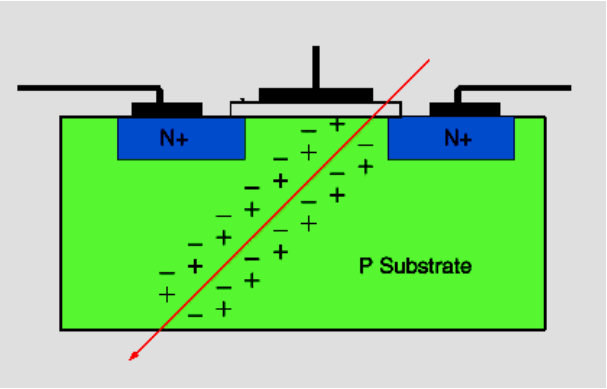


Figure 1.3: Ionized track in a transistor, caused by cosmic radiation [34].

analyze and decrease the soft-error rate (SER) of a circuit through careful design. This is especially true of circuits in mission-critical applications, such as servers and aircraft and medical devices.

1.1.2 Technology Trends

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 Figure 1.4. In order to facilitate this growth, chip features have become smaller, alongside 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 for further reductions in power consumption. Keyes and Landauer [47] 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 given by CV^2, where V is the supply voltage and C is the capacitance of the gate given by $C = WC_{out} + \sum_{fanout} C_{in}W_j + C_L$. 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 the transistor. Therefore, as W and V decrease, the switching energy approaches $KT\ln 2$, causing logic gates to become more susceptible to noise.

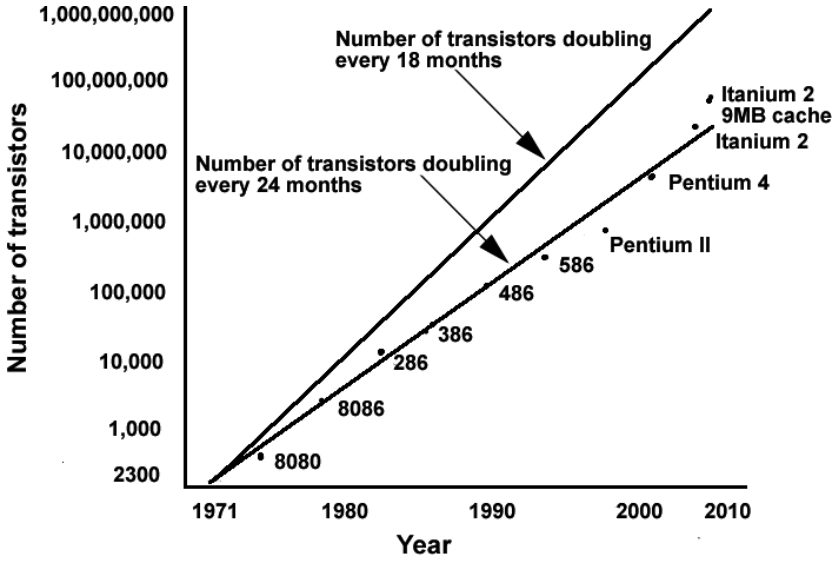


Figure 1.4: Moore's law, showing IC density increase per year.

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 [44]. Processors with 40MHz operating frequency were tested, and 400MHz processors were simulated. The results, shown in Figure 1.5, 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.

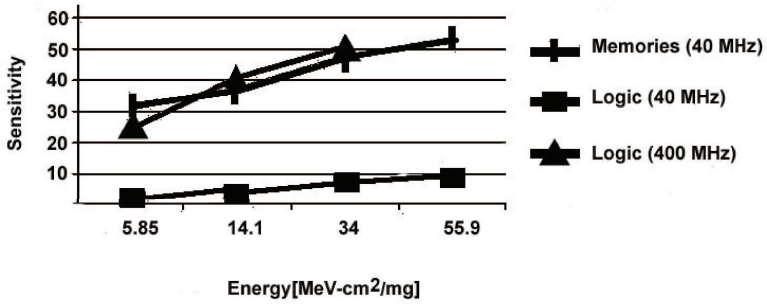


Figure 1.5: Memory and logic sensitivity to soft errors in the RISC processor [44].

IC technologies beyond CMOS are expected to exhibit even more probabilistic behavior. Examples of new device technologies under active investigation include carbon nanotube transistors (CNTs), resonant-tunneling diodes (RTDs), quantum cellular automata (QCA), and various quantum computing technologies, like ion traps that handle quantum bits (qubits). CNTs and RTDs experience high error probabilities because they operate near the thermal limit of $KT\ln 2$ [69, 14]. QCAs 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 [62, 104]. Quantum computing devices are inherently probabilistic (even during fault-free operation) because qubits exist in superposition states and collapse to either 0 or 1, with different probabilities upon measurement.

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, they can often be modeled probabilistically. Figure 1.6 illustrates the lithography wavelengths associated with smaller IC feature sizes by year. As the gap between the

wavelength and feature sizes continues to widen, it becomes difficult for 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 speed up signal transitions and sometimes causes glitches that resembles SEUs to appear [96]. 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 [15]. These variations can cause inherent uncertainty in circuit behavior.

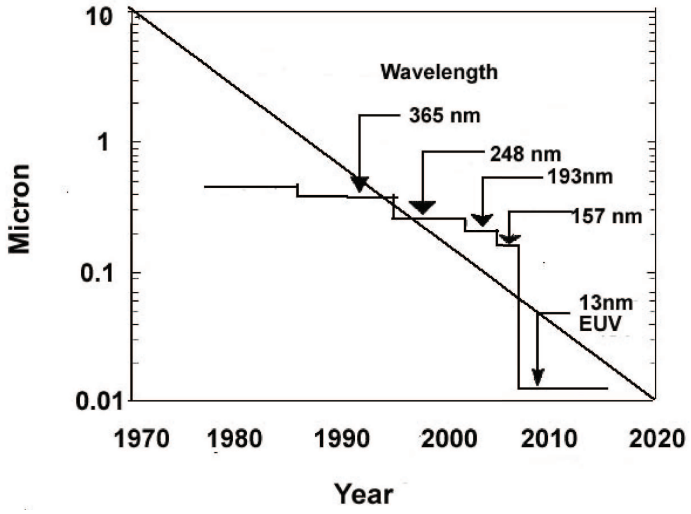


Figure 1.6: Feature-size trends in ICs by year [34].

1.2 Related Work

In this section, we discuss related work in soft-error rate (SER) analysis, fault-tolerance techniques, soft-error testing, and probabilistic circuit analysis.

1.2.1 Soft-Error Rate Analysis

We first introduce the problem of SER estimation and discuss solutions that appear in the literature, often alongside our work. The aim here is to reveal the intuition behind SER analysis methods and to motivate techniques introduced in later chapters.

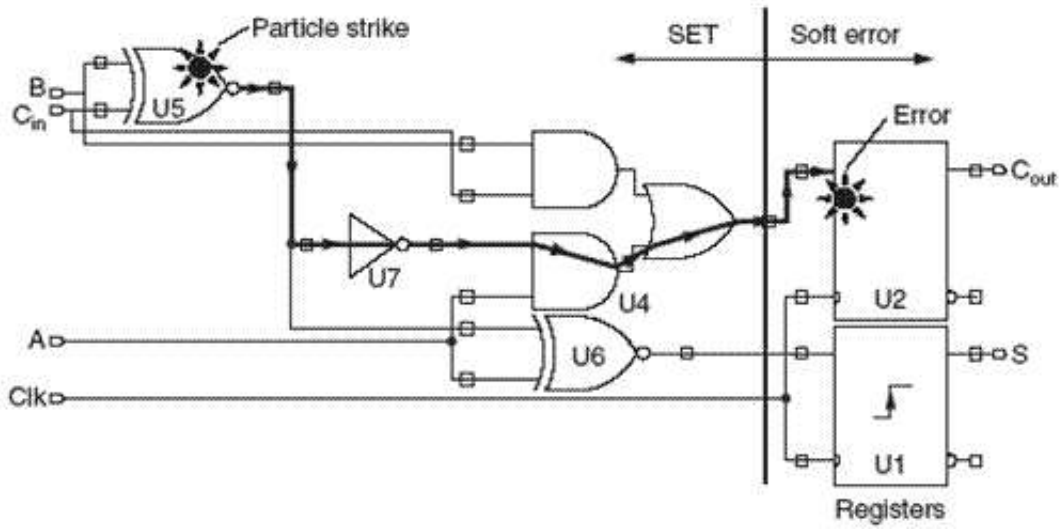


Figure 1.7: Illustration of transient-fault propagation in combinational logic.

There are several factors to consider in determining the SER of a logic circuit. Figure 1.7 illustrates the three main conditions that are required for an SEU to be latched, and these conditions are explained below.

- The SEU must have sufficient energy to change a signal and propagate the erroneous signal value through subsequent gates. If not, the fault is electrically masked.
- The change in a signal's value must be propagated through the logic to 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, nor-

mally 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 the 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 strike per clock cycle, the SER can be computed using the brute-force algorithm given in Figure 1.9. 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,
- $P_{strike}(n)$, the probability of a fault at location n,
- $P_{attenuate}(path(p))$, the probability of attenuation along path p, and
- $P_{latch}(p, o)$, the probability of an error arriving on path p at output o during a latching phase of a clock cycle.

Since the four values are probabilities, neglecting to model any of these factors leads to overestimation of the SER. Figure 1.8 shows an example of an SEU in the ISCAS-85 circuit C17, along with logically sensitized paths for different input vectors.

The algorithm of Figure 1.9 is only practical for the smallest of circuits. The number of input vectors to a circuit is exponential in the number of inputs, and the number of sensitized paths can grow exponentially in the size of the circuit [99]. Therefore, even

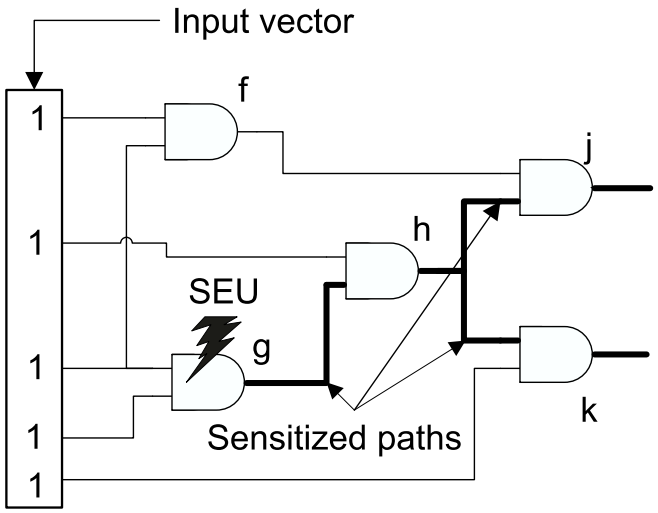


Figure 1.8: Illustration of logically sensitized paths (in heavy lines) for error propagation with respect to a specific input vector.

```
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)
}
```

Figure 1.9: Basic SER computation algorithm.

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 shown to approximate the SER for combinational circuits. Below, we describe three of these tools and their SER computation techniques [134, 135, 100]. Of the three algorithms, SERA is closest to that of Figure 1.9. SERA 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 is 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, FASER [135] uses binary decision diagrams (BDDs) to enumerate all possible input vectors. A BDD is created for each gate in a circuit—a static BDD for gates outside the fan-out cone of the glitch location, and duration and amplitude BDDs for gates in the fan-out 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 according to FASER's estimation of electrical masking. Due to complete input-vector enumeration, FASER's BDD representations can require a lot of memory for many 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 [100] 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 each 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 on to 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.

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

Table 1.1: Summary of differences between three SER evaluation tools.

Our work aims to build SER analysis tools that are scalable and can be used early in the logic design phase [58, 59, 55]. Due to our emphasis on reliability-driven logic design, we focus on modeling logical masking both accurately and efficiently. We then use our tools to guide several design techniques to improve circuit resilience against soft errors.

1.2.2 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 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) [82, 107] is another common method of error detection at the architectural level. RMT refers to running multiple threads of the same program and comparing results. The first thread, known as the leading thread, often executes ahead of other threads to allow time for transient glitches to dissipate.

The DIVA method [7, 126] 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 given by 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 [97, 6]. The main masking mechanism in 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, as computed by statistical fault injection or other forms of performance analysis [80].

At the logic level, designers have complete information about the function of the 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 [125], 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. CTMR contains TMR units that are, in turn, replicated thrice, and this

process is repeated until the required reliability is reached.

In NAND multiplexing, 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, is used as a simple majority function. Some of these NAND gates 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 [76] propose to partially triplicate logic by selecting regions of the circuit that are especially susceptible to soft errors. Such regions are selected by simulating faults with random test vectors. Dominant-value reduction [76] is also used to duplicate, rather than triplicate, selected logic. Dominant-value reduction 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. [3] used a design modification technique, called rewiring, to increase reliability. In the spirit of [3], our work focuses on lightweight modifications to the circuit that increase reliability without requiring significant amounts of redundancy. These types of modifications will be discussed further in Chapter III.

At the transistor level, gates can be electrically characterized, and electrical masking can be used as an error-mitigation mechanism. Gate sizing is a common technique for increasing electrical masking: increasing the area of a gate increases its internal ca-

pacitance and, therefore, the critical charge Q_{crit} necessary for a particle strike to alter a signal. However, this technique increases 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. [9] and, later, by Zhang et al. [132]. Dual-port gates, illustrated in Figure 1.13, decrease charge-collection efficiency, using two extra transistors placed in a separate well from the original transistors.

In the 1990s, Nicolaidis proposed another method of increasing electrical masking [87]. In this method, three latches sample a signal with small delays between, and a voter is used to decide the correct value of the signal. Since stray glitches tend to have short durations, the erroneous value induced by a glitch is likely to be sampled by only one of the three latches. Razor [33] uses this idea for dynamic voltage scaling, sampling signals twice and, when an error is found, restoring program correctness via a detect-and-playback scheme. The recently-proposed BISER [131] 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. See Figure 1.14 for the BISER flip-flop design.

Finally, after the placement and routing of a circuit are completed, gate and interconnect delays can be determined. In earlier IC technology, timing was usually analyzed at the gate level, since wire delay contributed only a small (often negligible) fraction of the

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

Table 1.2: Summary of error-masking mechanisms and fault-tolerance techniques at various levels of abstraction.

critical path delay. However, in current technology, wire delay dominates gate delay and needs to be incorporated into any accurate timing analyzer. Once we can analyze the timing, we can also obtain information about timing masking [114, 74]. To date, very 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, 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 indicates that fine-grained, accurate SER analysis at low levels is computationally feasible and decreases overhead [58, 59, 55]. Table 1.2 summarizes the fault-tolerance techniques and masking mechanisms discussed in this section.

1.2.3 Soft-Error Testing

Chip manufacturers including IBM, Intel, and Toshiba, as well as medical equipment manufacturers like Medtronics, routinely test their chips for SER [129, 70, 50, 127]. 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 [130], 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.

There has been some difficulty, however, in translating the SER obtained by accelerated testing to that of field testing [50]. 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 vast number of devices and dedicated testers for each device, Polian et al.[39] have proposed a non-concurrent built-in self-test (BIST) architecture for online testing. They define the impact of various soft faults on the 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 for various objectives, such as ensuring a minimum fault-detection probability.

Recently, researchers have sought to accelerate testing by selecting test patterns that sensitize faults. Conceptually, the main difference between testing for hard errors versus soft errors is that soft errors are only present for a fraction of the test time. Therefore, test vectors must be repeated to detect faults, and they must be selected to sensitize the most frequent faults. Sanyal et al. [108] 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 our work, which preceded [108], we developed a way of identifying error-sensitive test

vectors for multiple faults, which could include other masking mechanisms like electrical masking, and we devised algorithms for generating test sets to accelerate SER testing [54, 53].

1.2.4 Probabilistic Analysis of Circuits

Soft errors and new device technologies are projected to make circuit behavior generally more uncertain. Therefore, circuit design and testing require new types of probabilistic analysis that goes beyond soft error analysis only. In this section, we provide background on the probabilistic analysis of logic circuits, using Bayesian networks and Markov random fields.

In our work, we develop a novel probabilistic matrix-based model for gates, and we use matrix operations and symbolic methods to evaluate overall circuit error probabilities [60, 61]. More recently, Rejimon et al. [104] 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

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

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. The probability distribution of x_i can be given in tabular form or by specifying a known distribution. Certain nodes (such as

those corresponding to primary inputs) are given pre-defined probabilities. The probabilities of other nodes are then computed using a method called belief propagation. Joint probabilities are computed in large Bayesian networks using sampling methods such as importance sampling. Many tools [35, 89] exist for Bayesian network analysis.

Bahar et al.[14] propose 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 [14], 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 [14]. Related to this, Nepal et al. [86] present a method for implementing MRF-based circuits in CMOS, and Bhadhuri et al. [12] describe a software tool, known as Nanolab, which uses the algorithm from [14] to automate the design of fault-tolerant architectures, like CTMR, in nanotechnologies.

1.3 Thesis Outline

In this dissertation, we focus on gate-level SER analysis, probabilistic circuit analysis, and fault-tolerant design. We carefully study the input-vector dependence in logic, as well as timing masking, in order to design circuits with better reliability. We further develop methods to model inherently probabilistic methods in logic circuits and to test circuits for determining their reliability after they are manufactured. Our main goals are:

- To develop scalable and accurate methods of SER and susceptibility analysis, usable

during the CAD flow at the gate level,

- To devise methods that guide logic design towards greater resilience against soft errors,

- To develop general and accurate methods for modeling and reasoning about probabilistic behavior in logic circuits, and

- To develop test methods for accurately and efficiently measuring soft-error susceptibility in circuits after they are manufactured.

The remainder of this dissertation is organized as follows. Chapter II presents an efficient technique to analyze SER at the logic level. Here, we formulate probabilistic fault models, based on the stuck-at model used in the testing literature. We propose ways to account for the three basic masking mechanisms, using 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. Our analysis methods are also extended to sequential circuits.

In Chapter III, we apply the analysis techniques from the previous chapter to the design of reliable circuits. Our techniques include logic rewriting, gate hardening, and a novel technique we call SiDeR. This technique uses functional relationships between 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 IV presents a general matrix-based reliability analysis technique, the probabilistic transfer matrix (PTM) framework, to model faulty gate behavior. PTMs form an algebra for reasoning about uncertain behavior in logic circuits. The algebra includes 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 introduced to derive information about circuit reliability and output error probabilities, under various types of faults.

Chapter V 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 VI 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 the test-vector set.

Finally, Chapter VII summarizes our work and discusses possible directions for future research.

```
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)
}
```

(a)

```
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)
}
```

(b)

```
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)
}
```

(c)

Figure 1.10: Algorithms for SER computation used by (a) SERA, (b) FASER, and (c) SET.

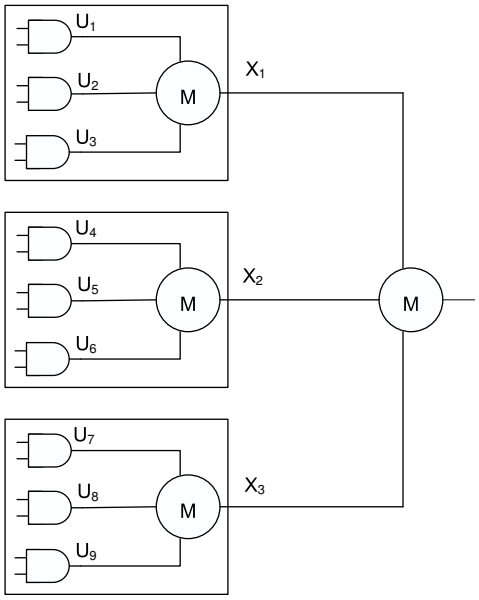


Figure 1.11: The cascaded TMR scheme; *M* denotes a Majority gate [125].

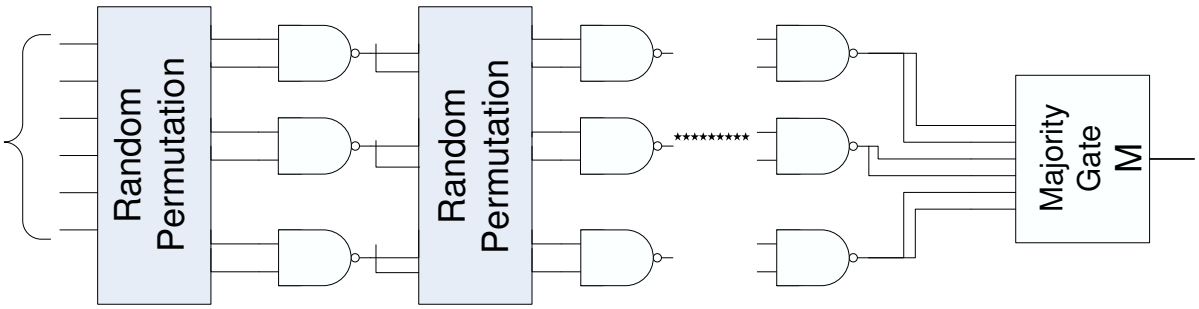
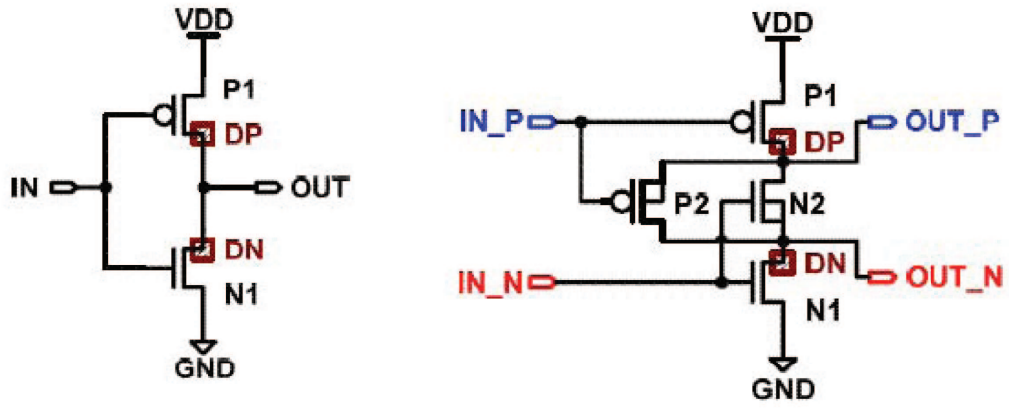


Figure 1.12: The NAND-multiplexing scheme [125].



(a) (b)

Figure 1.13: (a) Normal and (b) dual-port CMOS inverter with two additional transistors in an isolated well [9].

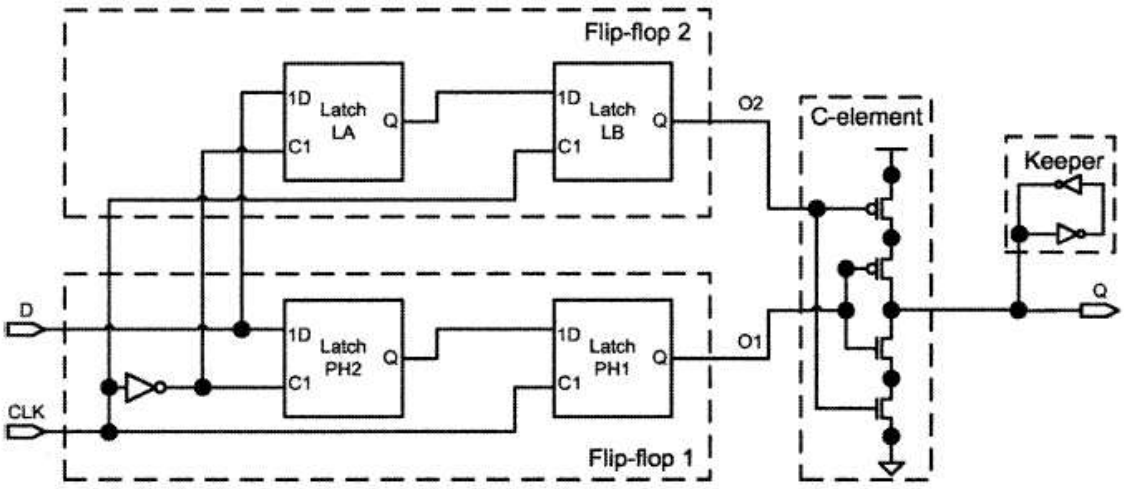


Figure 1.14: The error-correcting BISER flip-flop design with a *C*-element and a keeper circuit [131].

CHAPTER II

Signature-based Soft-error Analysis

As soft errors become increasingly prevalent in logic circuits, soft-error rate (SER) prediction becomes important in all phases of design. As discussed in Section 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. Therefore, it is important to efficiently and accurately analyze the SER during the actual design process.

This chapter presents the SER analyzer called AnSER. AnSER employs functional-simulation signatures extensively in order to estimate logic masking and to account for the input-vector dependence in timing and electrical masking. Signatures provide an efficient way of computing testability measures like signal probability and observability, which are, in turn, 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 of a test vector for the fault being applied at the

primary inputs. 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 and efficient signature-based algorithm.

Figure 2.1 illustrates the flow of computation in AnSER. Functional-simulation signatures are computed from logical information, error-derating factors from gate-characterization information, and error-latching windows from static-timing analysis. These smaller computations are combined to form an estimate of circuit SER. Since AnSER is intended to be used alongside logical and physical design tools, we pay particular attention to runtime, memory requirements and the incremental-use model. Figure 2.1 also shows how AnSER can be incorporated into a typical RTL-to-GDSII flow through incremental calls after each change to the netlist or placement.

The remainder of this chapter is organized as follows. Section 2.1 develops our method for computing the SER of logic circuits by accounting for logic masking. Section 2.2 extends this methodology to sequential circuits. Finally, Section 2.3 incorporates timing and electrical masking into our SER estimates. Most of the techniques and results presented in this chapter also appear in [58, 55, 59, 57].

2.1 SER in Combinational Logic

In this section, we present an SER analysis method for combinational logic which, by definition, contains no memory. We first develop fault models for soft errors. Then, we provide background on functional-simulation signatures, which we use extensively in

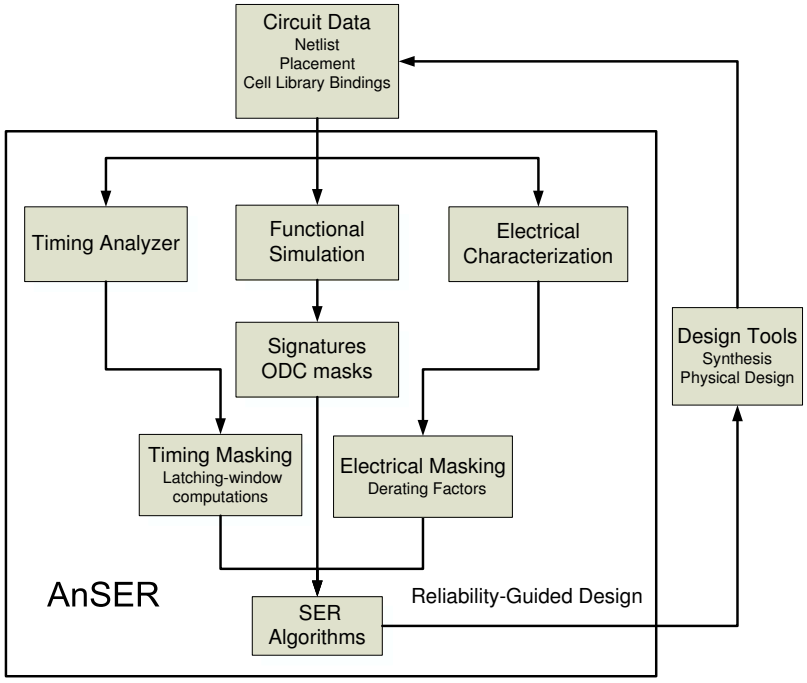


Figure 2.1: Computational flow of AnSER.

AnSER. Next, we derive SER algorithms for single- and multiple-fault assumptions using signal probability and observability measures that are computed using signatures. Finally, we show how to account for electrical and timing masking.

2.1.1 Fault Models for Soft Errors

For the purposes of logic-level reasoning, we formulate a model for single transient faults with extensions to account for multiple faults. 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 show in Chapter VI, can be useful for testing. However, in this section their primary use is in SER analysis; the close connections between testability and SER facilitate this use.

We conceptualize external noise (such as an SEU) as a probabilistic fault. 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 temporary single stuck-at-1 (TSA-1) fault with probability a $Perr_1(g)$, and a temporary single stuck-at-0 (TSA-0) fault with probability $Perr_0(g)$.

Definition 1 *A* transient stuck-at (TSA) *fault 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 are correlated if the occurrence of one fault changes the probability of another fault. An example is a multiple-bit upset where a single particle strike causes multiple

upsets in nearby gates. Such upsets are spatially correlated.

Definition 2 *A* transient multiple-correlated stuck-at fault (TMCSA) *is the triple* $(G, V, Perr)$ *where G is a set of node* $\{g_1, g_2, g_3, \ldots\}$*, 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 3 *A* transient multiple stuck-at fault (TMSA) *fault 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 TMSA 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 mainly 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 seconds. If the soft-error probability per cycle is p, then the expected number of failures per 10^9 seconds is simply $p \times freq \times 10^9$, 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. In general, we denote probabilities of error *Perr* and gate SER as *gerr*.

2.1.2 Signatures and Observability Don't-Cares

We systematically use node 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 4 *A* signature*, denoted,* $sig(g) = F_g(X_1)F_g(X_2)\ldots F_g(X_K)$ *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 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 [137, 92]. A related speedup technique is also the basis for "parallel" fault simulation [19]. The basic algorithm for computing signatures is shown for reference in Figure 2.2. Here, $Op < g >$ refers to the operation gate g. This operation is applied to the signatures of the input nodes of gate g, denoted $inputsigs(g)$.

```
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))
}
```

Figure 2.2: Basic algorithm for signature computation.

Figure 2.3 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 vector simulations are conducted in parallel with each signature processed. Generating K-bit signatures in an N-node circuit takes $O(NK)$ time.

Observability don't-cares (ODCs) occur at a node g for input vectors for which the value at g does not affect the primary outputs. For example, in the circuit $\mathrm{AND}(a, \mathrm{OR}(a,b))$, the output of the OR gate is inconsequential when $a = 0$. Hence, input vectors 00 and 01 are ODCs for b.

Definition 5 *Corresponding to the K-bit signature $sig(g)$, the* ODC mask *of g, denoted $ODCmask(g)$, is the K-bit sequence whose i^{th} bit is* 0 *if input vector X_i is in the don't-care set of g; otherwise the i^{th} bit is* 1*, i.e., $ODCmask(g) = X_1 \notin ODC(F_g) X_2 \notin ODC(F_g) \ldots X_K \notin ODC(F_g)$.*

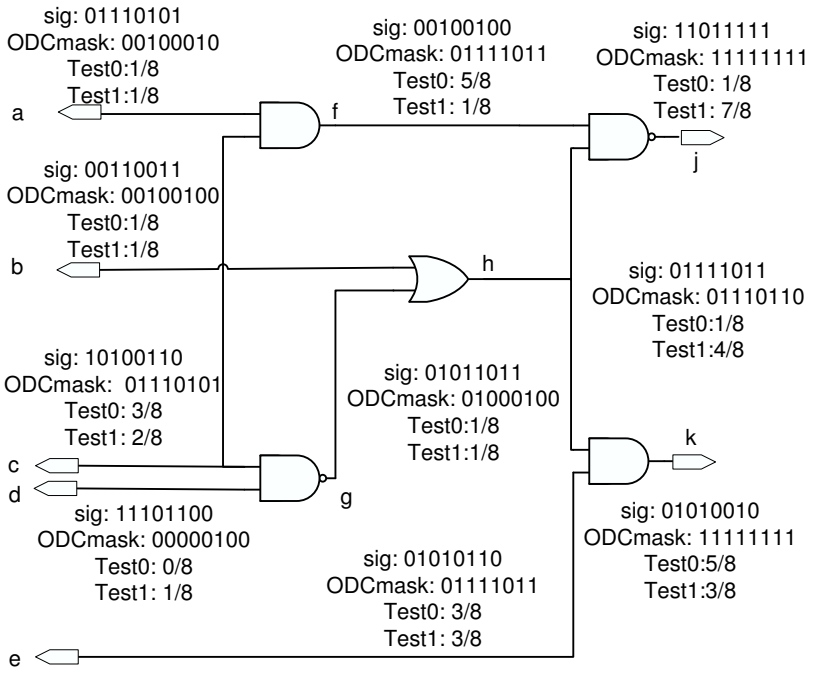


Figure 2.3: Signatures, ODC masks, and testability information associated with circuit nodes.

The ODC mask is computed by bitwise inverting $sig(g)$ and re-simulating 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 Figure 2.4a 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.

We found that the heuristic algorithm for ODC mask computation presented in [92], which has only $O(N)$ complexity, particularly convenient to use. This algorithm, shown in Figure 2.4b, 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

```
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)

}
```

(a)

```
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)
}
```

(b)

Figure 2.4: (a) Exact and (b) approximate ODC mask computation algorithms.

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 1 *Figure 2.3 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 ODC mask of c by ORing the ODC masks for paths through* f *and g, which yields* 01110101*.*

2.1.3 SER Evaluation

We compute the SER by counting the number of test vectors that propagate the effects of a transient fault to the output(s). Test-vector counting was also used in [39] to compute SER, although the algorithm there uses BDD-based techniques. Intuitively, if a large number of 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 faults on a signal.

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 [19].

Figure 2.5 summarizes our algorithm for SER computation. It involves two topological traversals of the target circuit: one to propagate signatures forward and another to

```
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 2.5: Algorithm to compute SER under the TSA fault model.

propagate ODC masks backwards. The fraction of 1s in a node's signature is 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 6 *The* controllability *of a signal g, denoted* $P[g=1]$*, is the probability that g has logic value* 1.

$$P[g=1] = ones\big(sig(g)\big)/K \tag{2.1}$$

Definition 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.

$$P[obs(g)] = ones\big(ODCmask(g)\big)/K \tag{2.2}$$

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

$$P[test_1(g)] = \text{ones}\big(sig(g)\&ODCmask(g)\big)/K \tag{2.3}$$

Similarly, 0-testability is 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 2 *Consider again the circuit in Figure 2.3. 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.

$$Perr(C) = \sum_{g \in C} P[test_1(g)]Perr_0(g) + P[test_0(g)]Perr_1(g) \tag{2.4}$$

Example 3 *The $test_0$ and $test_1$ measures for each gate in the circuit are given in Figure 2.3. If each gate has TSA-1 probability $Perr_0 = p$ and TSA-0 probability $Perr_1 = q$, then the SER is given by $Perr(C) = 2p + (13/8)q$.*

The metrics $test_0$ and $test_1$ implicitly incorporate error sensitization and propagation conditions. Hence, Equation 2.4 accounts for the possibility of an error being logically masked. Note that the $Perr_1(g)$ refers to the 1-controllability of g and so is weighted by the 0-testability, similarly for $Perr_1(g)$.

2.1.4 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 the ODC mask for each node 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 be applied to a set of nodes rather than a single node.

Recall that exact ODC computation of Figure 2.4a requires resimulation of the entire fan-out cone of the fault location, while the algorithm from [92], shown in Figure 2.4b,

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 G. We then bitwise invert the signatures of all the nodes in the 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 Figure 2.6. 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 Figure 2.7. 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.

```
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
}
```

Figure 2.6: Algorithm to compute 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
}
```

Figure 2.7: Algorithm to compute SER under the TMSA fault model.

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.

2.2 SER Analysis in Sequential Logic

In this section, 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 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 [36]. 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 subsections develop a simulation-based framework to address these issues.

2.2.1 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. A finite state machine (FSM) is periodic if its states can be visited only at regular intervals, and, otherwise, is aperiodic. Periodic FSMs do not reach steady state. Hachtel et al. [36, 23] show that aperiodic FSMs with strongly-connected state spaces eventually reach a steady-state distribution. A modulo-d counter is an example of such an FSM. In [23], it is shown that most ISCAS 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 benchmarks reach steady-state in 10 cycles or fewer, under the above operating conditions [75].

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 2.8 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 [36, 75]. For example, Markov matrices are used to encode state-transition probabilities explicitly, and therefore, can be large due to the problem of state-space explosion [36, 75].

Figure 2.9 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 input of x and y are transferred to the output, and the process continues. At the conclusion of the simulation, the values for x and y at T_3 are

```
simulate_sequential(Circuit C, int K)
{
 for(all flip-flops l ∈ C)
  output sig(l) = input sig(l)
 for(all inputs in_0 ∈ C)
  in_0 = new_random_input()
 compute_sigs(C,K)
}
```

Figure 2.8: Algorithm for multi-cycle sequential-circuit simulation.

saved for sequential-error analysis, which is explained in the next subsection.

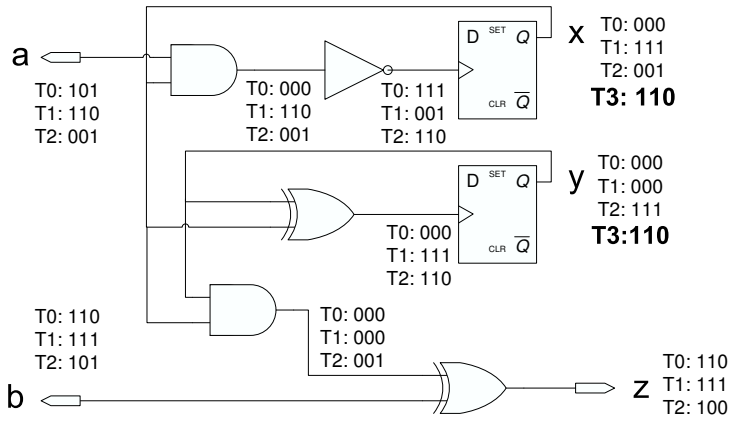


Figure 2.9: Illustration of bit-parallel sequential simulation.

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.

2.2.2 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 circuit. 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 $k+1$ frame (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. Figure 2.10 shows a three-time-frame circuit that corresponds to Figure 2.9. 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.

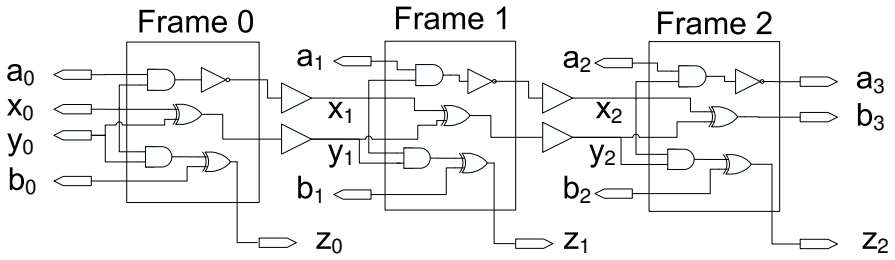


Figure 2.10: Illustration of time-frame expansion into three frames: C_0, C_1, C_2.

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 multi-cycle 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 in-

stance, the x_0 and y_0 inputs of the circuit in Figure 2.10 are simulated with the corresponding signatures, marked T_3 (the final signature after multi-cycle simulation is finished), from Figure 2.9. Randomly generated signatures are used for primary inputs not corresponding to flip-flops (such as a_0 and b_0 in Figure 2.10).

After 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. Figure 2.11 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.

$$P[test_0(g,f,n)] = zeros(sig(g,f,n)\&ODCmask(g,f,n))/K \tag{2.5}$$

$$P[seqobs(g,f,n)] = \text{ones}\big(ODCmask(g,f,n)\big)/K \tag{2.6}$$

$$Perr(C_0,f,n) = \sum_{g_i \in C_0} P[test_1(g_i,f,n)]Perr_0(g_i) + P[test_0(g_i,f,n)]Perr_1(g_i) \tag{2.7}$$

```
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')
}
```

Figure 2.11: Algorithm to compute SER in sequential circuits under TSA faults.

The SER algorithm in Figure 2.11 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.

2.3 Additional Masking Mechanisms

Due to various physical and electrical properties of circuits, phenomena other than logic masking can stop error propagation in certain cases. While these factors cannot be assessed during logic synthesis, they can be considered during technology mapping and physical design and can serve to guide these processes. It should be noted that timing and electrical masking are expected to diminish in strength as technology scales. Roughly speaking, flip-flops can latch SEUs every time the latching clock-edge is triggered. Therefore, an increase in operating frequency increases the frequency of latching opportunities [111]. Additionally, as power-supply voltage decreases and gate sizes shrink, fewer SEUs are expected to be attenuated.

To capture electrical masking in AnSER, we derate gate-error probabilities ($gerr0, gerr1$) by a factor dependent upon characterization of successor gates. Previous research has shown that electrical masking eliminates low-energy SEUs in 3-4 levels of logic and has little effect thereafter [100]. This implies that considering paths of limited length starting from the gate in question is often sufficient to approximate this effect. In the remainder of this section, we develop a linear-time algorithm in the spirit of static timing analysis (STA) for computing the error-latching window (ELW) of each gate in a circuit. The size of the ELW relative to the clock period is an estimate of the circuit's ability to mask the error through timing properties. The input-vector dependence of timing masking is incorporated into our estimates by functional simulation. We also briefly discuss the incorporation of electrical masking through derating factors that are used to scale the intrinsic error probability of a gate. We assume sequential circuits with edge-triggered flip-flops separated by combinational logic blocks.

2.3.1 Static Analysis of Timing Masking

The timing constraints associated with each edge-triggered D flip-flop are as follows:

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

A soft error is 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

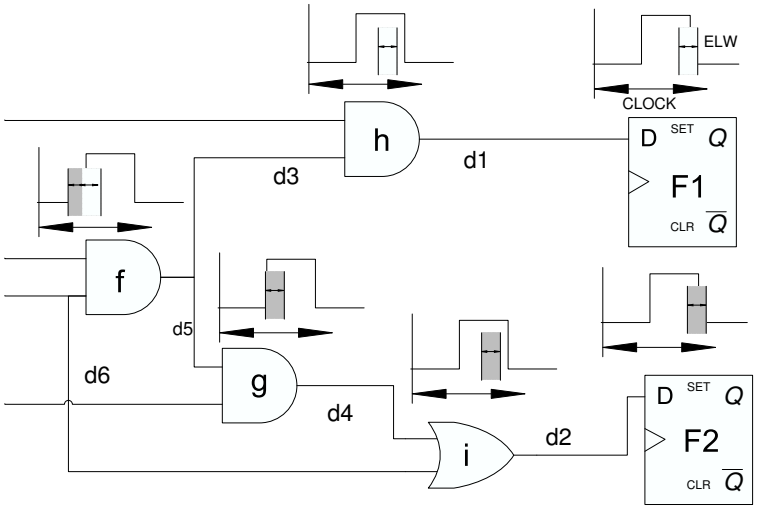


Figure 2.12: Error-latching windows illustrated.

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 fan-out gates backwards by appropriate path delay and 2) taking the union of the resulting ELWs. In contrast, during static timing analysis, only the minimum required time is computed at each gate, even though a similar backwards traversal is used. Figure 2.13 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 $[S_{gi}, E_{gi}]$.

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

```
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 fan-outs f)
     ELW'(f) = translate(ELW(f), delay(l, f))
    ELW(l) = union(ELW(l), ELW'(f))
}
```

Figure 2.13: Computing error-latching windows (ELWs).

Example 4 *Our proposed ELW computation algorithm is illustrated by the circuit in Figure 2.12. Each wire is marked with a delay, and each gate i is assumed to have delay $d(i)$. The corresponding ELWs are shown below. Note that f has a larger ELW than other gates because its two output paths have different delays.*

$ELW(F1) = ELW(F_2) = [T_s, T_h]$

$ELW(i) = [T_s - d_2, T_h - d_2]$

$ELW(g) = [T_s - d_2 - d(i) - d_4, T_h - d_2 - d(i) - d_4]$

$ELW(h) = [T_s - d_1, T_h - d_1]$

$ELW(f) = [T_s - d_2 - d(i) - d_4 - d(g) - d_5, T_h - d_1 - d(h) - d_3]$

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:

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

Taking timing masking into account, the SER contribution of each gate is computed by scaling the testability and error probability by T_{mask}.

$$SER(C) = \sum_{g \in C} \big(P[test_1(g)]Perr_0(g) + P[test_0(g)]Perr_1(g)\big) T_{mask}(g) \tag{2.8}$$

```
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])
}
```

Figure 2.14: Computing the union of two ELWs.

2.3.2 Statistical-Interval Weighting

The latching windows computed in the previous section were a result 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 latches. In order to compute such a probability, we use bit-parallel logic simulation. Recall that the ones-count of the signature of a node is a measure of signal probability, and the one-count of the ODC mask is 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. For such a

fault, an entire path, rather than a single stuck-at fault, is sensitized. For our purposes, we consider sets of paths associated with each ELW interval, rather than single paths. Therefore, we associate an interval ODC mask $intODCmask(f,i)$ with each interval i in an $ELW(f)$.

Definition 9 *The interval ODC mask of g, denoted* $intODCmask(g)$ *is the K-bit sequence whose ith bit is* 0 *if input vector* X_i *is in the don't-care set of g during interval i; otherwise, the ith bit is* 1.

The one-count of the interval ODC mask is the interval weight.

We compute interval ODC masks in reverse topological order, along with ELWs. We initially consider gates that feed primary outputs; for such a gate g, all interval ODC masks are simply equal to $ODCmask(g)$, i.e., for all intervals i in $ELW(g)$, $intODCmask(g,i) = ODCmask(g)$. For subsequent gates, ELWs are computed by translating and merging the ELWs of successor gates. Here, each interval ODC mask associated with a successor gate is ANDed with the ODC mask of the current gate. Intuitively, the interval ODC mask keeps track of the observability of a signal along a specific path. Therefore, ANDing the ODC corresponding to a path by the ODC of the additional gate simply adds that gate to the path.

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

Suppose that a gate f has fan-out branches g and h and that during ELW computation, intervals $ELW(g)[i]$ and $ELW(h)[j]$ are merged together to form $ELW(f)[k]$. In this case,

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

The SER computation, including timing masking, is simply the sum of interval testabilities weighted by the lengths of the corresponding intervals. The testabilities are in turn derived using the interval ODC and signal probabilities. These computations are shown below.

$$P[test1(f,i)] = \text{ones}\big(sig(f)\&ODCmask(f,i)\big)/K$$

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

$$SER(C) = \sum_{f \in C} \sum_{i \in ELW(f)} \big(P[test_1(f,i)]Perr_0(f) + P[test_0](f,i)Perr_1(f)\big)T_{mask}(f,i) \tag{2.9}$$

2.4 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 [63]. 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

Circuit	No. 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
x2	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

Table 2.1: Comparison of SER (FIT) data for AnSER and ATALANTA.

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 Equation 2.4. While the exact computation can be performed only for small circuits, Table 2.1 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 [92] 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 2.1 indicate that most of the inaccuracy is due to sampling, not approximate ODCs. The last two columns of Table 2.1, corresponding to exact ODC computation, show an av-

erage 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 [134, 100, 3] given the same computation time.

To characterize the gates in the circuits accurately, we adapted data from [100], where several gate types are analyzed in a 130nm, $1.2V_{DD}$ 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 [134, 135, 100] report error rates that differ by orders of magnitude. SERA tends to report error rates on the order of 10^{-3} for 180nm technology nodes, and FASER reports error rates on the order of 10^{-5} for 100nm. 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 [100]. In Figure 2.15, 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 2.2 compares AnSER with the previous work on ISCAS 85 benchmarks, using similar or identical host CPUs. While the runtimes in [24] include 50 runs, the runtimes in [100] are reported per input vector. Thus, we multiply data from [100] 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.

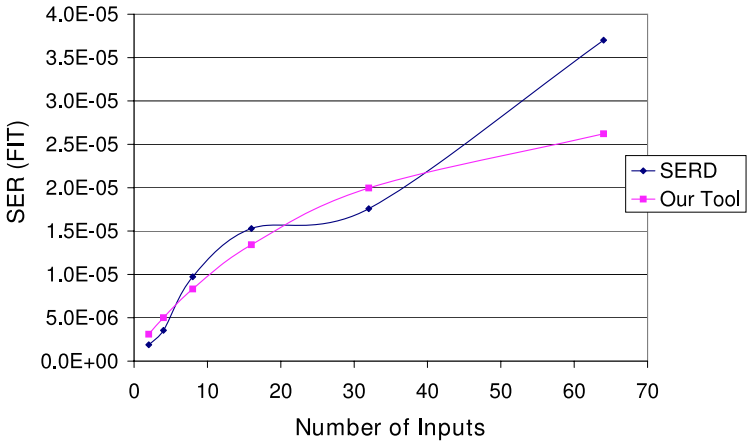


Figure 2.15: Comparison of SER trends on inverter chains produced by SERD [100] and AnSER.

Circuit	No. gates	Time (s)			
		AnSER	SERD[100]	FASER [135]	[24]
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 2.2: Runtime comparisons of four SER analyzers.

Table 2.3 shows SER and runtime results for the IWLS benchmarks, 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 distribution 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 2.4. Again, the results show that approximate ODCs are

Circuit	No. 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

Table 2.3: SER (in FITs) and runtime for AnSER on the IWLS 2005 benchmarks.

sufficient for most benchmark circuits, since the loss of accuracy due to ODC approximation is negligible.

Table 2.5 compares the multi-cycle simulation runtimes of AnSER with those of MARS-S, the sequential circuit SER analyzer from [75]. MARS-S employs symbolic simulations, using a BDD/ADD-based method to compute steady-state probability distributions, while we use signature-based bit-parallel functional simulation. The number of cycles needed to reach steady-state is also listed in the table. Table 2.6 shows the results of SER analysis on sequential circuits from the ISCAS-89 benchmark suite under time-frame expansion. The listed runtimes in Table 2.6 are for processing signatures and ODCs on 10 frames. These

Circuits	No. gates	SER (FIT)	Time (s)	SER (FIT)	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 2.4: SER evaluation of various benchmarks with exact and approximate ODCs.

Circuit	No. gates	No. 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

Table 2.5: Comparison of multi-cycle simulation runtimes.

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 [39]. 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

Circuit	No. 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 2.6: Change in SER for sequential circuits with increasing number of time frames.

outputs.

Table 2.7 shows SER results under the TMSA model, which represents single-event multiple-bit upsets. In this experiment, we included as TMSA faults, sets of 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 Figure 2.4b, 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 2005 benchmark suite [46], with design utilization set to 70% to match recent practice in industry. Our wire and gate characterizations are based on a 65nm technology library. We perform static timing analysis using the D2M delay metric [5] on rectilinear steiner minimal trees (RSMTs) produced by FLUTE [25]. These designs are placed using Capo, version 10.2 [20, 121], and relocations are legalized, i.e., gates are moved to the nearest empty or legal locations, using the legalizer provided by GSRC Bookshelf [121].

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

Table 2.7: SER under single-event multiple-bit upsets.

Circuit	No. 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

Table 2.8: SER evaluation with logic and timing masking.

Table 2.8 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 2.8 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 de-

creased by manipulating timing masking.

2.5 Summary

Efficient analysis methods are necessary for assessing and reducing the SER of a circuit. This chapter presented AnSER, our linear-time method for the logic-level soft-error analysis. AnSER achieved its low runtimes by functional simulation signatures, which enabled a fast and accurate method for computing signal probability and observability, even in the presence of reconvergent fan-out. We analyzed sequential circuits using AnSER and employing multicycle simulation and time-frame expansion. In addition, we incorporated timing masking through error-latching windows which where computed using timing analysis information. We derived results on IWLS and ISCAS benchmarks, which generally showed 2-3 orders of magnitude speed-up over previous SER analyzers and high accuracy when validated against the ATALANTA ATPG tool.

CHAPTER III

Design for Robustness

At the gate level, soft errors have traditionally been eliminated through the use of time or space redundancy. The cascaded-TMR scheme illustrated in Figure 1.11 [125] is one 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 the desired level of fault tolerance is reached. However, as we demonstrate in this chapter, it is possible to achieve improvements in reliability without resorting to explicit or massive redundancy.

In combinational logic, an SEU only affects the primary outputs if it is propagated through the intermediate gates. For instance, in Figure 1.7, the error does not propagate if $A = 0$. In this case, A controls the output of the AND gate, and stops error propagation. 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-tolerant 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.

Section 3.1 presents a signature-based method for identifying partial redundancy in the circuit. Section 3.2 develops an impact metric for selecting error-sensitive gates. Section 3.4 describes a gate-relocation technique which increases timing masking. Finally, Section 3.5 offers empirical validation of these techniques. Most of the techniques and results described in this chapter also appear in [58, 55, 59].

3.1 Signature-Based Design

In this section, we describe 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 Chapter II, signatures provide partial information about the Boolean function of a node. Therefore, candidate nodes with similar functionality can be identified by matching signatures.

We take 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.

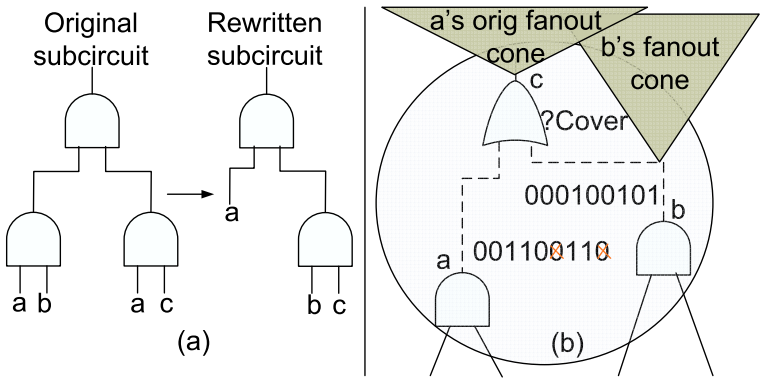


Figure 3.1: (a) Rewriting a subcircuit to improve area. (b) Finding a candidate cover for node a.

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 candidates 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. In terms of signatures, this corresponds to bitwise ANDing $sig(f)$ and $sig(x)$ by $ODCmask(x)$ to check for the following relation:

$$sig(f) \& ODCmask(x) = sig(x) \& ODCmask(x)$$

Figure 3.1a 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 $sig(x) > sig(y)$, lexicographically; 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., $|sig(x)| > |sig(y)|$, where $|sig(x)|$ 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.

Generally, several candidates satisfy implication relations for each node x. Among the candidates, we choose a node y: 1) that most often controls the output of the additional OR/AND gate, and 2) whose fan-in cone is maximally disjoint from that of x. Errors in the mutual fan-in 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 can be used to verify the implication relation. The basic process of verifying circuit optimizations with SAT is as follows [17]. 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 (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 checks if the output of the OR gate is ever 1 (satisfied). If it is, the two circuits cannot be equal. 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 [92], verification is done incrementally, starting from f and x, and moving closer to the primary outputs if the relation is unverified.

3.2 Impact Analysis and Gate Selection

Gate selection is important in many optimizations that 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 [77]. 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 fan-in cone. For instance, if a gate f is hardened, and a gate $f' \in fan\text{-}in(f)$ 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 fan-in 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 fan-in of the fan-out 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, $impact(c,(c,c')) = Perr(C'_c) - Perr(C_c)$, 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 Figure 3.2, runs in linear time and employs a notion of the observability of one node g relative to another node f.

$$relODCmask(g,f) = ODCmask(g) \& ODCmask(f) \tag{3.1}$$

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 fan-out 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 fan-in cone F of node f may be more observable than f, due to fan-out in F. For the circuit in Figure 2.3, $relODCmask(g,h) =$ 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 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 Equation 2.8.

```
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
}
```

Figure 3.2: Algorithm to approximate *impact*.

3.3 Local Rewriting

Rewriting is a general technique that optimizes small subcircuits to obtain overall area improvements [73]. 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 Figure 3.3a is more error-tolerant than the imbalanced one of Figure 3.3b, 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 [1, 73], 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. Com-

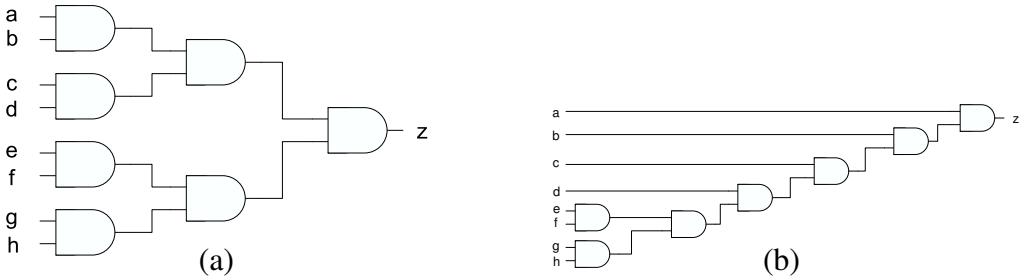


Figure 3.3: Two different realizations of an 8-input AND.

putational efficiency is achieved through fast incremental updates by AnSER. As shown in Figure 3.1a, 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.

3.4 Gate Relocation

In this section, 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 fan-out 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 fan-out 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 3.4 illustrates an example of a gate relocation. Here, gate h is moved from the position shown in Figure 3.4a to the position shown in Figure 3.4b such that $ELW(f)$ is 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.

3.5 Empirical Validation

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

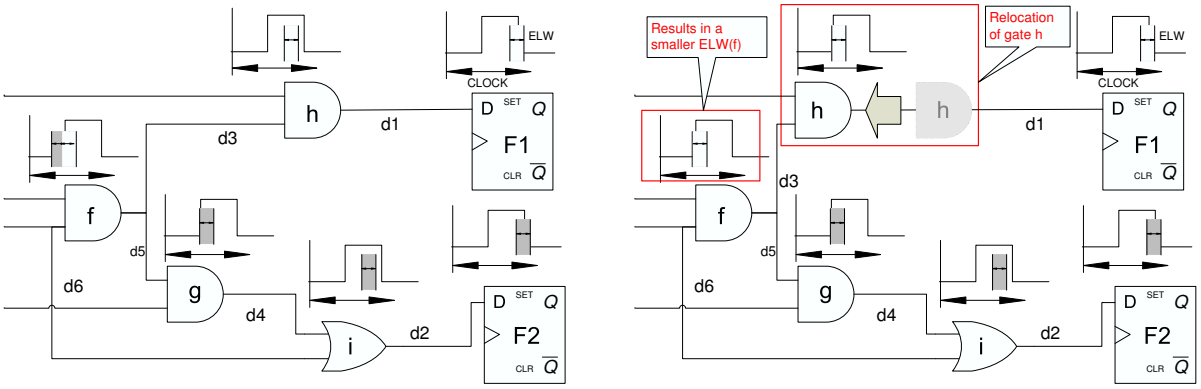


Figure 3.4: (a) Original circuit with ELWs; (b) Modified circuit with gate h relocated to decrease the size of $ELW(f)$.

Table 3.1 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 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 [76], which achieve a 91% improvement but increase area by 104% on average.

Table 3.2 illustrates the use of AnSER to guide the local rewriting method implementated in the ABC logic-synthesis package [1]. 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,

Circuit	Area	With exact covers		With approx. 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
Ave.		**29.1**	**5.5**	**39.8**	**13.1**

Table 3.1: Improvements in SER obtained by SiDeR.

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 3.3 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 benchmark suite [46], with design utilization set to 70% to match recent practice in industry. Our wire and gate characterizations are based on a 65nm technology library. We perform static timing analysis using the D2M delay metric [5] on rectilinear steiner minimal trees (RSMTs) produced by FLUTE [25]; these designs are placed using Capo, version 10.2 [20, 121], and relocations are legalized (i.e., gates are moved into the

Circuits	No. gates	No. 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
Ave.			**10.7**	**2.3**	**18.1**

Table 3.2: Improvements in SER and area with local rewriting.

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
Ave.	68.46	26.33

Table 3.3: Improvements in SER, by a combination of rewriting and SiDeR.

nearest empty cells) using the legalizer provided by GSRC Bookshelf [121].

Table 3.4 shows improvements achieved by guiding gate hardening. 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 Section 3.2. 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.

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
Ave.	61.34		43.56

Table 3.4: SER improvements through gate hardening.

Table 3.5 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 65nm 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 ITRS 2005 chapter on interconnect, which projects that at 32nm, wiring

Circuit	$65nm$		$< 45nm$	
	%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
Ave.	13.97	0.55	41.59	2.10

Table 3.5: Improvements in SER, through gate relocation.

will contribute 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.

3.6 Summary

We have developed several novel design techniques to improve circuit SER with low area and performance overhead. Our techniques are based on the careful analysis of the interplay between signal probability, observability, and masking mechanisms. The first technique, called SiDeR, finds logical implications between signals, through signature matching, and adds a few gates to decrease SER. In our second technique, several alternate non-redundant realizations of the same subcircuit are assessed for global SER improvement and selected based on an objective function that accounts for both SER and area. Our third technique provides a better method for selecting gates to harden or replicate, by accounting for errors propagating through the gate. Our fourth technique takes advantage of improvements in timing masking, by relocating gates (post placement) to minimize their

error latching windows. Our results generally show significant improvements in SER, at low cost.

CHAPTER IV

Probabilistic Transfer Matrices

Thus far, we have concentrated on soft-error analysis. Soft errors are a result of relatively rare external-particle strikes or disturbances in circuit behavior. In this chapter, we move to a more general reliability-analysis framework 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 wire are made from ”quantum dots”, are examples of such devices. 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 [62, 104]. Other examples of 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 [91], [32], [109]. 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.

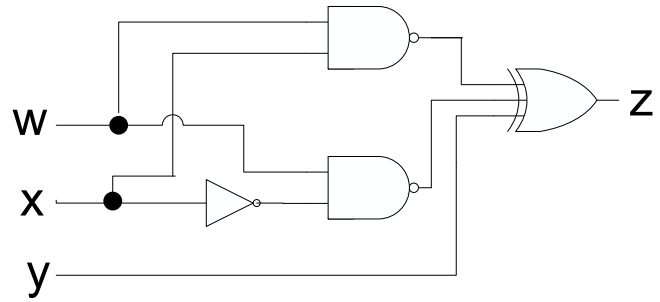
Consider the circuit in Figure 4.1. Given that each gate experiences an error with probability $p = 0.1$, the circuit's output error probability for the input combination 000 is 0.244. The input combination 111 leads to an output error probability of 0.205. The overall error rate of the circuit is the sum of the error probabilities, weighted by the input combination probabilities. The probability of error for the circuit in Figure 4.1, 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 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. Also, PTM-based computations implicitly capture signal correlations that are caused by fan-out. Most of the concepts and results described in this chapter also appear in [60, 61].

4.1 PTM Algebra

In this section, we describe the PTM algebra and some key operations needed to manipulate PTMs. First, we discuss the basic operations needed to describe 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 $in_0, \ldots in_{n-1}$; similarly, the m outputs are labeled $out_0, \ldots out_{m-1}$. 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 $\mathbf{i} = i_0 i_1 \ldots i_n$ is an n-bit input vector, then row $M(\mathbf{i})$ gives the output probability distribution for n input values $in_0 = i_0, in_1 = i_1 \ldots\ in_{n-1} = i_{n-1}$. 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 outputs have values $out_0 = j_0, out_1 = j_1 \ldots\ out_{m-1} = j_{m-1}$ given input values $in_0 = i_0, in_1 = i_1 \ldots\ in_{n-1} = i_{n-1}$, i.e, $P[outputs = \mathbf{j} | inputs = \mathbf{i}]$. Therefore, each entry in M gives the conditional probability that a certain output combination occurs given a certain input combination.



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

Figure 4.1: Sample logic circuit and its symbolic PTM formula.

$$\begin{array}{c} \\ 000 \\ 011 \\ 010 \\ 011 \\ 100 \\ 101 \\ 110 \\ 111 \end{array} \begin{array}{c} \begin{array}{cc} 0 & 1 \end{array} \\ \left[\begin{array}{cc} 1 & 0 \\ 0 & 1 \\ 1 & 0 \\ 0 & 1 \\ 0 & 1 \\ 1 & 0 \\ 0 & 1 \\ 1 & 0 \end{array}\right] \end{array} \qquad (a)$$

$$\begin{array}{c} \\ 000 \\ 011 \\ 010 \\ 011 \\ 100 \\ 101 \\ 110 \\ 111 \end{array} \begin{array}{c} \begin{array}{cc} 0 & 1 \end{array} \\ \left[\begin{array}{cc} 0.756 & 0.244 \\ 0.244 & 0.756 \\ 0.756 & 0.244 \\ 0.244 & 0.756 \\ 0.295 & 0.705 \\ 0.705 & 0.295 \\ 0.295 & 0.705 \\ 0.705 & 0.295 \end{array}\right] \end{array} \qquad (b)$$

Figure 4.2: (a) ITM for the circuit in Figure 4.1; (b) circuit PTM, where each gate experiences error with probability $p = 0.01$

Definition 10 *Given a circuit C with n inputs and m outputs, the* probabilistic transfer matrix *for C is a* $2^n \times 2^m$ *matrix M whose entries are* $M(\mathbf{i},\mathbf{j}) = P[outputs = \mathbf{j} | inputs = \mathbf{i}]$.

Definition 11 *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 $in_0 = i_0, in_1 = i_1 \ldots in_{n-1} = i_{n-1}$ 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.

4.1.1 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 $M(\mathbf{i},\mathbf{j}) = \sum_{all\ \mathbf{l}} M_1(\mathbf{i},\mathbf{l})M_2(\mathbf{l},\mathbf{j})$ This operation corresponds to the ordinary matrix product M_1M_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. Each 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 12 *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:*

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

.

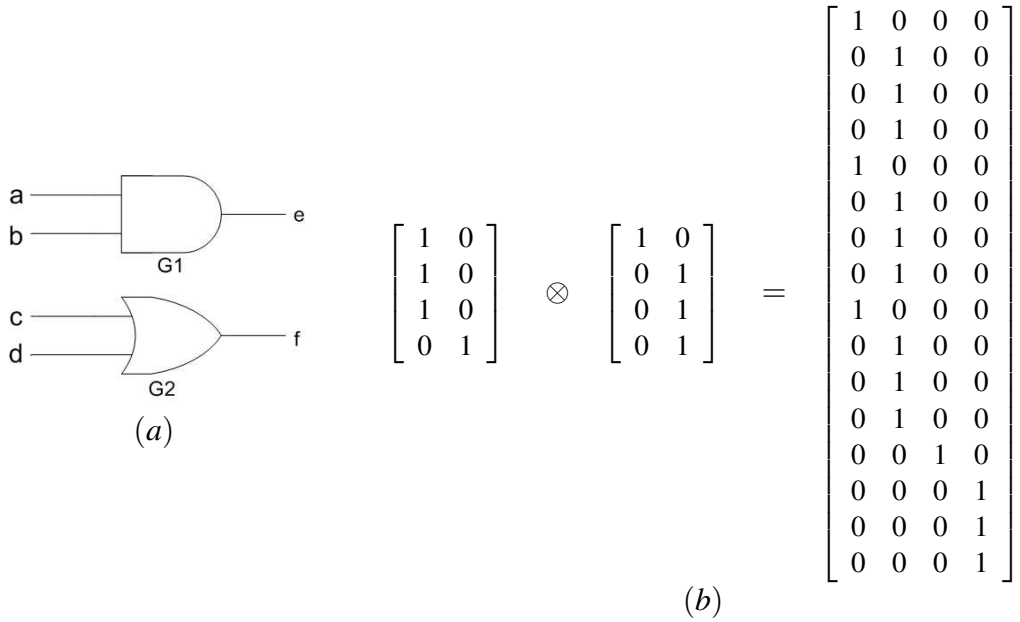


Figure 4.3: Illustration of the tensor product operation: (a) circuit with parallel AND and OR gates; (b) circuit ITM formed by the tensor product of the AND and OR ITMs.

Figure 4.3 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.

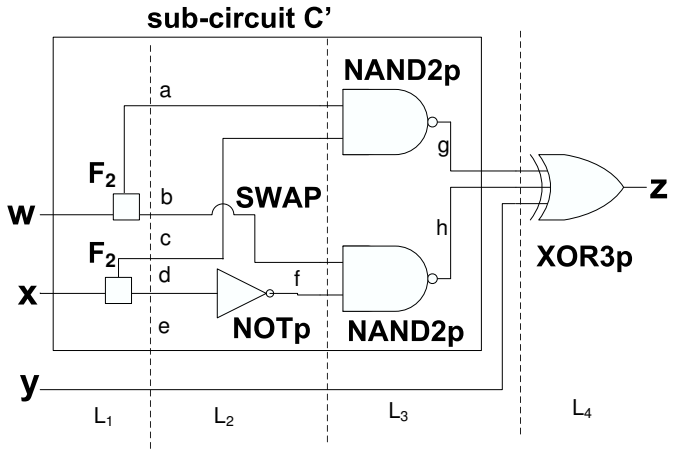
Besides the usual logic gates (*AND*, *OR*, *NOT*, etc.), it is useful to define three special gates for circuit PTM computation. These are (i) the n-input identity gate, with ITM denoted I_n; (ii) the n-output fan-out gate F_n; and (iii) the swap gate, *swap*. These wiring PTMs are shown in Figure 4.1.

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

$$\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \quad \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad swap = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

(a) (b) (c)

Figure 4.4: Wiring PTMs: (a) identity gate (I) ; (b) 2-output fan-out gate (F_2); (c) adjacent swap gate ($swap$).



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

Figure 4.5: Circuit to illustrate PTM calculation; vertical lines separate levels of the circuit; the parenthetical subexpressions correspond to logic levels.

outputs. The ITM of a 2-output fan-out gate, shown in Figure 4.1b, 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 Figure 4.1c. Any permutation of wires can be modeled by a network of swap gates.

Example 5 *Consider the circuit in Figure 4.5—this is the same circuit as in Figure 4.1, with the wiring gates made explicit. The PTMs for the gates with error probability p are as follows:*

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

$NAND2_p$ $XOR3_p$ NOT_p

The circuit PTM is expressed symbolically by the formula in Figure 4.5. Each parenthesized term in the equation 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.

4.1.2 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 13 *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, $||\mathbf{v}||$ denotes the l_1 norm of vector $\mathbf{v}$.

Definition 14 *Given a circuit C with PTM M, ITM J, and input vector* $\mathbf{v}$*, the fidelity of M is given by*

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

The fidelity of a circuit is a measure of its *reliability*. Figure 4.6 illustrates the $fidelity$ computation on the circuit from Figure 4.1. The ITM, shown in Figure 4.2a, is denoted J, and the PTM, shown in Figure 4.2b, is denoted M.

$$v^T = \begin{bmatrix} 0.125 \\ 0.125 \\ 0.25 \\ 0.25 \\ 0 \\ 0 \\ 0.125 \\ 0.125 \end{bmatrix} \quad (a) \qquad J.*M = \begin{bmatrix} 0.756 & 0 \\ 0 & 0.756 \\ 0.756 & 0 \\ 0 & .756 \\ 0 & 0.705 \\ 0.705 & 0 \\ 0 & 0.705 \\ 0.705 & 0 \end{bmatrix} \quad (b) \qquad v(J.*M) = \begin{bmatrix} 0.567 & 0.177 \end{bmatrix} \quad (c)$$

Figure 4.6: Matrices used to compute $fidelity$ for the circuit in Figure 4.1: (a) input vector; (b) result of element-wise multiplication of its ITM and PTM; (c) result of left-multiplication by the input vector.

Example 6 *Consider the circuit C from Figure 4.1, with inputs* $\{w,x,y\}$ *and output* $\{z\}$ *. The circuit PTM is calculated using the PTMs from Example 5, with probability of error* $p = 0.05$ *at each gate, on all inputs. Figure 4.6 shows intermediate matrices needed for this computation. The quantity* $fidelity(v,M,J)$ *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* $fidelity(v,M,J) = (0.567+0.177) = 0.743$*. The probability of error is* $1-0.743 = 0.257$*.*

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.

Definition 15 *Given a PTM matrix M that represents a circuit C with inputs* $in_0 \dots in_{n-1}$*,* $eliminate\_variables(M, in_k)$ *is the matrix* M' *with* $n-1$ *input variables* $in_0 \dots in_{k-1} in_{k+1} \dots in_{k+1} \dots in_{n-1}$ *whose rows are*

$$M'(i_0 \dots i_{k-1} i_{k+1} \dots i_{n-2}, \mathbf{j}) = M(i_0 \dots i_{k-1}\ 0\ i_{k+1} \dots i_{n-2}, \mathbf{j}) + M(i_0 \dots i_{k-1}\ 1\ i_{k+1} \dots i_{n-2}, \mathbf{j})$$

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. Figure 4.7 demonstrates the elimination of column variables from a subcircuit C' of the circuit in Figure 4.5, formed by the logic between inputs w,x and outputs g,h. The PTM for C' with probability of error $p = 0.05$ on all its gates is given by:

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

[1] The *eliminate_variables* operation is analogous to the existential abstraction of a set of variables x in a Boolean function f [37], 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 on PTMs relies on arithmetic addition of matrix entries instead of the Boolean disjunction of cofactors.

$$\begin{array}{cccc} 00 & 01 & 10 & 11 \\ \left[\begin{array}{cccc} 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \end{array}\right] \end{array}$$

(a)

$$\begin{array}{cccc} 00 & 01 & 10 & 11 \\ \left[\begin{array}{cccc} 0.0025 & 0.0475 & 0.0475 & 0.9025 \\ 0.0025 & 0.0475 & 0.0475 & 0.9025 \\ 0.04525 & 0.00475 & 0.85975 & 0.09025 \\ 0.09025 & 0.85975 & 0.00475 & 0.04525 \end{array}\right] \end{array}$$

(b)

$$\begin{array}{cc} 0 & 1 \\ \left[\begin{array}{cc} 0.0025+0.0475 & 0.0475+0.9025 \\ 0.0025+0.0475 & 0.0475+0.9025 \\ 0.04525+0.00475 & 0.85975+0.09025 \\ 0.09025+0.85975 & 0.00475+0.04525 \end{array}\right] \end{array}$$

(c)

$$\begin{array}{cc} 0 & 1 \\ \left[\begin{array}{cc} 0.0025+0.0475 & 0.0475+0.9025 \\ 0.0025+0.0475 & 0.0475+0.9025 \\ 0.04525+0.85975 & 0.00475+0.09025 \\ 0.09025+0.00475 & 0.85975+0.04525 \end{array}\right] \end{array}$$

(d)

Figure 4.7: Example of the *eliminate_variables* operation: (a) ITM of subcircuit C' from Figure 4.5; (b) PTM of C'; (c) Output variable h eliminated; (d) Output variable g eliminated.

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 Figure 4.7b. To eliminate this variable, columns with indices 00 and 01 of Figure 4.7b are added, and the result is stored in the column 0 of the resultant matrix (Figure 4.7c). 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 elimination of g in the PTM of Figure 4.7d. 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 Figure 4.5 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 5 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 16 *Given a circuit C with n inputs* $in_0, \ldots in_{n-1}$*, and PTM M, let* in_k *and* in_l *be two inputs that are identified with (connected to) each other. Then* $eliminate\_redundant\_variables(M, in_k, in_l) = M'$*, where* M' *is a matrix with* $n-1$ *input variables whose rows are*

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

The definition of *eliminate_redundant_variables* can be extended to a set of input variables that are redundant. Figure 4.8 shows an example of the *eliminate_redundant_variables* 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.

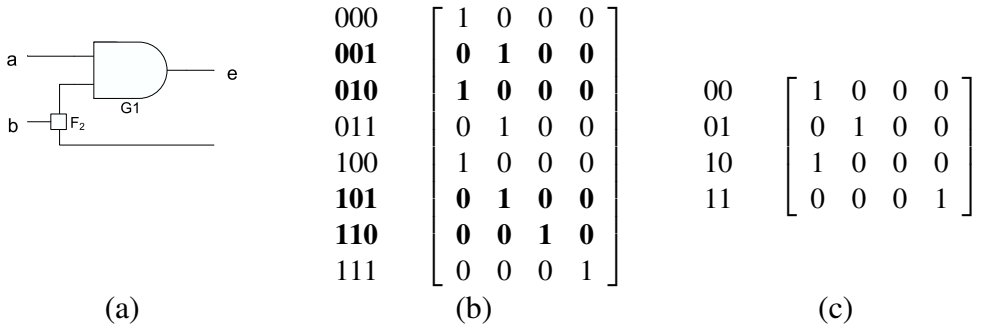


Figure 4.8: Signal forwarding using *eliminate_redundant_variables*: (a) circuit with signal b fanning out to two different levels; (b) $NAND \otimes I$, adding b as an input and output; (c) final ITM for circuit computed by removing rows in boldface.

The *eliminate_redundant_variables* operation can efficiently handle fan-out to different levels by "signal forwarding," as seen in Figure 4.8. 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 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.

4.1.3 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.

Example 7 *Suppose two wires have a* 0.25 *probability of swapping. The matrix corre-*

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

(a) (b) (c)

Figure 4.9: 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*.

sponding to this error is given in Figure 4.9a. If we try to separate the probability of each output, using eliminate_variables, the output probabilities both have the PTM of Figure 4.9b. If these outputs are tensored (with redundant inputs eliminated), they result in the erroneous combined matrix of Figure 4.9c. 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, which 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 in soft-error analysis and error-threshold computation.

4.2 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.

4.2.1 Fault Modeling

The PTM model can represent a wide variety of faulty circuit behaviors, including both hard and soft errors. The fact that there are separate probabilities for each input and output, and the fact that they are propagated simultaneously make this possible. Figure 4.10 lists some errors that can be represented by PTMs.

$$
\begin{array}{c|cc}
 & 0 & 1 \\
\hline
000 & 1 & 0 \\
011 & 1 & 0 \\
010 & 1 & 0 \\
011 & 0 & 1 \\
100 & 0 & 1 \\
101 & 1 & 0 \\
110 & 0 & 1 \\
111 & 0 & 1
\end{array}
\qquad
\begin{array}{c|cc}
 & 0 & 1 \\
\hline
000 & 1 & 0 \\
011 & 1 & 0 \\
010 & 1 & 0 \\
011 & 0 & 1 \\
100 & 1 & 0 \\
101 & 1 & 0 \\
110 & 1 & 0 \\
111 & 0 & 1
\end{array}
\qquad
\begin{array}{c|cc}
 & 0 & 1 \\
\hline
000 & 1 & 0 \\
011 & 1 & 0 \\
010 & 0 & 1 \\
011 & 1 & 0 \\
100 & 1 & 0 \\
101 & 0 & 1 \\
110 & 0 & 1 \\
111 & 0 & 1
\end{array}
\qquad
\begin{array}{c|cc}
 & 0 & 1 \\
\hline
000 & 0.95 & 0.05 \\
011 & 0.95 & 0.05 \\
010 & 0.95 & 0.05 \\
011 & 0.05 & 0.95 \\
100 & 0.05 & 0.95 \\
101 & 0.95 & 0.05 \\
110 & 0.05 & 0.95 \\
111 & 0.05 & 0.95
\end{array}
\qquad
\begin{array}{c|cc}
 & 0 & 1 \\
\hline
000 & 1 & 0 \\
011 & 0 & 1 \\
010 & 0 & 1 \\
011 & 1 & 0 \\
100 & 0 & 1 \\
101 & 1 & 0 \\
110 & 0 & 1 \\
111 & 1 & 0
\end{array}
$$

(a) (b) (c) (d) (e)

Figure 4.10: PTMs for various types of gate errors: (a) a fault-free ideal 2-1 MUX gate; (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.

Figure 4.10a shows the ITM for a fault-free ideal 2-1 multiplexer (MUX). Figure

4.10b 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. Figure 4.10c shows an example with the first two wires swapped; this is captured by permuting the rows of the ITM, accordingly. Figure 4.10d 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. Figure 4.10e shows a design error where a MUX has been replaced by an XOR gate. As these examples indicate, PTMs can capture both gate errors and wiring errors.

PTMs can also represent errors 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. Bit-flips on wires can be represented by the 1-input identity gate I, with probabilistic errors as shown below.

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

As mentioned in Chapter I, adjacent wires in nanoscale CMOS circuits can suffer from crosstalk. When two adjacent wires run parallel to each other, there is capacitive and inductive coupling, which can cause neighboring signals to erroneously switch [96]. Crosstalk can be represented by a faulty 16×2 matrix, which takes transitions rather than signal values into account. Hence, we need circuit information for two consecutive time units.

Consider the circuit shown in Figure 4.11. Let $a_0, b_0, \ldots y_0, z_0$ denote signals at time unit t_0 and $a_1, b_1, \ldots y_1, z_1$ denote the same signals at time t_0. Suppose we identify f and g as likely candidates for crosstalk faults, with f being the aggressor signal and g being the victim. If f transitions from 0 to 1, then due to crosstalk, g also has a chance of erroneously

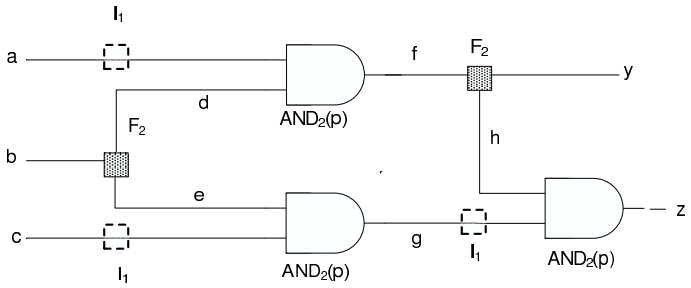


Figure 4.11: Circuit to illustrate crosstalk faults.

$$
\begin{array}{ll}
0000 & \begin{bmatrix} 1 & 0 \end{bmatrix} \\
0001 & \begin{bmatrix} 0 & 1 \end{bmatrix} \\
0010 & \begin{bmatrix} 1 & 0 \end{bmatrix} \\
0011 & \begin{bmatrix} 0 & 1 \end{bmatrix} \\
0100 & \begin{bmatrix} 1-p & p \end{bmatrix} \\
0101 & \begin{bmatrix} 0 & 1 \end{bmatrix} \\
0110 & \begin{bmatrix} p & 1-p \end{bmatrix} \\
0111 & \begin{bmatrix} 0 & 1 \end{bmatrix} \\
1000 & \begin{bmatrix} 1 & 0 \end{bmatrix} \\
1001 & \begin{bmatrix} p & 1-p \end{bmatrix} \\
1010 & \begin{bmatrix} 1 & 0 \end{bmatrix} \\
1011 & \begin{bmatrix} p & 1-p \end{bmatrix} \\
1100 & \begin{bmatrix} 1 & 0 \end{bmatrix} \\
1101 & \begin{bmatrix} 0 & 1 \end{bmatrix} \\
1110 & \begin{bmatrix} 1 & 0 \end{bmatrix} \\
1111 & \begin{bmatrix} 0 & 1 \end{bmatrix}
\end{array}
$$

Figure 4.12: Representing a crosstalk error using PTMs.

transitioning similarly. This is represented by the PTM with inputs f_0, f_1, g_0, g_1 and output g_1' which contains an erroneous transition with probability p, as shown in Figure 4.12.

While we have explicitly modeled various types of errors in conventional CMOS nanoscale circuits, we have not modeled errors whose probabilities change dynamically. For instance, we do not model time-varying error probabilities or errors whose probabilities are described by probability-distribution functions (PDFs). However, in the future, using PTMs it may be possible to model such cases. For instance, in order to use PTMs

with entries formed by PDFs, the tensor and matrix multiplications operations will need to be redefined to handle sums (convolutions) and products of probability distributions. However, as we show in the next section, if error probabilities depend on specifically identifiable discrete variables, apart from the input variables, these too can be incorporated into PTMs.

4.2.2 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 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 $p_{occur}(g)$. However, SEU strikes create glitches which can be differentiated by a combination of shape and amplitude. These differentiations are important in the propagation of a glitch through circuit gates. There-

fore, we utilize a modified identity matrix known as $I_{1,n}(p_{occur})$ to represent a probability distribution on a glitch induced by an SEU strike.

We use the specific glitch-propagation model from [88] to determine which signal characteristics to capture; a different model might require other characteristics to be represented. In [88], glitches are classified into three types, depending on their duration D, and amplitude A, relative to the gate propagation delay T_p, and threshold voltage V_t. Glitches are assumed to change only logic-0 to logic-1 when they strike (but can later be inverted).

- 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.

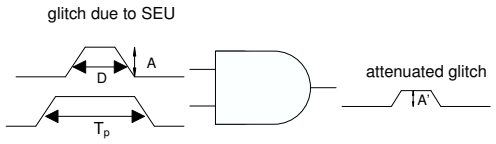
Since amplitude is already indicated by the logic value, we need an additional bit to indicate whether the duration is larger or smaller than the propagation delay of the gate (when the amplitude is higher than the threshold voltage). The duration is irrelevant for glitches with amplitude lower than the threshold voltage, since these are likely to be attenuated. Figure 4.13a 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, Figure 4.13a assumes uniform distribution, with respect to glitches.

Once an SEU strikes a gate 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 $p_{prop}(g)$. The glitch-transfer characteristics of a logic gate are described by a modified gate PTM that represents relevant characteristics of the glitch. For instance, Figure 4.13b shows a modified *AND* PTM, denoted $AND_{2,2}(p_{prop})$.

In the selected glitch model [88], 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 value given by the logic function of the gate. This transfer function can be described by the PTM of Figure 4.13b. 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 instance, if a signal has 2-bit distribution $[.2 \;\; .1 \;\; .3 \;\; .4]$, since the second bit indicates duration, the probability of a logic 0 is $.2 + .1$ and the probability of a logic 1 is $.3 + .4$. Figure 4.14 shows a circuit with the corresponding ITM and PTMs with multi-bit signal representations.

Example 8 *For the circuit in Figure 4.15, 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*



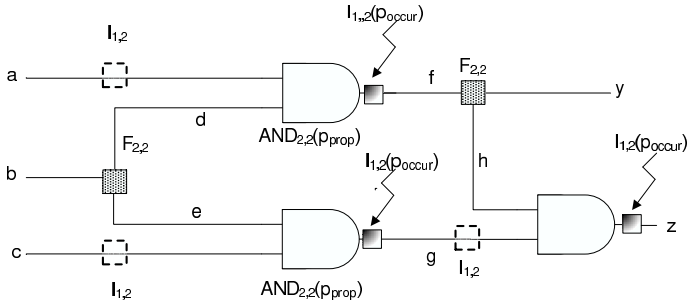
$$
\begin{array}{c}
00 \\ 00 \\ 10 \\ 11
\end{array}
\begin{bmatrix}
1-p_{occur} & p_{occur}/3 & p_{occur}/3 & p_{occur}/3 \\
0 & 1 & 0 & 0 \\
0 & 0 & 1 & 0 \\
0 & 0 & 0 & 1
\end{bmatrix}
$$

(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)

Figure 4.13: PTMs for SEU modeling where the row labels indicate input signal type: (a) $I_{2,2}(p_{occur})$ describes a probability distribution on the energy of an SEU strike at a gate output, (b) $AND_{2,2}(p_{prop})$ describes SEU-induced glitch propagation for a 2-input AND gate. The type-2 glitches become attenuated to type 3 with a probability $1-p_{prop}$.



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

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

Figure 4.14: Circuit with ITM and PTMs describing an SEU strike and the resultant propagation with multi-bit signal representations.

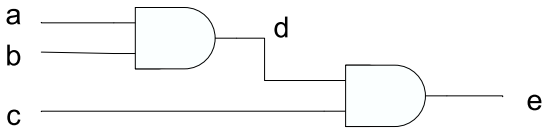


Figure 4.15: Circuit used in Example 8 to illustrate the incorporation of electrical masking into PTMs.

sensitizes the appropriate path to d and then e. If we let $p_{occur} = 0.001$ *and* $p_{prop} = 0.5$*,*

and $AND_{2,2}(p_{prop})$ *is as shown in Figure 4.13, then the circuit PTM is given by:*

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

The corresponding PTM and fidelity are given in Figure 4.16.

$$fidelity = .99994791$$

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

4.2.3 Error Transfer Function

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

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

Figure 4.16: PTM for the circuit used in Example 8 which incorporates electrical properties of the gates.

the error transfer functions of standard benchmark circuits. Such functions can be used to derive upper bounds for tolerable levels of gate error.

Definition 17 *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 4.17 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 14. Generally, such error transfer curves can be described by polynomials. If two gates have error p, then their composition (series, parallel, or a combination of both) has terms that are linear combinations of p^2 and p,

the overall probability of error $O(p^2)$. 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, Figure 4.17 has probabilities of gate error up to 1 to make the polynomial nature of the curves evident.

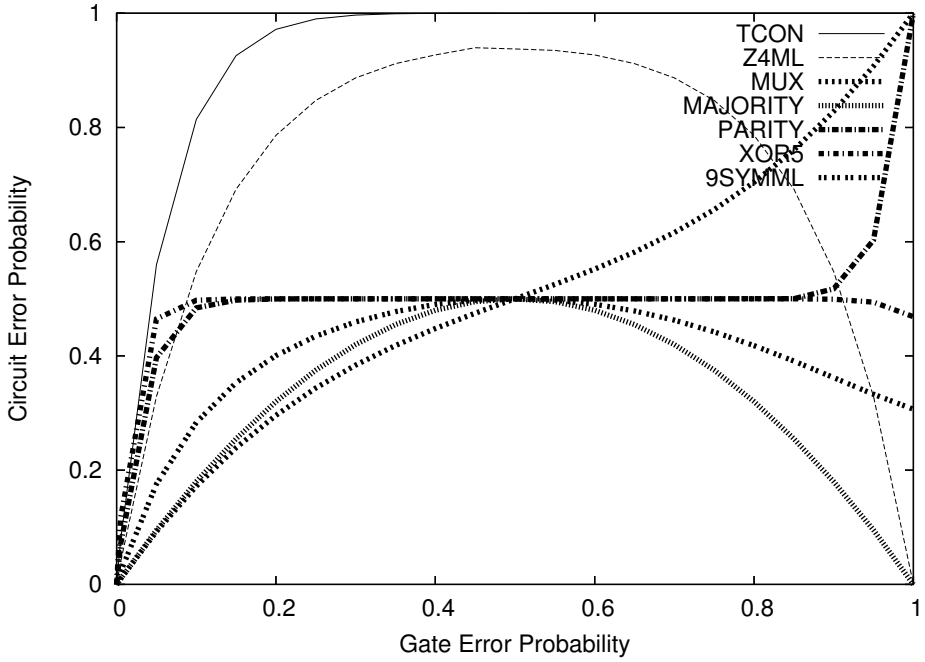


Figure 4.17: Circuit error probability under various gate error probabilities.

Table 4.1 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 [94]. However, Figure 4.17 suggests that for most circuits, replicating the entire circuit at gate errors of 0.20 or more will only increase circuit error.

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

Table 4.1: Polynomial approximations of circuit error transfer curves and residual errors. The fitted polynomials are of the form $e(x) \approx a_0 + a_1x + a_2x^2 + a_3x^3 \ldots$.

4.3 Summary

In this chapter, we proposed the probabilistic transfer matrix (PTM) to capture non-deterministic behavior in logic circuits. PTMs provide a concise description of both normal and faulty behavior and are well-suited to reliability analysis. A few simple composition rules based on connectivity can be used to recursively build larger PTMs representing entire logic circuits from smaller gate PTMs. PTMs can accurately calculate joint output probabilities in the presence of reconvergent fan-out and inseparable joint input distributions. In addition, we defined new matrix operations to eliminate variables, remove redundancy, and compute the overall probability of circuit error. We showed how PTMs can be used to capture various sorts of errors inherent in nanocircuits, including soft errors. We also used PTMs to derive polynomial approximations for circuit error probabilities in terms of gate error probabilities for the purpose of determining thresholds of acceptable gate error for specific circuits.

CHAPTER V

Computing with PTMs

Circuit PTMs have exponential space complexity because they contain information about all possible input vectors. Their complexity makes numerical computation with PTMs impractical for circuits with more than 10-15 inputs. In order to improve scalability, we develop an implementation of the PTM framework that uses algebraic decision diagrams (ADDs) to compress matrices. We also derive several ADD algorithms to combine PTMs directly in their compressed forms.

Figure 4.2 gives a PTM for the circuit in Figure 4.1 along with the corresponding ADD, representing the case where all gates experience output bit-flips with probability $p = 0.1$. As the figure shows, the same values occur multiple times in the matrix and suggest a possibility of compression. Due to the canonicity of ADD/BDD representation, identical subgraphs, corresponding to identical submatrices, can be identified and eliminated during the process of ADD-construction. In some cases, ADDs contain exponentially fewer nodes than the number of entries in the explicit matrix representation. In such cases, linear-algebraic transformations can be applied exponentially faster to the ADD than to the matrix.

Developing efficient ADD algorithms for PTM operations is a significant technical challenge that we address in this chapter. We adapt previous ADD algorithms from [8] and [123] for tensor and matrix products. The original versions of their algorithms handle only square matrices, while PTMs are generally rectangular. In addition, we develop ADD algorithms for the new operations defined in Chapter IV. 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 [60, 61].

5.1 Compressing 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.

Recall 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.

ADDs are variants of BDDs in which terminal nodes can take on any real value. Bahar et al. [8] present a method of encoding a matrix, using an ADD. The ADD encoding of a matrix M is a rooted directed acyclic graph whose entries depend on the 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 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 pre-defined order and facilitate reductions, through the use of a single subgraph for identical submatrices.

We use the QuIDDPro library [123] to encode PTMs as ADDs. We also added functions to this library for perform operations on PTMs. QuIDDPro includes the CUDD library [117] 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-$then$-$else$ 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 [8, 26]. 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; Figure 5.1 illustrates a situation in which missing variables can create ambiguity.

$$\begin{array}{c} \begin{matrix} 0 & 1 \end{matrix} \\ \begin{bmatrix} p & 1-p \\ p & 1-p \\ p & 1-p \\ 1-p & p \end{bmatrix} \\ (a) \end{array} \qquad \begin{array}{c} \begin{matrix} 00 & 01 & 10 & 11 \end{matrix} \\ \begin{bmatrix} p & 1-p & p & 1-p \\ p & 1-p & p & 1-p \\ p & 1-p & p & 1-p \\ 1-p & p & 1-p & p \end{bmatrix} \\ (b) \end{array}$$

Figure 5.1: PTMs with identical ADDs without zero-padding: (a) matrix with only one column variable; (b) matrix without dependency on the second column variable.

Figure 5.2 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 Figure 5.2, 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 incorrect.

```
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
}
```

Figure 5.2: Algorithm to pad matrices with zeros.

Figure 5.3 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 Figure 4.8. 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

$$\underset{(a)}{\begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}} \otimes \underset{(b)}{\begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix}} = \underset{(c)}{\begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}}$$

Figure 5.3: (a) NOT gate ITM; (b) zero-padded NAND gate ITM; (c) their tensor product with incorrect placement of all-zero columns.

in Figure 5.2.

5.1.1 Computing Circuit PTMs

We present an algorithm for computing the ADD of a circuit's PTM in Figure 5.4, following the method illustrated in Example 5 of Chapter IV. 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 to 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 Figure 5.4, 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.

A detail not shown in Figure 5.4 is that when a circuit has fan-out 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 Figure 4.8. 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 fan-out gates, we use the $eliminate\_redundant\_variables$ operation (Definition 16), whose ADD implementation is given in Figure 5.5. By removing each duplicated (due to fan-out) 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.

In Figure 5.5, 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

```
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
}
```

Figure 5.4: 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.

and column variables, since they both remove variables. Therefore, it may be necessary to introduce zero-padding. (Figure 5.2).

Once the ADD for the PTM and ITM of a circuit is known, we can compute the $fidelity$ of the circuit to extract reliability information (see Figure 5.6). 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 ADDs in their arguments.

Circuit	Characteristics		Reliability, $p = 0.05$		No.	Stats., $p = 0$		Stats., $p = 0.05$	
	No gates	Width	Two -way	One -way	ADD nodes	Mem. (MB)	Time (s)	Mem. (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

Table 5.1: Statistics on various small benchmarks.

Results from the calculation of circuit ITMs, PTMs, and $fidelity$ are listed in Table 5.1. We used the smaller LGSynth 91 and LGSynth 93 benchmarks with uniform input distributions. These simulations were conducted on a Linux workstation with a 2GHz Pentium 4 processor. In these experiments, CPU time was limited to 24 hours. 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

```
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
}
```

Figure 5.5: Algorithm to eliminate redundant variables.

is also listed in Table 5.1, including intermediate computations.

Table 5.1 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

```
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
}
```

Figure 5.6: Algorithm to compute $fidelity$.

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 5.1. Circuits with a high output-to-input ratio, such as decod.blif, tend to magnify gate errors at fan-out 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 0s, which lend them-

selves 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 6. 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.

5.2 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 Section 5.2.1, we propose to improve the efficiency of PTM computation by pre-scheduling 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 inputs and outputs in between the clusters. In effect, such an ordering would enclose fan-out branches and reconvergences within the clusters.

In Section 5.2.2, 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 Section 5.2.3, 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.

5.2.1 Dynamic Evaluation Ordering

The ADD-based multiplication algorithm used in our PTM algebra implementation, from [8], 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 Figure 5.4 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.

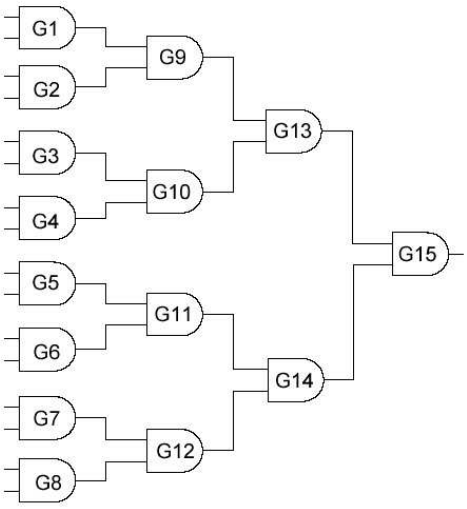


Figure 5.7: Tree of AND gates used in Example 9 to illustrate the effect of evaluation ordering on computational efficiency.

Example 9 *Consider the tree of AND gates in Figure 5.7. Suppose we wish to compute its circuit PTM. The algorithm of Figure 5.4 requires topologically sorting the gates, calculat-*

ing 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* $2^2 \times 2, 2^4, 2^2, 2^8 \times 2^4$, *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 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 [27], 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.

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

Table 5.2: Comparison of runtimes and memory usage for levelized ordering and ordering computed by dynamic programming.

The results of applying the improved ordering for multiplication of levels are given in Table 5.2. The data in this table were produced on a Pentium 4 processor running at 2GHz. In general, this ordering method uses less memory, with only a modest increase in runtime. The runtime increase seen in Table 5.2 is partially due to the overhead of the dynamic programming. However, this tradeoff is acceptable since memory was the main bottleneck.

5.2.2 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.

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.

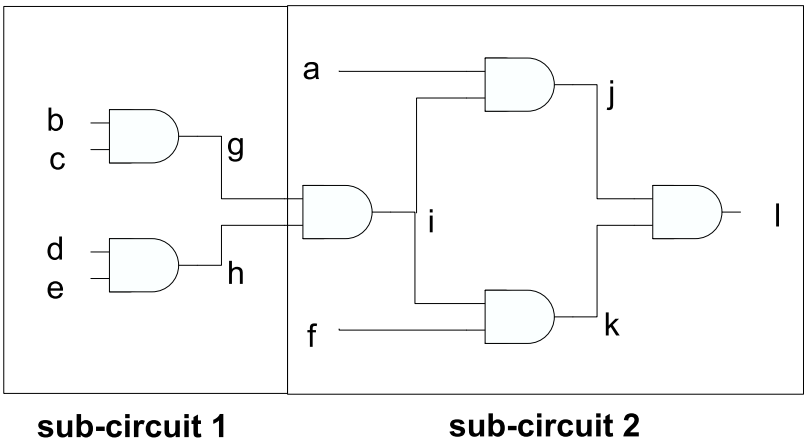


Figure 5.8: Circuit used in Example 10 to illustrate hierarchical reliability estimation .

Definition 18 *For a PTM M and an output variable o_k, $M' = abstract(M,k)$ is the matrix which results from the elimination of all variables* except o_k *from M. Therefore, $M' = eliminate\_variables(M,0,1,2\ldots k-1,k+1\ldots m)$*

Definition 19 *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 Figure 5.9, 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 fan-out between the subcircuits. In fact, the error probabilities are exact when each output bit has the same error on all input combinations

```
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')
}
```

Figure 5.9: The *Bit_fidelity* estimation algorithm.

because, in such cases, averaging does not cause a loss of information. In other cases, the accuracy will depend on the amount of correlation between signals and the variation in signal errors.

Example 10 *We apply the algorithm of Figure 5.9 to the circuit in Figure 5.8. Assume that each of the AND gates in Figure 5.8 has the following PTM and ITM:*

$$AND2_{0.1} = \begin{bmatrix} 0.9000 & 0.1000 \\ 0.9000 & 0.1000 \\ 0.9000 & 0.1000 \\ 0.1000 & 0.9000 \end{bmatrix} \quad AND2 = \begin{bmatrix} 1 & 0 \\ 1 & 0 \\ 1 & 0 \\ 0 & 1 \end{bmatrix}$$

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* $Distro[a] = Distro[B] = Distro[c] = Distro[e] = Distro[f] = [0.5 \;\; 0.5]$. *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:*

$$ITM1 = AND2 \otimes AND2$$

$$PTM1 = AND2_{0.1} \otimes AND2_{0.1}$$

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

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

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

$$Distro[g] = abstract(vin_1, g) = [0.75 \;\; 0.25]$$

$$Distro[h] = abstract(vin_1, h) = [0.75 \;\; 0.25]$$

$$PTM1' = (I(1) \otimes I(1) \otimes I(1) \otimes I) * PTM1 = PTM1$$

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

$$Bfid[h] = 0.9$$

Similarly for subcircuit 2*:*

$$ITM2 = (I \otimes AND2 \otimes I)(I \otimes F_2 \otimes I)(AND2 \otimes AND2)(AND2)$$

$$PTM2 = (I \otimes AND2_{0.1} \otimes I)(I \otimes F_2 \otimes I)(AND2_{0.1} \otimes AND2_{0.1})(AND2_{0.1})$$

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

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

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

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

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

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

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.

5.2.3 Approximation by Sampling

Reliability estimation requires computing the 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 Chapter II 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 Figure 2.7 can be used to compute the error probability, 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 Chapter VI.

5.3 Summary

To improve computational efficiency, we encoded PTMs as algebraic decision diagrams (ADDs). We also developed equivalent ADD algorithms for our newly defined matrix operations that can be used to compute circuit fidelity and to obtain marginal error probabilities for subsets of outputs. We also developed methods to handle non-square matrices, which were not represented as decision diagrams in past literature. In addition, we presented several heuristic methods for improving the scalability of PTMs, including sam-

pling, partitioning, evaluation ordering, and hierarchical computations which allow PTM computations to scale to larger circuits.

CHAPTER VI

Testing for Probabilistic Faults

After circuits are manufactured, it is important to insure 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 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 section, we develop methods selecting test vectors such that test application time is minimized.

Generating tests for probabilistic faults is fundamentally different from previous 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 6.1 summarizes these differences.

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

Table 6.1: Key differences between deterministic and probabilistic testing.

In Section 6.1, we define and provide computation methods for test-vector sensitivity to faults. Section 6.2 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 [53, 54].

6.1 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 11 *Consider the circuit in Figure 6.1. If an SEU occurs with a certain probability at input b, then the test vectors that propagate the induced error to the outputs are:* $t_1 = 001$ *(to output Y),* $t_2 = 100$ *(to output Z), and* $t_3 = 101$ *(to both Y and Z).*

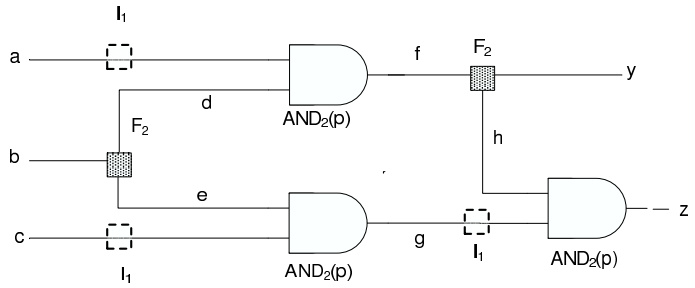


Figure 6.1: Circuit to illustrate test-vector sensitivity computation.

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 $p_{t_1} = p_{prop}^2$ of propagating the error to an output, t_2 has probability $p_{t_2} = p_{prop}$, and t_3 has probability $p_{t_3} = p_{t_1} + p_{t_2} - p_{t_1} p_{t_2}$. 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_3. 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 Chapter IV. 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 $F = \{f_1, f_2, \ldots f_n\}$ 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 13) of the correct and faulty output vectors. This operation is similar to the $fidelity$ operation of Chapter IV, defined for vectors rather than matrices.

$$sens(F,t) = 1 - ||(v_t M_f) .* (v_t M)||_{l_1} \tag{6.1}$$

The sensitivity of test vector

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

for the circuit in Figure 6.1, with the probability of error $p = 0.1$, can be computed from the circuit's ITM M, and PTM M_F, as shown in Figure 6.2. 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 Chapter IV, 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

$$v_t \times M_F = \quad v_t \times \begin{bmatrix} 0.8100 & 0.0900 & 0.0820 & 0.0180 \\ 0.8100 & 0.0900 & 0.0820 & 0.0180 \\ 0.8100 & 0.0900 & 0.0820 & 0.0180 \\ 0.8100 & 0.0900 & 0.0180 & 0.0820 \\ 0.8100 & 0.0900 & 0.0820 & 0.0180 \\ 0.8100 & 0.0900 & 0.0820 & 0.0180 \\ 0.0900 & 0.0100 & 0.7380 & 0.1620 \\ 0.0900 & 0.0100 & 0.1620 & 0.7380 \end{bmatrix} \quad = \begin{bmatrix} 0.8100 \\ 0.0900 \\ 0.0820 \\ 0.0180 \end{bmatrix}^T$$

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

$$sens(F,t) = 1 - |||[1\ \ 0\ \ 0\ \ 0].*[0.81\ \ 0.09\ \ 0.082\ \ 0.018]||$$

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

Figure 6.2: Sensitivity computation on the circuit of Figure 6.1.

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 Equation 6.1 to compute the sensitivity.

A caveat in output vector computation is that fan-out branches result in inseparable probability distributions 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 12 *Consider the circuit in Figure 6.1. Suppose the primary input vectors are* v_a, v_b, v_c*, and the 2-input AND gates have PTM* $AND_2(p)$*. The faulty output vector is given by computing as follows.*

$v_{d,e} = v_b \times F_2$
$enlargedAND_2(p) = (AND_2(p) \otimes I_1)$
$v_{a,d,e} = (v_a \otimes v_{d,e})$
$v_{f,e} = v_{a,d,e} \times enlargedAND_2(p)$
$v_{f,g} = (v_{f,e} \otimes v_c) \times (I_1 \otimes AND_2(p))$
$v_{y,h,g} = v_{f,g} \times (F_2 \otimes I_1)$
$v_{y,z} = v_{y,h,g} \times (I_1 \otimes AND_2(p))$

Additionally, note that signal probabilities incorporate the effects of reconvergent fan-out from b to z, since signals in the fan-out branch (d, g, f, x, z) were always jointly represented and processed. 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 Figure 6.3. 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 fan-out 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 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 Equation 6.1, using the $fidelity$ algorithm of Figure 5.6 applied to the ideal and faulty primary-output vectors. Table 6.2 shows average runtime and memory usage statistics for test vector sensitivity computation for various input vectors on standard benchmark circuits from the ISCAS85 suite. These simulations were conducted on a Linux workstation with an Intel

```
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]
}
```

Figure 6.3: Algorithm for output-vector computation.

Xeon CPU 2.0GHz processor and cache size 512 KB. Here, the faulty circuit consisted of gates with output bit-flip probabilities of 0.05.

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

Table 6.2: Runtime and memory usage for sensitivity computation for benchmark circuits. Faulty gates all have error probability 0.05 for all inputs.

6.2 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 [49]. In the set cover prob-

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

This connection between the 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. Furthermore, modifying the test-multiset objective simply amounts to altering the ILP objective function. We first describe the case of a single fault in a circuit and then extend our argument to two multiple error assumptions often used in the literature. Then, we give algorithms for multiset test generation with the goal of 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 $sens(f,t)$.

2. Apply t to C $k = \lfloor 1/sens(f,t) \rfloor$ 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 $sens(f,t)$ is higher than the estimate.

4. If $sens(f,t)$ is higher than estimated, we can update our estimate and repeat this process.

In order to extend the above method to multiple faults, we consider two fault 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-error 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 nano-technologies 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 $(1-p)^n \leq 1 - p_{th}$. Figure 6.4 gives the 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 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.

```
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
}
```

Figure 6.4: Greedy algorithm for minimizing the number of test vectors (with repetition) required for fault detection.

Kleinberg and Tardos [49] prove that a similar algorithm for the SET COVER algorithm produces covers that have size within $O(\log(|testmultiset|))$ of the minimal size. Note the runtime is lower-bounded by the size of the multiset, as this is the number of iterations through the **while** loop.

Our ILP problem for minimal test multiset generation is shown in Figure 6.5a. 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, $sens(f_j, t_i)$. If we were to require a minimum detection probability

Minimize $\sum_{i=0}^{m} x_i$ 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 is an integer
(a)

Minimize $\sum_{j=0}^{n} \sum_{i=0}^{m} x_i \times sens(f_j, t_i)$ 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 is an integer
(b)

Figure 6.5: ILP formulations for test-set generation for a fixed number of expected detections: (a) to minimize the number of test vectors required (b) to maximize fault resolution (minimize overlap).

p_{th}, as in Figure 6.4, the constraint that for all f_j, $\prod_j (1 - sens(f_i, t_j)) < 1 - p_{th}$ 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 $x_i \times sens(f_j, t_i)$, i.e., the expected value of a binomial random variable with parameters $(x_i, sens(f_j, t_i))$. Since the expectation is linear, we can add the contributions of all test vectors for each fault f_j as $\sum_i (x_i \times sens(f_j, t_i))$, leading to the constraint in Line 3 of Figure 6.5a. 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 [122]. In randomized rounding, each x_i is rounded up, with a probability equal to the fractional part of x_i.

Assumption 2 generalizes the single-fault case we 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 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 $k/sens(F', t)$ times for k expected

detections. In Table 6.3, we consider a situation in which each gate in the circuit has a small probability of error $p = 10^{-5}$. The difference in the number of repetitions needed between a random vector and the most sensitive test vector is 53.3%, on average. This implies a proportional decrease in test application time.

Circuit	Ave. sensitivity	No. repetitions	Max sensitivity	No. repetitions	% improvement
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
Ave.					53.3

Table 6.3: Number of repetitions required using random vectors versus maximally sensitive test vectors.

In addition, we can 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 Figure 6.5b. 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 size decreases with the expected fault probability.

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 Figure 6.5a 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 Figure 6.4 or solve the ILP shown in Figure 6.5.

Table 6.4 shows the number of test vectors required to detect probabilistic stuck-at faults using the method of Figure 6.4, 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.

	$p_{th} = .05$		$p_{th} = .75$		$p_{th} = .85$		$p_{th} = .95$		$p_{th} = .99$	
Circuit	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
% improv.		64.5		59.7		57.26		53.71		53.05

Table 6.4: Number of test vectors required to detect input signal faults with various threshold probabilities p_{th}. *Rand* is the average number of test vectors selected during random test generation.

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 [29]. In our case, the prior domain knowledge is the expected or modeled fault probabilities in a circuit, and the data comes from testing.

6.3 Summary

Probabilistic faults, like SEUs, require a reformulation of test-generation methods to account for faults occurring with varying probabilities and for test vectors detecting faults with varying probabilities. The same fault can be detected by several test vectors, each of which has its own detection probability or sensitivity with respect to the fault. We called this probability the sensitivity of a test vector to a fault. We showed that test vector sensitivity can be computed by an efficient PTM-based algorithm. We also proposed several test generation and compaction methods for probabilistic faults, with the goals of bounding and estimating fault detection probabilities. These methods use ILP to optimize test sets. Results show that our methods can generate tests quickly and require only half as many (repeated) vectors as random pattern testing.

CHAPTER VII

Conclusions

In this dissertation, we have performed in-depth analysis of non-deterministic behavior in logic circuits, behavior that is due to soft faults and inherent device unreliability. Unlike memories, logic circuits experience several masking mechanisms that can stop the propagation of soft errors. Estimating the circuit soft-error rate (SER) involves carefully analyzing the logical, electrical, and timing masking properties of the design. Beyond soft errors, we analyzed general stochastic behavior in digital circuits. When components in a circuit (gates, at the logic level) behave probabilistically, their effects interact in complex ways and determine overall circuit behavior.

A major computational problem in SER and reliability analysis is the input-vector dependence of the error probability. For a circuit with N inputs, there are 2^N input vectors to consider. In the case of SER estimation, different input vectors can lead to different logically viable paths for error propagation and to different masking conditions. In the case of probabilistic circuits, different input vectors trigger different failure probabilities in gates, which accumulate into different overall probabilities of error. Our SER analyzer, AnSER, and our PTM-based reliability analysis methods offered two different solutions to

this problem. The first solution involved analyzing SER on a random sample of input vectors. Unlike previous methods, which use fault simulation on a vector-by-vector basis, we related the SER to testability measures, like observability and signal probability in logic, and determined these by sampling. Our second solution involved completely enumerating input vectors, using a PTM-based representation of probabilistic circuit behavior.

Our analysis led 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, our SER analyzer can do more than black-box analysis. Observability and node functionality computations done during SER analysis allow us to identify circuit flexibility and inherent redundancy, which can be used to bolster the reliability of the circuit. Further, error-latching window computations can guide physical design changes to improve timing masking, in addition to logic masking.

We also presented methods to test circuits for probabilistic faults, which can be applied to SER testing. SER testing is normally done by irradiating devices with neutron or α-particle sources. We improve on such methods by generating test vectors that maximally sensitize high-impact faults in the design. This is akin to test acceleration by pattern selection rather than irradiation. This type of acceleration can be useful in the field testing of devices to determine the SER in normal operating conditions rather than under heavy, artificially induced radiation. Our main contributions are listed and explained in Section 7.1. We discuss directions for future work in Section 7.2.

7.1 Summary of Contributions

The main contributions of this dissertation are as follows:

- A functional-simulation signature-based SER analyzer for logic circuits, known as AnSER.
- Several logic synthesis techniques that improve the SER of a design, with low area and performance overhead.
- The development of probabilistic-transfer matrix (PTM) algebra into a useful computational technique for CAD.
- 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.

AnSER uses functional-simulation signatures to analyze the SER of logic designs. Simulation signatures are generated by randomly sampling input vectors from a given distribution and propagating resultant logic values throughout the circuit. Observability don't cares (ODCs) are also computed for each node by flipping bits of the signature and propagating changes. Together, signatures and ODCs can be used to estimate testability measures that directly relate to the SER of a circuit. We also incorporated the effects of timing masking with STA-like algorithms to determine the error-latching windows of gates in a circuit. The fraction of the cycle within the error-latching window is a measure of the timing masking capability of the circuit.

We proposed several novel techniques for SER-aware design. Our first technique is known as Signature-based Design for Reliability (SiDeR). Here, signature-comparison is used to identify partial redundancy in circuits. Partially redundant nodes can increase

logic masking with the addition of a small number of gates. Our second technique is known as local rewriting. Here, we generate various alternative, non-redundant topologies for sub-circuits and choose among them for maximal reliability. Third, we propose a technique that targets timing masking instead of logic masking, through a post-placement gate-relocation technique that decreases the size of error-latching windows for critical gates.

We developed a matrix-based algebraic framework, known as the PTM framework, to model stochastic behavior in logic gates. We also introduced several matrix operations that are required for reliability analysis. To improve scalability, PTMs were compressed into algebraic decision diagrams (ADDs). Matrix operations were directly applied to the compressed ADD representations of PTMs. We also presented several heuristics that improve the scalability of the PTM model, including hierarchical computation and input vector sampling.

Finally, we presented a method for testing circuits for multiple probabilistic faults. Using PTMs, we showed how to identify test vectors that are highly affected by errors in the circuit. Then, we select a multiset of test vectors to generate tests optimized, through ILP-based optimization, for the goals of maximizing fault-detection and fault-resolution probabilities. In the next section, we discuss future directions and possible extensions of our work, so as to meet further challenges in design, analysis and test.

7.2 Directions for Future Work

There are several ways to extend our work to accommodate emerging concerns in circuit reliability. First, we propose to continue tackling the central problem of improving the

scalability of exact reliability computations, using edge-valued decision diagrams. Next, we propose to improve the SER of sequential circuits by taking advantage of the increased resynthesis opportunities available. Third, we discuss possible methods of improving SER estimates, by accounting for process variations. Finally, we discuss the possibility of using our probabilistic circuit analysis methods to make design decisions in future device technologies such as QCA and CNTs.

7.2.1 Scalability of PTM-Based Analysis

By enumerating all possible input combinations, PTMs can be used to analyze the error probability of a circuit dependent upon gate-error probabilities. However, their scalability is limited by the exponentially many input vectors that are possible. In Chapter V, we combated this problem by compressing PTMs using ADDs. ADDs are multi-terminal decision diagrams with nodes deciding on matrix row and column variables and terminals encoding real values (probabilities, in the case of PTMs). However, ADDs only recognize structure in the form of identical matrix entries. The size of an ADD can explode if there are many different matrix entries.

Large circuit matrices are constructed by tensoring smaller matrices and eliminating (summing out) common variables. While computing the tensor product of two matrices, pairs of entries, one from each operand, are multiplied together to generate entries in the resultant matrix. Therefore, entries in the resultant matrix can have several common multiplicative factors. One may be able to achieve further compression if each matrix entry is represented as a product of a smaller set of factors. To capture these types of products, we can use an alternative data structure calleda *binary moment diagram (BMD)* [18].

In a BMD, every edge is labeled with a real value; intermediate nodes and terminal nodes can have real values associated with them. A path in a BMD encodes an input-output pair whose value is given by the product of edge-values along the path. Hence, the tensor product of a set of gate matrices amounts to a chaining of the BMDs. The BMD chain avoids the flattening associated with constructing tensor-product ADDs. We believe that BMDs and operations on BMDs should be explored further to improve scaling of exact PTM-based computations. Even if only heuristic methods of error-rate computations are used, it is important to have exact methods against which the heuristic methods can be validated, for reasonably sized circuits.

7.2.2 SER-aware Design of Sequential Circuits

We improved the SER of combinational circuits with the SiDeR technique. However, there may be more opportunities for resynthesis in sequential circuits. For instance, unlike combinational circuits, whose inputs can take on any value, certain states (flip-flop values) can be proven unreachable. Such states can serve as additional don't-cares when searching for implication or equivalence relations between nodes.

Case et al. [21] recently showed that an average of 10% more nodal equivalences can be found by using sequential reachability and sequential observability. These additional equivalent nodes can be used to mask errors, as in SiDeR, through the addition of AND/OR gates. It is likely that the additional number of related signals we find for improving SER will be higher than 10%, since our SiDeR method does not require nodes to be equivalent.

Flip-flops are an additional target for errors in a sequential circuit. Protecting flip-flops from soft errors is vital because latched soft errors cannot be eliminated by electrical

or timing masking. Several techniques have been proposed to harden latches [101] or add extra logic to protect latches [131]. However, these techniques tend to incur high area overhead. We propose to use retiming to enhance logic masking without high area overhead.

Retiming refers to the process of relocating flip-flops, usually to minimize the area or the clock period in a given circuit [64]. The minimum-area and minimum-period retiming problems can both be formulated as linear programs (LPs) and solved using the simplex method. Therefore, exact optima are achievable using retiming. The idea of SER-aware retiming is to stop error propagation from latches to any primary outputs by relocating latches to regions of low observability. Recall that sequential observability is calculated by transforming latches into buffers and expanding the circuit by several time frames. In this context, the signatures and ODC masks of the buffers corresponding to latches can be computed in the same way as any other gate. Therefore, existing retiming formulations can be modified to jointly optimize for either period and SER or area and SER.

7.2.3 Impact of Process Variations on Soft Errors

There are three main sources of variation in transistor behavior that result from the difficulty of controlling parameters during manufacture. These are 1) dopant fluctuations, 2) gate length and wire width variations, and 3) varying heat flux across the chip [16]. First, the number of dopant atoms in the channel of a transistor decreases quadratically with transistor size. For instance, at the 1um technology node, transistors have thousands of dopant atoms [15]. In the 32nm technology node 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 65nm technology node are being manufactured with a 157nm lithography wavelength, causing variations of up to 30% [16]. Variations in wire widths normally increase delay, but variations in channel width can also change the V_t. Third, changes in the switching activity of a chip can cause varying heat flux and "hot spots" in certain areas of the chip. These hot spots can lead to sudden drops in power-supply voltage.

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 [28]. If the lower dopant level causes higher V_t, it can sometimes lead to a higher rate of SEU occurrence (though lower rate of propagation), because of increased charge-collection efficiency for particle strikes. If the gate length is increased, then this exposes more area for particles to strike, causing more SEUs [112]. These effects often counterbalance each other—although not perfectly. Therefore, it becomes important to model these effects probabilistically in the context of SER. Also, delay variability, especially due to dynamic changes in heat flux, can cause differences in timing masking. The results of statistical-timing analysis must be translated into timing masking in order to accurately estimate the SER.

Interestingly, it may be possible to use accelerated-SER tests to actually determine the static process parameters (such as V_t and gate length) of a circuit. 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 Chapter VI, may be able to pinpoint locations with anomalous behavior.

7.2.4 Probabilistic Analysis of New Technologies

In our work, we have analyzed circuits probabilistically, without making assumptions about the actual device technology. However, more specific types of analysis can be carried out for particular technologies such as QCA and CNTs.

One unique aspect of QCA is that both wires and gates are constructed from quantum dots. Different arrangements of quantum dots can yield different Boolean functions or equivalent functions with different fault-tolerance properties. Therefore, layout, functionality, and reliability are intimately linked in QCA design. Generally, QCA are designed modularly, with fixed tile sizes and basic gates. The SQUARES design paradigm [11] proposes 5×5 tiles; others have proposed 3×3 tiles, orthogonal tiles, etc. Universal gates can be synthesized in various ways from these tiles. Generally, the MAJORITY gate, the NAND-NOR-INVERTER gate, or the AND-OR-INVERTER gate are used as universal gates [110]. Due to the high propensity for error in QCA, each of these structures can be analyzed and selected for its error tolerance. Dysart et al. [30] have already adopted our PTM framework to analyze the reliability of adders and other small QCA modules. However, the analysis can be carried much further to analyze layouts, wire shapes, and possible gate libraries.

CNT circuits are also generally constructed as regular arrays of gates. The crossbar architecture has often been considered in literature. However, recently, other combinational logic blocks have been studied, including multi-valued logic gates, single-transistor XOR

gates, etc. [66]. Multi-valued gates are realized by varying the diameter of the nanotube to change the threshold voltage [105]. Circuits constructed from such gates need built-in fault tolerance to sustain reliable operation. The optimal number of logic values and acceptable levels of SER can be determined by either AnSER or PTM-based analysis.

In closing, we have presented methods for analyzing, designing, and testing circuits that are subject to non-deterministic effects. We hope that our work provides insights and stimulates future research. Research in this area will be vital to enabling advancements in device technology and improving IC performance.

BIBLIOGRAPHY

[1] Berkeley Logic Synthesis and Verification Group, "ABC: A System For Sequential Synthesis & Verification," http://www.eecs.berkeley.edu/∼alanmi/abc/

[2] Z. Alkhalifa, V.S.S. Nair, N. Krishnamurthy, J. A. Abraham, "Design and Evaluation of System-Level Checks for On-line Control Flow Error Detection," *IEEE Trans. on Parallel and Distributed Systems*, 1999, vol. 10, no. 6, pp. 627-641.

[3] S. Almukhaizim, et al., "Seamless Integration of SER in Rewiring-Based Design Space Exploration," *Proc. ITC*, 2006, pp. 1-9.

[4] D. Alexandrescu, L. Anghel, M. Nicolaidis, "New Methods for Evaluating the Impact of Single Event Transients in VDSM ICs," *Proc. DFT*, 2002, pp. 99-107.

[5] C. J. Alpert, A. Devgan, C. Kashyap, "A Two Moment RC Delay Metric for Performance Optimization," *Proc. ISPD*, 2000, pp. 69-74.

[6] D. N. Armstrong, H. Kim, O. Mutlu, Y. N. Patt,"Wrong Path Events: Exploiting Unusual and Illegal Program Behavior for Early Misprediction Detection and Recovery," *Proc. MICRO*, 2004, pp. 119-128.

[7] T. Austin,"DIVA: A Reliable Substrate for Deep Submicron Microarchitecture Design," *Proc. Micro*, 1999, pp. 196-207.

[8] R.I. Bahar, et al., "Algebraic Decision Diagrams and their Applications," *Journal of Formal Methods in System Design*, 1997, vol. 10, no. 2/3, pp. 171-206.

[9] M.P. Baze, S.P. Buchner, D. McMorrow, " A digital CMOS design technique for SEU hardening,"*IEEE Trans. on Nuclear Science*, 2000, vol. 47, no. 6, part 3, pp. 2603-2608.

[10] M.P. Baze, S.P. Buchner,"Attenuation of Single Event Induced Pulses in CMOS Combinational Logic," *IEEE Trans. on Nuclear Science*, 1997, vol. 44, no. 6, part 1, pp. 2217-2223.

[11] D. Berzon, T.J. Fountain, "A Memory Design in QCAs using the SQUARES Formalism," *Proc. GLSVLSI*, 1999, pp. 166-169.

[12] D. Bhaduri, S. Shukla, " NANOLAB-A Tool for Evaluating Reliability of Defect-tolerant Nanoarchitectures," *IEEE Trans. On Nanotechnology*, 2005, vol. 4, no. 4, pp. 381-394.

[13] A. Biswas, et al., "Computing Architectural Vulnerability Factors for Address-Based Structures," *Proc. ISCA*, 2005, pp. 532-543.

[14] R.I. Bahar, J. Mundy, J. Chan, "A Probabilistic Based Design Methodology for Nanoscale Computation," *Proc. ICCAD*, 2003, pp. 480-486.

[15] S. Borkar, et al., "Parameter Variations and Impact on Circuits and Microarchitecture,"*Proc. DAC*, 2003, pp. 328-342.

[16] S. Borkar, "Designing Reliable Systems from Unreliable Components: the Challenges of Transistor Variability and Degradation," *IEEE Trans. on Microarchitecture*, 2005, vol. 25, no. 6, pp. 10-16.

[17] D. Brand, "Verification of Large Synthesized Designs," *Proc. ICCAD*, 1993, pp. 534-537.

[18] R. Bryant, O. A. Chen "Verification of Arithmetic Functions with Binary Moment Diagrams," *Proc. DAC*, 1995, pp. 535-541.

[19] M. Bushnell, V. Agrawal, *Essentials of Electronic Testing*, Kluwer, 2000.

[20] A. Caldwell, A. Kahng, I. Markov, "Can Recursive Bisection Alone Produce Routable Placements?," *Proc. DAC 2000*, pp. 693-698.

[21] M. L. Case, V. N. Kravets, A. Mishchenko, R. K. Brayton, "Merging Nodes under Sequential Observability," *Proc. DAC*, 2008, pp. 540-545.

[22] S. Chen, J. Xu, "Modeling and Evaluating the Security Threats of Transient Errors in Firewall Software," *Proc. DSN-PDS*, 2002, pp. 53-72.

[23] H. Cho, S.-W. Jeong, F. Somenzi, C. Pixley, "Synchronizing Sequences and Symbolic Traversal Techniques in Test Generation," *Journal of Electronic Testing*, vol. 4, 1993, pp. 19.31.

[24] M. Choudhury, K. Mohanram, "Accurate and Scalable Reliability Analysis of Logic Circuits," *Proc. DATE*, 2007, pp. 1454-1459.

[25] C. Chu, Y. -C. Wong, "Fast and Accurate Rectilinear Steiner Minimal Tree Algorithm for VLSI Design," *Proc. ISPD*, 2005, pp. 28-35.

[26] E. Clarke, et al., "Multi-Terminal Binary Decision Diagrams and Hybrid Decision Diagrams," T. Sasao, M. Fujita, eds., *Representations of Discrete Functions*, Kluwer, 1996, pp. 93-108.

[27] T. Cormen, C. Lieserson, R. Rivest, C. Stein, *Introduction to Algorithms*, MIT Press, 2001, pp. 331-338.

[28] V. Degalahal, et al., "The Effect of Threshold Voltages on the Soft Error Rate," *Proc. ISQED*, 2004, pp. 503-508.

[29] M. DeGroot, *Optimal Statistical Decisions*, McGraw-Hill, 1970.

[30] T.J. Dysart, P. M. Kogge, "Probabilistic Analysis of a Molecular Quantum-Dot Cellular Automata Adder," *Proc. DFT*, 2007, pp. 478-486.

[31] S. Egner, M. Püschel, Th. Beth, "Decomposing a Permutation into a Conjugated Tensor Product," *Proc. International Symposium on Symbolic and Algebraic Computation*, 1997, pp. 101-108.

[32] S. Ercolani, et al., "Estimate of Signal Probability in Combinational Logic Networks," *Proc. European Test Conference*, 1989, pp. 132-138.

[33] D. Ernst, et al., "Razor: Circuit-Level Correction of Timing Errors for Low Power Operation," *IEEE Micro*, 2003, vol. 24, no. 6, pp. 10-20.

[34] A. Grove, "Changing Vectors of Moore's Law,"*International Electronic Devices Meeting*, 2002, http://www.intel.com/research/silicon/mooreslaw.htm.

[35] Genie SMILE software, http://genie.sis.pitt.edu/.

[36] G. D. Hachtel, E. Macii, A. Pardo, F. Somenzi, "Markovian Analysis of Large Finite State Machines," *IEEE Trans. on CAD*, 1996, vol. 15, pp. 1479-1493.

[37] G. Hachtel, F. Somenzi, *Logic Synthesis and Verification Algorithms*, Kluwer Academic Publishers, 1996, pp. 309.

[38] J. Han, P. Jonker, "A System Architecture Solution for Unreliable Nanoelectronic Devices," *IEEE Trans. on Nanotechnology*, 2002, vol. 1, pp. 201-208.

[39] J. P. Hayes, I. Polian, B. Becker, "An Analysis Framework for Transient-Error Tolerance," *Proc. VTS*, 2007, pp. 249-255.

[40] D. F. Heidel, et al., "Alpha-particle Induced Upsets in Advanced CMOS Circuits and Technology," *IBM Journal of Research and Development*, 2008, vol. 52, no. 3, pp. 225-232.

[41] D. F. Heidel, et al., "Single-Event-Upset Critical Charge Measurements and Modelling of 65nm Silicon-on-Insulator Latches and Memory Cells," *IEEE Trans. on Nuclear Science*, 2006, vol. 53, no. 6, pp. 3512-3517.

[42] J. Huang, M. Momenzadeh, F. Lombardi, "Defect Tolerance of QCA Tiles," *Proc. DATE*, 2006, pp. 1-6.

[43] ILOG CPLEX: High-performance Software for Mathematical Programming and Optimization, http://www.ilog.com/products/cplex.

[44] IROC Technologies, "White Paper on VDSM IC Logic and Memory Signal Integrity and Soft Errors," 2002, http://www.iroctech.com.

[45] F. Irom, F. F. Farmanesh,"Frequency Dependence of Single-Event Upset in Advanced Commercial PowerPC Microprocessors," *IEEE Trans. on Nuclear Science*, 2004, vol. 51, issue 6, part 2, pp. 3505-3509.

[46] IWLS05 Benchmarks, http://iwls.org/iwls2005/benchmarks.html.

[47] R. W. Keyes, R. Landauer, "Minimal Energy Dissipation in Computation," *IBM Journal of Research and Development*, 1970, vol. 14, no. 152.

[48] L. B. Kish, "The End of Moores law: Thermal (Noise) Death of Integration in Micro and Nano Electronics," *Physical Review Letters A*, 2002, v. 305, pp. 144-149.

[49] J. Kleinberg, E. Tardos, *Algorithm Design*, Addison-Wesley, 2005, pp. 612-617.

[50] H. Kobayashi, et al., "Comparison between Neutron-induced System-SER and Accelerated-SER in SRAMs," *Proc. International Reliability Physics Symposium*, 2004, pp. 288-293.

[51] B. Konemann, "LFSR-Coded Test Patterns for Scan Designs," *Proc. European Test Conference*, 1991, pp. 237-242.

[52] P. Korkmaz, B. E. S. Akgul, L. N. Chakrapani, K. V. Palem, "Advocating Noise as an Agent for Ultra Low-Energy Computing: Probabilistic CMOS Devices and Their Characteristics," *Japanese Journal of Applied Physics*, 2006, vol. 45, no. 4B, pp. 3307-3316.

[53] S. Krishnaswamy, I. L. Markov and J. P. Hayes, "Testing Logic Circuits for Transient Faults," *Proc. ETS 2005*, pp. 102-107.

[54] S. Krishnaswamy, I. L. Markov, J. P. Hayes, "Tracking Uncertainty with Probabilistic Logic Circuit Testing," *IEEE Design and Test*, 2007, vol. 24, no. 4, pp. 312-321.

[55] S. Krishnaswamy, I. L. Markov, J. P. Hayes, "On the Role of Timing Masking in Reliable Logic Circuit Design," *Proc. DAC*, 2008, pp. 924-929.

[56] S. Krishnaswamy, I. L. Markov, J. P. Hayes, "When Are Multiple Gate Errors Significant in Logic Circuits?" *SELSE Workshop*, 2006.

[57] S. Krishnaswamy, S. M. Plaza, I. L. Markov, J. P. Hayes, "AnSER: A Lightweight Reliability Evaluator for use in Logic Synthesis," *Digest IWLS*, 2007, pp. 171-173.

[58] S. Krishnaswamy, S. M. Plaza, I. L. Markov, J. P. Hayes, "Enhancing Design Robustness with Reliability-aware Resynthesis and Logic Simulation," *Proc. ICCAD*, 2007, pp. 149-154.

[59] S. Krishnaswamy, S. M. Plaza, I. L. Markov, J. P. Hayes, "Signature-based SER analysis and Design of Logic circuits," *to appear in IEEE Trans. on CAD*.

[60] S. Krishnaswamy, G. F. Viamontes, I. L. Markov, J. P. Hayes, "Accurate Reliability Evaluation and Enhancement via Probabilistic Transfer Matrices," *Proc. DATE*, 2005, pp. 282-287.

[61] S. Krishnaswamy, G. F. Viamontes, I. L. Markov, J. P. Hayes, "Probabilistic Transfer Matrices in Symbolic Reliability Analysis of Logic Circuits,"*ACM Trans. on Design Automation of Electronic Systems*, 2008, vol. 13, no. 1, article 8.

[62] R. K. Kummamuru, et al. "Operation of Quantum-Dot Cellular Automata (QCA), Shift Registers and Analysis of Errors," *IEEE Trans. on Electron Devices*, 1993, vol. 50-59, pp. 1906-1913.

[63] H. K. Lee, D. S. Ha, "On the Generation of Test Patterns for Combinational Circuits," TR No. 12-93, EE Dept., Virginia Polytechnic Institute.

[64] C.E. Leiserson, J. B. Saxe, "Retiming Synchronous Circuitry," *Algorithmica*, 1991, vol. 6, pp. 5-35.

[65] M-L. Li, "Understanding the Propagation of Hard Errors to Software and Implications for Resilient System Design," *Proc. ASPLOS*, 2008, pp. 265-276.

[66] J. Liu, I. OConnor, D. Navarro, F. Gaffiot, "Novel CNTFET-based Reconfigurable Logic Gate Design," *Proc. DAC*, 2007, pp. 276-277.

[67] F.T. Luk, H. Park, "An Analysis of Algorithm-Based Fault Tolerance Techniques," *Journal of Parallel and Distributed Computing*, 1988, vol. 5, no. 2, pp. 172 - 184.

[68] J.P. Marques-Silva, T. Glass, "Combinational Equivalence Checking Using Satisfiability and Recursive Learning," *Proc. DATE*, 1999, pp. 145-149.

[69] R. Martel, et al., "Carbon Nanotube Field- Effect Transistors and Logic Circuits," *Proc. DAC*, 2002, pp. 94-98.

[70] T.C. May, M.H. Woods,"Alpha-Particle-Induced Soft Errors in Dynamic Memories," *IEEE Trans. on Electron Devices*, 1979, vol. 26, pp. 2-9.

[71] G.M. Megson, D.J. Evans, "Algorithmic Fault Tolerance of Matrix Operations on Triangular Arrays," *Journal of Parallel Computing*, 1989, vol. 10, no.2, pp. 207-219.

[72] C. Metra, S. Di Francescantonio, G. Marrale, "On-line Testing of Transient Faults Affecting Functional Blocks of FCMOS, Domino and FPGA-implemented Self-Checking Circuits," *Proc. DFT*, 2002, pp. 207-215.

[73] A. Mishchenko, S. Chatterjee, R. Brayton, "DAG-aware AIG rewriting: A Fresh Look at Combinational Logic Synthesis," *Proc. DAC*, 2006, pp. 532-535.

[74] N. Miskov-Zivanov, D. Marculescu, "MARS-C: Modeling and Reduction of Soft Errors in Combinational Circuits," *Proc. DAC*, 2006, pp. 767-772.

[75] N. Miskov-Zivanov, D. Marculescu, "Soft Error Rate Analysis for Sequential Circuits," *Proc. DATE*, 2007, pp. 1436-1441.

[76] K. Mohanram, N. A. Touba, "Partial Error Masking to Reduce Soft Error Failure Rate in Logic Circuits," *Proc. DFT*, 2003, pp. 433-440.

[77] K. Mohanram, "Simulation of Transients caused by Single-Event Upsets in Combinational Logic," *Proc. ITC*, 2005.

[78] Y. Monnet, et al., " Case Study of a Fault Attack on Asynchronous DES Crypto-Processors," *Lecture Notes in Computer Science*, 2006, vol. 4236, 88-97.

[79] J. Muir,"Seiferts RSA Fault Attack: Simplified Analysis and Generalizations," *Lecture Notes in Computer Science*, 2006, vol. 4307, pp. 420-434.

[80] S.S. Mukherjee, et al., "A Systematic Methodology to Compute the Architectural Vulnerability Factors for a High-Performance Microprocessor," *Proc. MicroArch*, 2003, pp. 29-40.

[81] S.S. Mukherjee, J. Emer, S.K. Reinhardt, "The Soft Error Problem: An Architectural Perspective," *Proc. HPCA*, 2005, pp. 243-247.

[82] S.S. Mukherjee, M. Kontz, S.K. Reinhardt,"Detailed Design and Evaluation of Redundant Multithreading Alternatives," *Proc. ISCA*, 2002, pp. 99-110.

[83] D. Mukhopadhyay, D. RoyChowdhury, "Fault Based Attack on the Rijndael Cryptosystem," *Journal of Discrete Mathematical Sciences & Cryptography*, 2007, vol. 10, no. 2, pp 267-290.

[84] O. Musseau, et al., "Analysis of multiple bit upsets (MBU) in CMOS SRAM," *IEEE Trans. on Nuclear Science*, 1996, vol. 43, no. 6, pp. 2879-2888.

[85] K. Natori, N. Sano, "Scaling Limit of Digital Circuits due to Thermal Noise," *Journal of Applied Physics*, 1998, vol. 83, pp. 5019-5024.

[86] K. Nepal, et al., "Designing Logic Circuits for Probabilistic Computation in the Presence of Noise," *Proc. DAC*, 2005, pp. 485-490.

[87] M. Nicolaidis, "Time Redundancy Based Soft-Error Tolerance to Rescue Nanometer Technologies," *Proc. VTS*, 1999, pp. 86-94, 1999.

[88] M. Omana, et al., "A Model for Transient Fault Propagation in Combinatorial Logic," *Proc. IOLTS*, 2003, pp. 11-115.

[89] Openbayes software, http://www.openbayes.org/.

[90] K. V. Palem, "Energy Aware Computing through Probabilistic Switching: A Study of Limits," *IEEE Trans. on Computing*, 2005, vol. 54, no. 9, pp. 1123-1137.

[91] K.P. Parker, E.J. McCluskey, "Probabilistic Treatment of General Combinational Networks," *IEEE Trans. on Computers*, 1975, vol. C-24, no. 6, pp. 668-670.

[92] S. Plaza, K-H. Chang, I. Markov, V. Bertacco, "Node Mergers in the Presence of Don't Cares," *Proc. ASP-DAC*, 2007, pp. 414-419.

[93] V. Pless, *Introduction to the Theory of Error-Correcting Codes*, John Wiley & Sons, 1982.

[94] N. Pippenger, "Reliable Computation by Formulas in the Presence of Noise," *IEEE Trans. on Information Theory*, 1998, vol. 34, no. 2, pp. 194-197.

[95] I. Polian, et al., "Transient Fault Characterization in Dynamic Noisy Environments," *ITC 2005*, pp. 1039-1048.

[96] J. M. Rabey, A. Chankrakasan, B. Nikolic, *Digital Integrated Circuits*, Prentice Hall, 2003. pp. 445-490.

[97] P. Racunas, K. Constantinides, S. Manne, S.S. Mukherjee, "Perturbation-based Fault Screening," *Proc. HPCA*, 2007, pp. 169-180.

[98] P. Ramachandran, et al., "Statistical Fault Injection," *DSN*, 2008.

[99] A. Ramalingam, et al., "An Accurate Sparse Matrix Based Framework for Statistical Static Timing Analysis,"*Proc. ICCAD*, 2006, pp. 231-236.

[100] R. Rao, K. Chopra, D. Blaauw, D. Sylvester, "An Efficient Static Algorithm for Computing the Soft Error Rates of Combinational Circuits," *Proc. DATE*, 2006, pp. 164-169.

[101] R. Rao, D. Blaauw, D. Sylvester, "Soft Error Reduction in Combinational Logic using Gate Resizing and Flip-flop Selection," *Proc. ICCAD*, 2006, pp. 502-509.

[102] A.L.N. Reddy, P. Banerjee, "Algorithm-Based Fault Detection for Signal Processing Applications," *IEEE Trans. on Computers*, 1990, vol. 39, no. 10, pp. 1304-1308.

[103] S. Reddy, I. Pomeranz, S. Kajihara, "Compact Test Sets for High Defect Coverage," *IEEE Trans. on CAD*, 1997, vol.16, no. 8, pp. 923-930.

[104] T. Rejimon, S. Bhanja, "Probabilistic Error Model for Unreliable Nano-logic Gates," *Proc. NANO*, 2006, pp. 47-50.

[105] Raychodhury, K. Roy, "Carbon-Nanotube-Based Voltage-Mode Multiple-Valued Logic Design," *IEEE Trans. on Nanotechnology*, 2005, vol. 4, no. 2, pp. 168-179.

[106] K. Rodbell, et al., "Low-Energy Proton-Induced Single-Event-Upsets in 65 nm Node, Silicon-on-Insulator, Latches and Memory Cells," *IEEE Trans. on Nuclear Science*, 2007, vol. 54, no. 6, pp. 2474-2479.

[107] E. Rotenberg, "AR-SMT: A Microarchitectural Approach to Fault Tolerance in Microprocessor," *Proc. Fault-Tolerant Computing Systems*, 1999, pp. 84-91.

[108] A. Sanyal, K. Ganeshpure, S. Kundu, "On Accelerating Soft-Error Detection by Targeted Pattern Generation," *Proc. ISQED*, 2008, pp. 723-728.

[109] J. Savir, G. Ditlow, P.H. Bardell, "Random Pattern Testability," *Proc. IEEE Symposium on Fault Tolerant Computing*, 1983, pp. 80-89.

[110] B. Sen, B. K. Sikdar, "A Study on Defect Tolerance of Tiles Implementing Universal Gate Functions," *Proc. DTIS*, 2007, pp. 13-18.

[111] N. Seifert, et al., "Frequency Dependence of Soft Error Rates for Sub-Micron CMOS Technologies," *International Electron Devices Meeting Technical Digest*, 2001, pp. 14.4.1 - 14.4.4.

[112] N. Seifert, X. Zhu, L.W. Massengill, "Impact of Scaling on Soft-error Rates in Commercial Microprocessors," *IEEE Trans. on Nuclear Science*, 2002, vol. 49, no. 6, part 1, pp. 3100-3106.

[113] D.P. Siewiorek, R.S. Swarz, *Reliable Computer Systems*, 2nd ed., Digital Press, 1992.

[114] P. Shivakumar, et al., "Modeling the Effect of Technology Trends on Soft Error Rate of Combinational Logic," *Proc. ICDSN*, 2002, pp. 389-398.

[115] S.K. Shukla, R.I. Bahar, *Nano, Quantum and Molecular Computing,* Springer-Verlag, 2004, chapter 2.

[116] A. Solomatnikov, et al., "Skewed CMOS: Noise-Tolerant High-Performance Low-Power Static Circuit Family," *IEEE Trans. on VLSI*, 2002, vol. 10, no. 4, pp. 469-476.

[117] F. Somenzi, "CUDD: CU Decision Diagram Package," http://vlsi.colorado.edu/fabio/CUDD.

[118] A.D. Tipton, et al., "Multiple-Bit Upset in 130 nm CMOS Technoogy," *IEEE Trans. on Nuclear Science,* 2006, vol. 53, no 6, pp. 3259-3264.

[119] K.H. Tsai, et al.,"STARBIST: Scan Autocorrelated Random Pattern Generation," *Proc. DAC*, 1997, pp. 472-477.

[120] J. G. Tryon, "Quadded Logic," in W.C. Mann, R. H. Wilcox Ed.*Redundancy Techniques for Computing Systems*, 1962, pp. 205-228.

[121] UMICH Physical Design Tools http://vlsicad.eecs.umich.edu/BK/PDtools/.

[122] V.V. Vazirani, *Approximation Algorithms,* Springer-Verlag, New York, 2001.

[123] G.F. Viamontes, I.L. Markov, J. P. Hayes, "Improving Gate-Level Simulation of Quantum Circuits," *Quantum Information Processing*, 2003, vol. 2, no. 5, pp. 347-380.

[124] R. Venkatasubramanian, J.P. Hayes, B.T. Murray, "Low-cost On-line Fault Detection Using Control Flow Assertions," *Proc. IOLTS*, 2003, pp. 137-143.

[125] J. von Neumann,"Probabilistic Logics & Synthesis of Reliable Organisms from Unreliable Components," *Automata Studies*, 1956, pp. 43-98.

[126] C. Weaver, T. Austin, "A Fault Tolerant Approach to Microprocessor Design," *Proc. DSN*, 2001, pp. 411-420.

[127] J. Wilkinson, S. Hareland,"A Cautionary Tale of Soft Errors Induced by SRAM Packaging Materials," *IEEE Trans. on Device and Materials Reliability*, 2005, vol. 5 no. 3, pp. 448-433.

[128] S. Yajnik, N.K. Jha, "Analysis and Randomized Design of Algorithm-Based Fault Tolerant Multiprocessor Systems under an Extended Model," *IEEE Trans. on Parallel and Distributed Systems*, 1997, vol. 8, no. 7, pp. 757-768.

[129] J. F. Ziegler, et al., "IBM Experiments in Soft Fails in Computer Electronics (1978-1994)," *IBM Journal of Reearch and Development*, 1996, vol. 40, No. 1.

[130] J. F. Ziegler, et al., "Terrestrial Cosmic Rays," *IBM Journal of Reearch and Development*, 1996, vol. 40, No. 1.

[131] M. Zhang, et al. "Sequential Element Design with Built-In Soft Error Resilience," *IEEE Trans. VLSI*, 2006, pp. 1368-1378.

[132] M. Zhang, N. Shanbhag, "A CMOS Design Style for Logic Circuit Hardening," *Proc. International Reliability Physics Symposium*, 2005, pp. 223-229.

[133] M. Zhang, N. R. Shanbhag, "Dual-Sampling Skewed CMOS Design for Soft-Error Tolerance," *IEEE Trans. on Circuits and Systems II: Express Briefs*, 2006, vol. 53, no. 12, pp. 1461-1465.

[134] M. Zhang, N. R. Shanbhag, "A Soft Error Rate Analysis (SERA) Methodology," *Proc. ICCAD*, 2004, pp. 111-118.

[135] B. Zhang, W. S. Wang, M. Orshansky, "FASER: Fast Analysis of Soft Error Susceptibility for Cell-Based Designs," *Proc. ISQED*, 2006, pp. 755-760.

[136] Q. Zhou, K. Mohanram, "Gate Sizing to Radiation Harden Combinational Logic," *IEEE Trans. on CAD*, 2006, vol. 25, no. 1, pp. 155-166.

[137] Q. Zhu, N. Kitchen, A. Kuehlmann, A. Sangiovanni-Vincentelli, "SAT sweeping with Local Observability Don't-cares," *Proc. DAC*, 2006, pp. 229-234.