RICE UNIVERSITY

Logic design for reliability

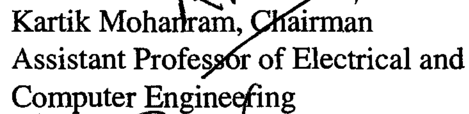
by

Mihir Choudhury

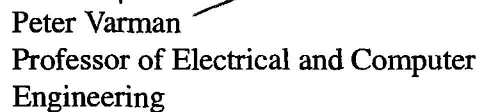
A THESIS SUBMITTED
IN PARTIAL FULFILLMENT OF THE
REQUIREMENTS FOR THE DEGREE

Master of Science

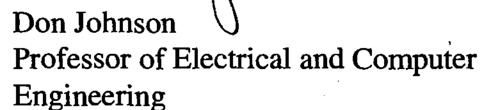
APPROVED, THESIS COMMITTEE:



Kartik Mohanram, Chairman
Assistant Professor of Electrical and Computer Engineering



Peter Varman
Professor of Electrical and Computer Engineering



Don Johnson
Professor of Electrical and Computer Engineering

HOUSTON, TEXAS

OCTOBER, 2007

UMI Number: 1455230

INFORMATION TO USERS

The quality of this reproduction is dependent upon the quality of the copy submitted. Broken or indistinct print, colored or poor quality illustrations and photographs, print bleed-through, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.

UMI®

UMI Microform 1455230

Copyright 2008 by ProQuest LLC.

All rights reserved. This microform edition is protected against unauthorized copying under Title 17, United States Code.

ProQuest LLC
789 E. Eisenhower Parkway
PO Box 1346
Ann Arbor, MI 48106-1346

Abstract

Logic design for reliability

by

Mihir Choudhury

Reliability of logic circuits is emerging as an important concern that may limit the benefits of continued scaling of process technology and the emergence of future technology alternatives. Scaling of CMOS devices below 100nm has revealed their vulnerability to process variations, thermodynamic variations, transient errors due to radiation, electromagnetic interference and extreme scaling effects. Variations cause the output of gates to deviate from the correct value potentially leading to logic errors. The search for new devices to replace CMOS in the future has led to advances in the synthesis and self-assembly of nanoelectronic devices like carbon nanotube transistors that indicate the ability to manufacture dense nanoelectronic fabrics. However, the tremendous device densities afforded by nanoelectronic technologies is expected to be accompanied by substantial increases in defect densities, transient error rates, and performance variability. Thus, high failure rates are inherent to computing devices of the future and have led to an increased interest in investigating the potential of logic design techniques to improve circuit reliability. This thesis contributes to two major aspects of logic design for circuit reliability:

- Computing the reliability of logic circuits built with unreliable devices.
- Detection of errors in logic circuits based on approximate logic functions.

Acknowledgements

I would like to thank Kartik for useful insights, suggestions, timely discussions, and trip to France :).

Contents

Abstract ii

Acknowledgements iii

List of Figures vi

List of Tables viii

1 Introduction 1

2 Reliability analysis 6

2.1 Observability-based reliability analysis 9

2.1.1 Example 13

2.1.2 Noise and correlation distort observability 14

2.2 Single-pass reliability analysis 16

2.2.1 Handling reconvergent fanout 20

2.2.2 Accuracy vs. computation complexity 22

2.3 Maximum-k gate failure reliability analysis 24

2.3.1 Why fault injection and simulation fails ? 25

2.3.2 Probability distribution of gate failures 26

2.3.3 Generating a sample of erroneous gates 27

2.3.4 Effect of multiple gate failures 31

2.4 Results 32

2.4.1 Observability-based reliability analysis 32

2.4.2 Single-pass reliability analysis 33

2.4.3 Maximum-k gate failure model 35

2.5 Applications 36

3 Approximate logic functions **39**

3.1 Concurrent error detection 39

3.2 Synthesis of approximate logic functions 42

3.2.1 Type assignment 44

3.2.2 Cube selection 45

3.2.3 Iterative cube selection algorithm 49

3.3 Approximate logic circuits for CED 51

3.3.1 Conventional CED 51

3.3.2 CED for speed-paths 51

3.3.3 Low overhead intrusive CED 52

3.4 Self-checking checker design 52

3.5 Results 53

3.5.1 Conventional CED 54

3.5.2 CED for speed-paths 55

3.5.3 Technology-independence 56

4 Conclusions and future work **58**

List of Figures

2.1 Effect of noise on the output voltage of a gate. 6

2.2 Reliability of a logic circuit illustrated with the help of an example. 7

2.3 Example circuit to illustrate of observability-based closed-form expression . 14

2.4 Circuit for illustrating the effect of noise and correlation on observability . 14

2.5 Illustration of single-pass reliability analysis 19

2.6 Computation/propagation of correlation factor 21

2.7 Derivation of $\Pr(l_{0\rightarrow1}|k_{0\rightarrow1})$ in terms of correlation factors of its inputs. . . 22

2.8 Handling reconvergent fanout in single-pass reliability analysis with 4 correlation factors. 23

2.9 Proposed sampling algorithm 29

2.10 Figure shows the circuit structures that introduce (a) first and (b) higher order correlation factors. 34

2.11 $\delta(\vec{\epsilon})$ curves for two outputs of i10. Average error in $\delta(\vec{\epsilon})$ per output of c499 over 1000 runs. On each run, $\epsilon_i \in \mathrm{Uniform}(0, 0.5)$ for each gate. 34

2.12 Redundancy-free improvements in reliability 37

3.1 Basic design for concurrent error detection. 39

3.2 Representation of SOPs in the 0/1 phase. 43

3.3 This example illustrates the two cube selection algorithms for synthesis of approximate logic circuits (Figs. b and c). It also illustrates synthesis driven towards high CED coverage on speed-paths (Fig. d). 46

3.4 Figure illustrates the application of approximate logic functions to concurrent error detection. 50

3.5 Self-checking checker design for the proposed error detection technique. . . 53

List of Tables

2.1 Expressions for weighted input error components 18

2.2 Sample space of possible gate failures 27

2.3 Proposed sampling algorithm table. 31

2.4 Table shows a comparison between the observability-based reliability analysis and Monte Carlo for 10 benchmark circuits. Five values of ϵ have been used for comparison. 33

2.5 Comparison of accuracy and runtimes of single-pass reliability analysis with 0, 4 and 16 correlation factor. 35

2.6 Runtimes for different benchmark circuits under maximum-k gate failure model. A total of 50K samples were used to evaluate the reliability. 36

3.1 Approximation percentage and CED coverage for output cones extracted from benchmark circuits. 54

3.2 Approximate percentage and CED coverage for MCNC benchmark circuits. 55

3.3 Minterm coverage achieved for CED on speed-paths. 56

3.4 Technology-independence of CED coverage 57

Chapter 1

Introduction

The problem of noise affecting computation has been a concern since the beginning of the computer age 50 years ago. The advent of the microelectronics era, which promised highly reliable MOS transistors, evaded this problem to a large extent. Using MOS transistors, reliable integrated circuits, with very low failure probabilities, could be fabricated. The main issue with MOS circuits were manufacturing defects like stuck-at and bridging faults. These faults could be detected using post-manufacturing test methods. However, scaling of CMOS devices below 100nm has posed a new challenge in circuit design, i.e. variability. Slight variations in the parameters of sub-100nm devices can cause significant variations in design objectives like performance, power and reliability [4, 7]. Sources of variations include:

- *Manufacturing process variations*: Limitations in the manufacturing process cannot guarantee accurate fabrication of sub-100nm devices. Process variations include variation in impurity concentration density, oxide thickness, diffusion depth, etc. As a result of these variations, accurate control over the device parameters like threshold voltage is lost, thus leading to degradation in critical device specifications like performance and power dissipation. This leads to manufactured chips being slower or dissipating more power. Thus, manufacturing yield is adversely affected as a result of process variations.

- *Single-event upsets*: In addition to variability and manufacturing defects, single-event upsets, caused due to high energy particle strike (e.g. neutrons, α-particles), can corrupt the state of a transistor. Soft errors occur as a result of single-event upsets (SEUs) if the transient error propagates to a primary output. Scaling of CMOS transistors increases their vulnerability to single-event upsets because a lower energy particle strike can upset the transistor state. Although soft errors cause no permanent damage, they can severely limit the reliability of electronic systems.

- *Transistor aging*: New failure mechanisms such as transistor wear-out due to negative bias temperature instability (NBTI) can cause delay degradations on the speed paths in logic circuits by increasing the threshold voltage of stressed pMos devices. The NBTI-induced delay degradation is a slow, accumulative process, and may not be visible during offline testing of a chip. In CMOS devices, the NBTI-induced threshold voltage shift occurs over a period of months or years, depending on the operating conditions of the device. If the delay degradation due to NBTI-effects are not properly accounted for during design, it may affect the lifetime reliability of a chip.

Thus, lifetime reliability and robustness, alongside performance and power, emerges as a key factor that drives synthesis and design, and there is significant interest in the development of cost-effective techniques for robust error-resilient design of logic circuits.

The negative impact of variations on performance, power and reliability have caused a significant slow-down in the aggressive scaling of CMOS and have also led to the search for new nanoelectronic devices e.g. carbon nanotube transistors, single electron transistors etc. to possibly replace traditional CMOS in the future. These are fast emerging as the next generation computing devices that promise device densities orders of magnitude greater than current MOS device density. However, the state-of-the-art fabrication process for these nanoelectronic devices is not scalable and cannot guarantee reliable devices. Moreover, the failure rates observed in the emerging devices is much higher than the failure rate predicted for future CMOS devices. Thus, circuit reliability is an issue in future CMOS technologies

and in emerging nanoelectronic technologies.

Many techniques have been developed over the years to overcome these issues. These can be traced back to as early as 1956. The various techniques for circuit reliability can be classified into two groups on the basis of the failure rates targeted by them.

1. *Concurrent error detection/masking techniques*: Traditionally, concurrent error detection/masking techniques have targeted low failure rate model, i.e. a single fault model. The focus has been at minimizing area, performance and power overhead while achieving high reliability. Thus, concurrent error detection/masking techniques are best suited for application in future CMOS technologies. For instance, a single gate failure model accurately captures the effects the effect of single-event upsets in logic circuits.

   Error detection techniques provide the capability of signaling when an error has reached the output. Error detection techniques have been used for a long time to protect data in memories and caches. Here, error detection codes like parity, SEC/DED are padded to the normal data bits for protection. The application of coding techniques to concurrent error detection (CED) in logic circuits have been investigated. Efficient implementation of CED circuitry using parity-check, Berger codes, Bose-Lin codes is possible for simple logic functions and regular multi-level logic structures like adders, multipliers, shift registers [24]. Duplication based techniques [21] have also been explored. These techniques assume a single fault model, and aim at minimizing the overhead for error detection [21,29].

   Error masking techniques suppress errors before the state of the circuit is corrupted. Techniques for error masking include duplication-based techniques e.g. [22], post-technology mapping techniques e.g. [17] that uses implications within a circuit to mask $0 \rightarrow 1$ or $1 \rightarrow 0$ errors.

2. *Fault-tolerant architectures*: Fault-tolerant architectures target a high failure rate model i.e. a fault model in which there is non-zero probability of multiple gate

failures. However, fault-tolerant architectures incur high overheads because the multiple gate failures are countered with massive redundant computation. The challenge lies in using the redundancy most effectively to achieve high reliability. Thus, fault-tolerant architectures are best suited for emerging technologies that offer high device densities. The most widely used fault-tolerant architectures were proposed by von Neumann in 1956 [30].

- Majority voting: This approach involves performing multiple computations of the same function and then returning the majority value as the correct output. The majority value is computed using a majority gate with odd number of inputs (to avoid ties). The majority gate acts like an amplifying unit, that amplifies the probability of correct values at the output, given a certain probability of correctness at the inputs. This is called the restoring/amplifying property. Thus, if each set of computation has a certain probability of being incorrect, the output of the majority gate has a smaller probability of being incorrect. Instead of a majority gate, other types of gates for e.g. NAND gates can be used for computation and restoration. The analysis of such an approach has been done by Pippenger and Evans [9], and later using bifurcation and simulation techniques by Fortes [11,25] (for any k-input NAND gates).

- Multiplexing technique: Like majority voting, multiplexing also uses redundant computation to counter the effects of gate errors. In the multiplexing technique, each wire in the circuit is replaced by a bundle of wires (say of size B), and every gate is replaced by B gates for redundant computation. Thus, for each output there is a bundle of B wires instead of a single wire. A fraction, called the threshold ($\triangle$), is defined for designating the output as a zero or a one. For instance, for a bundle size of 10, and the threshold is 0.3, if less than 3 wires are high it means that the output is zero and if greater than 7 wires are high it means means the output is one. If $3 - 7$ wires are high, an error is flagged. The main difference between majority voting and multiplexing is that multiplex-

ing combines the results of redundant computation at the output of the circuit, while majority voting restores the correct value using a majority gate after every redundant stage.

This thesis adds to our understanding of circuit reliability on two fronts:

- *Reliability analysis*: The first step towards understanding the effect of failures in a logic circuit is to be able to compute the reliability of a circuit. Reliability analysis of logic circuits is $\mathcal{NP}$-hard because of the exponential number of inputs, combinations and correlations in gate failures, and their propagation and interaction at multiple primary outputs. By coupling probability theory with concepts from testing and logic synthesis, accurate and scalable algorithms for reliability analysis of logic circuits are presented in Chap. 2.

- *Error detection based on approximate logic functions*: A scalable, technology-independent algorithm for the synthesis of approximate logic circuits is described in Chap. 3. The reliability analysis framework is used to guide the approximate logic synthesis to provide a low overhead, non-intrusive solution for concurrent error detection (CED). The proposed synthesis algorithm for approximate logic circuits provides fine-grained trade-offs between area-power overhead and CED coverage. CED based on approximate logic circuits incurs no performance penalty and also provides extensive coverage to delay faults. This can be leveraged to detect errors caused by delay faults that arise from failure mechanisms such as transistor wearout on speed-paths in the original design.

Chapter 2

Reliability analysis

Scaling of CMOS devices into the sub-100nm region has significantly increased their vulnerability to various sources of noise like crosstalk, terrestrial cosmic radiation, electromagnetic interference and other nanometer effects that cause the output voltage of a gate to deviate from its nominal value with a certain probability. In other words, the output voltage of the gate can be characterized by a probability distribution around the nominal value, rather than by a single number. A Gaussian distribution with the nominal value as the mean and a variance σ (Fig. 2.1) is a good approximation of this distribution [14]. A logic circuit performs computation in the discrete domain (involving boolean logic) and the noise at the gates, modeled as Gaussian distribution, is in the continuous domain. Thus, reliability analysis under this model would involve Boolean operations (AND/OR) on Gaussian distributions to evaluate the effect of noise at the primary outputs. This incompatibility can

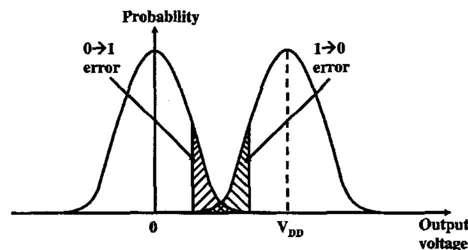


Figure 2.1: Effect of noise on the output voltage of a gate.

be resolved by discretizing the model for noise as follows. This Gaussian distribution for the output voltage can be translated into a $0 \rightarrow 1$ and $1 \rightarrow 0$ error probability at the output of the gate for e.g. the $0 \rightarrow 1$ probability of error is the probability that the output voltage exceeds the noise margin when the nominal value is 0. The $1 \rightarrow 0$ probability of error can be defined in a similar manner. For the sake of clarity the $0 \rightarrow 1$ error probability has been assumed to be equal to the $1 \rightarrow 0$ error probability, and is referred to as the gate failure probability (ϵ). In general, different values of the two error probabilities ($0 \rightarrow 1$ and $1 \rightarrow 0$) can be handled by all the reliability analysis algorithms proposed in this thesis.

The discrete model for noise at the gates behaves like a binary symmetric channel (BSC) at the output of the gate. In other words, following the computation at the gate, the BSC can cause the gate output to flip symmetrically (from $0 \rightarrow 1$ or $1 \rightarrow 0$) with the same probability of error, ϵ. Each gate has an $\epsilon \in [0, 0.5]$ associated with it, where ϵ equals 0 for an error-free gate and ϵ equals 0.5 for perfectly noisy gate (a gate with random output). When $\epsilon > 0.5$, it is equivalent to a gate with $\epsilon < 0.5$, but with the complement Boolean function. Note that gates are assumed to fail independently of each other. Although this may not be a realistic assumption, where noise due to process variations or external disturbances is highly localized and correlated, it helps to simplify reliability analysis while still providing valuable insights into circuit reliability.

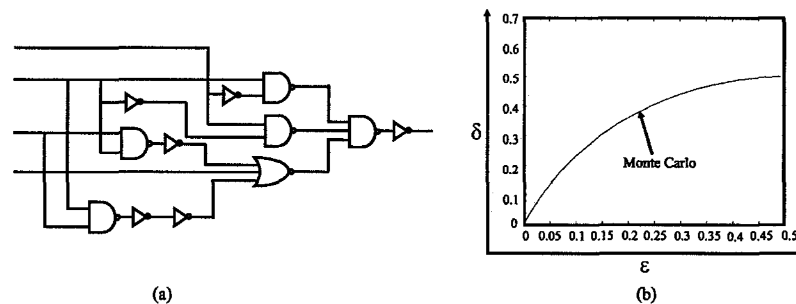


Figure 2.2: Reliability of a logic circuit illustrated with the help of an example.

Reliability of a logic circuit is defined as the probability of error at the output of the logic circuit, δ, as a function of failure probabilities $\vec{\epsilon}$ of the gates, where $\vec{\epsilon}$ is the vector

containing the failure probabilities $\{\epsilon_1, \epsilon_2, ...\}$ of the gates in the circuit. Reliability $\delta(\vec{\epsilon})$ can lie in the interval from 0 to 1. The concept of circuit reliability is illustrated in Fig. 2.2. Each gate in the circuit shown in Fig. 2.2(a) has the same failure probability(ϵ). The plot shown in Fig. 2.2(b) shows the probability of error(δ) at the output of the circuit, when ϵ varies from 0 to 0.5.

Given that each gate in the circuit can fail independently with a probability ϵ_i, the probability of any subset, S, of gates failing simultaneously can be computed as $\prod_{i \in S} \epsilon_i \prod_{j \notin S}(1-\epsilon_j)$. For large circuits, there is a finite probability of a large number of gates failing simultaneously. The large number of simultaneous gate failures is an artifact of the independent gate failure model. Thus, the independent gate failure model can be used to model failures in emerging nanoelectronic devices, where failure rates are high. However, for CMOS technologies in the near future the independent gate failure model is pessimistic and can be improved by imposing a limit on the number of simultaneous gate failures. For instance, in the single gate failure model the limit is 1. In general, the independent gate failure model is extended by imposing a limit, k ($0 \leq k \leq N$), on the number of simultaneous gate failures. This model will be referred to as the maximum-k gate failure model. An efficient reliability analysis algorithm for this model is also proposed in this thesis.

Reliability analysis is $\mathcal{NP}$-hard because of the exponential number of inputs, combinations and correlations in gate failures, and their propagation and interactions at multiple primary outputs. It is $\mathcal{NP}$-hard even under the single fault (i.e., single noisy gate) assumption. The presence of reconvergent fanout introduces correlations among gate failures, and reliability analysis quickly becomes intractable. The traditional approach to reliability analysis uses fault injection and simulation in a Monte Carlo framework. Recent progress in reliability analysis has seen the use of probabilistic transfer matrices (PTMs) [16] and Bayesian networks [26]. Without exception, these approaches suffer from the problem of scalability. Monte Carlo simulations have the added disadvantage of inflexibility, since the entire simulation has to be repeated for any change in circuit structure or $\vec{\epsilon}$. PTM-based reliability analysis uses transfer matrices to represent input-output behavior of noisy cir-

cuits. PTMs store the probability of occurrence of every input-output vector pair for each level in the circuit to compute the probability of error at the output of the circuit. This leads to massive matrix storage and manipulation overhead. Even with compaction of the matrices using ADDs, the high runtimes for benchmark circuits with 20–50 gates suggest their inapplicability to large circuits. Although this problem is somewhat mitigated in the Bayesian network approach for small circuits, manipulating Bayesian networks for large circuits is potentially intractable. Alternatively, analytical approaches developed to study fault-tolerant approaches like nand multiplexing and majority voting can be used for reliability analysis [27,30]. However, the simple compositional rules that these approaches use work best on regular structures. When used on irregular multi-level structures such as logic circuits, they suffer significant penalties in accuracy even on small circuits.

This thesis presents two algorithms for reliability analysis under the independent gate failure model. The observability-based reliability analysis algorithm provides a closed-form expression for the probability of error at the primary outputs as a function of the failure probabilities and observabilities of the gates in the circuit. The observability-based reliability analysis algorithm provides good accuracy for small values of ϵ. However, for larger values of ϵ, the accuracy decreases because the effect of multiple gate failures is not captured accurately by the closed-form expression. These issues are addressed by the single-pass reliability analysis algorithm in which the effect of gate failures is propagated to the primary outputs in a topological pass through the circuit. The single-pass reliability analysis algorithm is used as the core engine for the reliability analysis under the maximum-k gate failure model.

2.1 Observability-based reliability analysis

In this section, an intuitive approach to reliability analysis is described. It is based upon the observation that a failure at a gate close to the primary output has a greater probability of propagating to the primary output than a gate several levels of logic away from the primary

outputs. This is because a failure that has to propagate through several levels of logic has a higher probability of being logically masked. This can be quantified by applying the concept of observability, which has historically found use in the testing and logic synthesis domains [20].

For reliability analysis, the observability of any wire in the circuit can be defined as the probability that a $0 \longrightarrow 1$ or $1 \longrightarrow 0$ error at that wire affects the output of the circuit. Note that the observabilities are the noiseless observabilities, i.e., all the gates are assumed noise-free when the observabilities are calculated. Observabilities can be calculated using Boolean differences, symbolic techniques based on binary decision diagrams (BDDs), or simulation. Using the observabilities, a closed-form expression for the reliability, $\delta_y(\vec{\epsilon})$, is derived.

Let us begin with the trivial case of a circuit with a single gate having a failure probability ϵ. Since there is only one gate, the output of the gate is the primary output of the circuit and has an observability o of 1. Hence, the probability of failure of the output is equal to $\epsilon \cdot o = \epsilon$. Now, consider the case of a circuit with two gates g_1 and g_2 having failure probabilities ϵ_1 and ϵ_2 and observabilities o_1 and o_2. When both gates are error-free, the primary output of the circuit is always error-free. When at least one gate is in error, the error at the primary output is computed by analyzing two cases when (i) only one gate is in error and ii) both g_1 and g_2 are in error. When exactly one gate has failed, the primary output is in error when the failed gate is observable. Hence, the first error component of the output is $\epsilon_1(1-\epsilon_2)o_1 + (1-\epsilon_1)\epsilon_2 o_2$.

When both g_1 and g_2 are in error, the primary output is in error when g_1 and g_2 are *jointly* observable. Joint observability of two gates g_1 and g_2 is the probability that the primary output toggles when the outputs of both g_1 and g_2 toggle. Note that the joint observability is different from the simultaneous observability of g_1 and g_2, which is the probability that a toggle at g_1 causes a toggle at the primary output and a toggle at g_2 also causes a toggle at the primary output. Computing joint observability for all combinations of multiple gate failures is expensive because joint observability computation is itself computationally

demanding and the number of combinations of multiple gate failures is exponential in the number of gates in the circuit. Thus, two simplifying assumptions are made for estimating the effect of multiple gate failures:

1. The effect of gate failures at the primary output are decoupled from each other, i.e., a failure at each gate g_i is assumed to affect the output with a probability o_i regardless of other gate failures. This assumption allows the joint observability to be replaced by simultaneous observability, which is computationally less expensive, in computing the effect of multiple gate failures at the output. Note that joint observability and simultaneous observability are not equal when the observability of a gate changes due to errors at other gates.

2. The observability of the gates are assumed to be independent of each other. Using this assumption, the computation of simultaneous observability of two gates can be simplified to the product of the individual gate observabilities. For instance, the probability that g_1 is observable and g_2 is not observable is $o_1(1 - o_2)$.

Let G be the set of all the gates that are in error. The exact effect of these failed gates is given by the joint observability of the gates in G. Using the first assumption, it can be argued that the output is in error whenever an odd number of gates in G are simultaneously observable. Using the same assumption, when an even number of gates in G are simultaneously observable, the effect of these gate failures cancel each other. Thus, the joint observability of gates in G is estimated as the sum of simultaneous observability of an odd number of gates in G. Using the second assumption, the simultaneous observability of the two gates is given by the product of their individual observabilities. Note that in practice, there are cases when an odd number of simultaneously observable gates in G do not cause an error at the primary output and also cases when an even number of simultaneously observable gates in G cause an error at the primary output. These are averaged and absorbed into the probability of an odd number of failed gates being simultaneously observable by the first assumption.

For two gate example described above, the second error component of the output (both gates in error) is given by $\epsilon_1\epsilon_2(o_1(1-o_2)+o_2(1-o_1))$. The first term $o_1(1-o_2)$ is the probability that g_1 is observable and g_2 is not and vice-versa for the second term. The joint observability of g_1 and g_2 is estimated by $o_1(1-o_2)+o_2(1-o_1)$ (odd number of erroneous gates being simultaneously observable). The inaccuracies introduced due to the two simplifying assumptions are discussed in greater detail in Sec. 2.1.2.

With this background, we shall derive the expression for the probability of error at the output for a general circuit with N gates. Without loss of generality, we assume that the circuit has a single output y. Denote the error probability (observability) of the ith gate by $\epsilon_i(o_i)$. Let Ω be the set of all the gates in the circuit and S be the set of all non-empty subsets of Ω. Consider a set $G \in S$ of gates that have failed. Using the first assumption, the output y will be in error when an *odd number* of gates in G are simultaneously observable. Using the second assumption, the simultaneous observability of two gates can be computed by simply multiplying the individual observabilities of the gates. For instance, the probability that gate 1 and gate 2 are observable but gate 3 is not observable is $o_1o_2(1-o_3)$. Thus, for 3 gates the probability of an odd number of gates being observable is given by $o_1(1-o_2)(1-o_3)+o_2(1-o_1)(1-o_3)+o_3(1-o_1)(1-o_2)+o_1o_2o_3$. Thus, in general, the probability of error (y_{error}) at the output y given that the gates in G have failed is given by the following expression:

$$\begin{aligned}
\Pr(y_{\text{error}}|G) &= \frac{1}{2}\left(\sum_{F\in 2^G}\prod_{i\notin F}(1-o_i)\prod_{j\in F}o_j - \sum_{F\in 2^G}\prod_{i\notin F}(1-o_i)\prod_{j\in F}-o_j\right) \\
&= 1/2\left(\prod_{j\in G}(o_j+(1-o_j)) - \prod_{j\in G}((1-o_j)-o_j)\right) \\
&= 1/2\left(1-\prod_{j\in G}(1-2o_j)\right) \qquad (2.1)
\end{aligned}$$

The probability that the gates in G are in error and the gates in G^c $(\Omega\backslash G)$ are error-free is

given by $\prod_{i \in G} \epsilon_i \prod_{j \in G^c}(1-\epsilon_j)$. Thus, the probability of error at the output y is given by

$$\Pr(y_{\text{error}}) = \sum_{G \in S} \prod_{i \in G} \epsilon_i \prod_{j \in G^c} (1-\epsilon_j) \left(\frac{1 - \prod_{j \in G}(1-2o_j)}{2} \right)$$

$$\Rightarrow \Pr(y_{\text{error}}) = 1/2 \sum_{G \in S} \prod_{i \in G} \epsilon_i \prod_{j \in G^c} (1-\epsilon_j)$$

$$- 1/2 \sum_{G \in S} \prod_{i \in G} \epsilon_i (1-2o_i) \prod_{j \in G^c} (1-\epsilon_j)$$

Since S contains all non-empty subsets of Ω, the first term $\sum_{G \in S} \prod_{i \in G} \epsilon_i \prod_{j \in G^c}(1-\epsilon_j)$ contains all the terms in $\prod_{i \in \Omega}((1-\epsilon)+\epsilon)$ (when it is expanded) except $\prod_{i \in \Omega}(1-\epsilon_i)$. Hence, the first term can be replaced by

$$\prod_{i \in \Omega}(1-\epsilon+\epsilon) - \prod_{i \in \Omega}(1-\epsilon_i) = 1 - \prod_{i \in \Omega}(1-\epsilon_i)$$

A similar transformation for the second term yields

$$\Pr(y_{\text{error}}) = 1/2 \left(1 - \prod_{i \in \Omega}(1-\epsilon_i) \right)$$

$$- 1/2 \left(\prod_{i \in \Omega} [1 - \epsilon_i + \epsilon_i(1-2o_i)] - \prod_{i \in \Omega}(1-\epsilon_i) \right)$$

$$\Rightarrow \Pr(y_{\text{error}}) = 1/2 \left(1 - \prod_{i \in \Omega}(1-2\epsilon_i o_i) \right) \tag{2.2}$$

Eqn. 2.2 is a closed-form expression for the reliability of the output of a circuit as a function of error probabilities at each gate. Since the product of $(1-2\epsilon_i o_i)$ is over all gates in the circuit, it can be computed very efficiently once the observability of each gate is known. Eqn. 2.2 is intuitive because the error probability of each gate ϵ_i is scaled by o_i. Hence, errors at gates close to the primary outputs (high o_i) are more likely to cause output errors than errors at gates that are several levels of logic deep (low o_i).

2.1.1 Example

The observability-based reliability analysis algorithm can be illustrated using Fig. 2.3. There are two values indicated for each node(gate). The first is the gate failure proba-

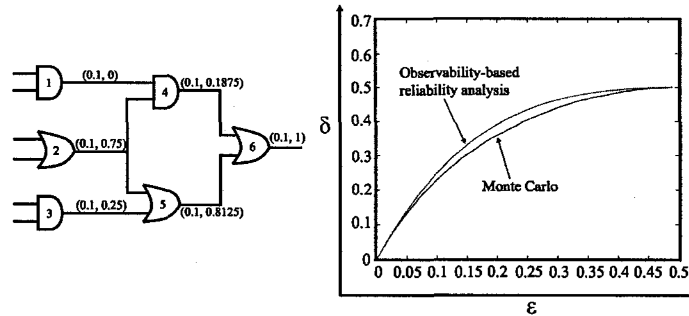


Figure 2.3: Example circuit to illustrate of observability-based closed-form expression

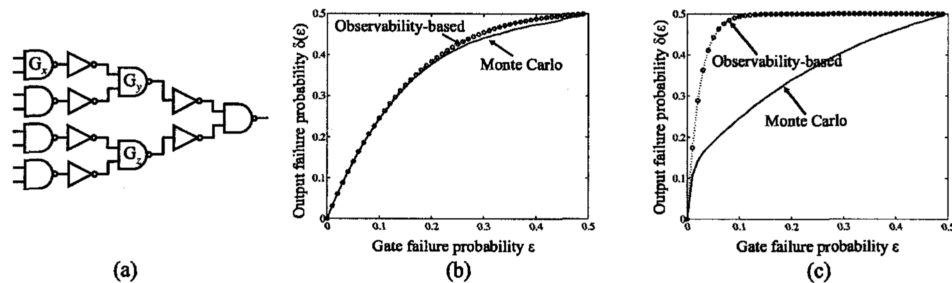


Figure 2.4: Circuit for illustrating the effect of noise and correlation on observability

bility and the second is the observability of the gate at the output. The probability of error at the output (δ) is computed as:

$$\delta = \frac{1 - \begin{array}{c}(1-2\times 0.1\times 0)(1-2\times 0.1\times 0.75)(1-2\times 0.1\times 0.25)\\(1-2\times 0.1\times 0.1875)(1-2\times 0.1\times 0.8125)(1-2\times 0.1\times 1)\end{array}}{2}$$

Fig. 2.3 compares the reliability curves obtained from observability-based reliability analysis and Monte Carlo simulations for a range of ϵ from 0 to 0.5 (the same value of ϵ is used for each gate).

2.1.2 Noise and correlation distort observability

Simulation results indicate that the closed-form expression for $\delta(\vec{\epsilon})$ is highly accurate for small circuits, and deviates by a small margin for ϵ close to 0.5. Note that the same value of

gate failure probability has been used for each gate, and hence $\vec{\epsilon}$ is replaced by ϵ. For example, the Monte Carlo and observability-based curves for $\delta(\vec{\epsilon})$ for the circuit in Fig. 2.4(a) are shown in Fig. 2.4(b). Simulation results also indicate that the closed-form expression performs well for small values of ϵ in large circuits, and that the accuracy depends on the number of gates in the circuit with $\epsilon > 0$. For example, Fig. 2.4(c) compares the $\delta(\vec{\epsilon})$ curves for a single output of the benchmark circuit b9, where a large error is observed as ϵ increases.

The observability-based reliability analysis is accurate for small ϵ because the probability of single gate failures is significantly higher than the probability of multiple gate failures. Since the effect of an error at a single gate is given by the gate failure probability scaled by its observability, it is exactly accounted for in the closed-form expression for reliability. As ϵ increases, the effect of multiple gate failures starts becoming significant and a deviation of the observability-based curve from the Monte Carlo curve is observed. There are two reasons for the inaccuracy of observability-based analysis in computing the effects of multiple gate failures. Both are related to the fact that the observability calculations are done statically:

i) On individual gates in the circuit: When observability computation is performed on gates one-at-a-time in the derivation of the closed-form expression for $\delta(\vec{\epsilon})$, the events of two or more gates being simultaneously observable is computed assuming that the events are independent. For instance, consider gates G_x and G_y in the circuit of Fig. 2.4(a). Assuming independence suggests that G_x is observable even when G_y is not because $o_x(1 - o_y) > 0$. However, since G_x is in the transitive fanin of G_y, it is clear that G_x is observable *only if* G_y is observable. Assuming independence thus introduces inaccuracies in the closed-form expression.

ii) In the absence of noise: When the observability calculations are performed in the absence of noise, it is assumed that a path remains sensitized irrespective of failures at gates that contribute to sensitizing that path. However, a failure at one or more of these gates may increase or decrease the observability of the original gate. For instance, consider

gates G_x and G_z in the circuit of Fig. 2.4(a). Exhaustive analysis indicates that if both G_x and G_z fail, the probability of an output failure is $46/256$. However, the closed-form expression ignores the effects of how failures at G_z influence the propagation of failures from G_x and estimates this probability to be $19/256$. This problem is further exacerbated by the effects of reconvergent fanout that is common in logic circuits, since observability calculation at the source of reconvergent fanouts becomes more complex and expensive.

In conclusion, the observability-based closed-form expression is highly suitable for reliability analysis of small circuits and for small values of gate failure probabilities in large circuits. The algorithm is simple, yet efficient and flexible because a change in the value of noise at any gate(s) just requires re-computation of the closed-form expression (2.2). Since gate failure rates in current CMOS technologies are of the order of 10^{-6}–10^{-2}, it can easily be applied to reliability analysis and design optimization to enhance reliability.

2.2 Single-pass reliability analysis

The efficient single-pass reliability analysis technique described here addresses the accuracy drawbacks of the observability-based algorithm. At the core of this algorithm is the observation that an error at the output of any gate is the cumulative effect of a local error component attributed to the ϵ of the gate, and a propagated error component attributed to the failure of gates in its transitive fanin cone. When the components are combined, the total error probability at gate g is given by (i) a $0 \rightarrow 1$ error probability given that its error-free value is 0, $\Pr(g_{0\rightarrow 1})$ and (ii) a $1 \rightarrow 0$ error probability given that its error-free value is 1, $\Pr(g_{1\rightarrow 0})$.

In general, $\Pr(g_{0\rightarrow 1}) \neq \Pr(g_{1\rightarrow 0})$ for an internal gate in a circuit. Initially, $\Pr(x_{i,0\rightarrow 1})$ and $\Pr(x_{i,1\rightarrow 0})$ are known for the primary inputs x_i of the circuit. In the core computational step of the algorithm, the $0 \rightarrow 1$ and $1 \rightarrow 0$ error components at the inputs to a gate are combined using a weight vector $\mathcal{W}$ to obtain a weighted input error vector $\mathcal{PW}$. The $\mathcal{PW}$ vector is then combined with the local gate failure probability ϵ to obtain $\Pr(g_{0\rightarrow 1})$ and

$\Pr(g_{1\to 0})$ at the output of the gate. Computation of the (i) weight vector and (ii) weighted input error vector is described below.

Single-pass reliability analysis is performed by applying the core computational step of the algorithm recursively to the gates in a topological order. At the end of the single-pass, $\Pr(y_{0\to 1})$ and $\Pr(y_{1\to 0})$ is obtained for the output y of the circuit. The reliability δ_y of an output y is then given by the weighted sum of $\Pr(y_{0\to 1})$ and $\Pr(y_{1\to 0})$ as follows:

$$\delta_y(\epsilon) = \Pr(y = 0)\Pr(y_{0\to 1}) + \Pr(y = 1)\Pr(y_{1\to 0})$$

Given the weight vectors at all gates, the time complexity of the algorithm is $O(N)$, where N is the number of gates in the circuit. Note that single-pass reliability analysis gives the exact values of probability of error at the output in the absence of reconvergent fanout.

i) Weight vector: The weight vector for a gate stores the probability of occurrence of every combination of inputs at that gate. For instance, the weight vector of a 2-input (3-input) gate consists of 4 (8) entries. Since the weight vector is just the joint signal probability distribution of the inputs of a gate, it can be computed by random pattern simulation or symbolic techniques based on BDDs. Weight vectors are independent of $\vec{\epsilon}$ and change only if the structure of the logic circuit changes. To improve the efficiency of the algorithm, weight vector computation may be performed once at the beginning and used over several runs of reliability analysis. The BDDs for the gates in the circuit are manipulated to compute the components $\mathcal{W}_{00}$, $\mathcal{W}_{01}$, etc. of $\mathcal{W}$. For example, if b_1 and b_2 are the inputs to a gate, $\mathcal{W}_{00}$ is given by the number of minterms in $\overline{b}_1\overline{b}_2$ divided by the total number of input vectors to the circuit.

ii) Expressions for weighted input error vector: Expressions for the components of $\mathcal{PW}$, for a 2-input AND gate with inputs i and j, are given in Table 2.1. The calculation of $\mathcal{PW}(0)$ to propagate the $0 \to 1$ error component using the entries in the upper part of Table 2.1 is described here. Propagation of the $1 \to 0$ input error component is similar, using the entries in the lower part of Table 2.1.

Since the probability of a $0 \to 1$ error is actually the probability of a $0 \to 1$ error given

Input vector	Weight	Weighted $0 \to 1$ input error component
00	$\mathcal{W}_{00}$	$\Pr(i_{0\to1})\Pr(j_{0\to1})\mathcal{W}_{00}$
01	$\mathcal{W}_{01}$	$\Pr(i_{0\to1})(1-\Pr(j_{1\to0}))\mathcal{W}_{01}$
10	$\mathcal{W}_{10}$	$(1-\Pr(i_{1\to0}))\Pr(j_{0\to1})\mathcal{W}_{10}$
Total	$\mathcal{W}(0)$	$\mathcal{PW}(0)$
Input vector	**Weight**	**Weighted $1 \to 0$ input error component**
11	$\mathcal{W}_{11}$	$\big(\Pr(i_{1\to0})+\Pr(j_{1\to0})-\Pr(i_{1\to0})\Pr(j_{1\to0})\big)\mathcal{W}_{11}$
Total	$\mathcal{W}(1)$	$\mathcal{PW}(1)$

Table 2.1: Expressions for weighted input error components

that the error-free output of the gate is 0, there are only 3 rows in the upper table, one for each input vector for which the output of the AND gate is 0. The first column in the table is the input vector under consideration. The input vector has been ordered as ij. The second column is the probability of occurrence of the input vector, i.e., the weight vector. The third column is the probability of a $0 \to 1$ error at g, caused only due to errors at its inputs (when g itself does not fail). The entries in the third column are computed using $\Pr(i_{0\to1})$, $\Pr(i_{1\to0})$, $\Pr(j_{0\to1})$, and $\Pr(j_{1\to0})$ as illustrated below with an example.

Consider the input 10, whose error-free output is 0. For g to be in error only due to errors at the inputs, j has to fail and i has to be error-free so that the input to the gate is 11 instead of 10. Thus, the probability of a $0 \to 1$ error at g due to this input vector is $(1 - \Pr(i_{1\to0}))\Pr(j_{0\to1})$. To compute the effect of the input vector 10, this probability of error is weighted by its probability of occurrence, i.e., by $\mathcal{W}_{10}$. Thus, the value in the third column for the vector 10 is $\mathcal{W}_{10}(1 - \Pr(i_{0\to1}))\Pr(j_{0\to1})$. Similar entries for the inputs 00 and 01 are derived, and summed to obtain an expression for the weighted input error probability $\mathcal{PW}(0)$.

Since we are calculating the weighted $0 \to 1$ input error probability at g *given* that the error-free output is 0, $\mathcal{PW}(0)$ has to be divided by $\mathcal{W}(0)$ to restrict the inputs to a set

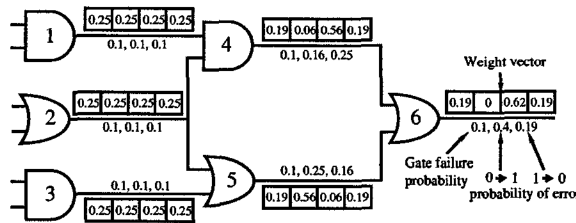


Figure 2.5: Illustration of single-pass reliability analysis

for which the error-free output is 0. Thus, the weighted $0 \rightarrow 1$ and $1 \rightarrow 0$ input error probability at g are given by

$$\Pr(g_{0\rightarrow 1}|g \text{ does not fail}) = \mathcal{PW}(0)/\mathcal{W}(0)$$

$$\Pr(g_{1\rightarrow 0}|g \text{ does not fail}) = \mathcal{PW}(1)/\mathcal{W}(1)$$

iii) Expressions for $\Pr(g_{0\rightarrow 1})$ **and** $\Pr(g_{1\rightarrow 0})$**:** If g fails with a probability of ϵ, $\Pr(g_{0\rightarrow 1})$ is given by

$$\Pr(g_{0\rightarrow 1}) = (1-\epsilon)\left(\frac{\mathcal{PW}(0)}{\mathcal{W}(0)}\right) + \epsilon\left(1 - \frac{\mathcal{PW}(0)}{\mathcal{W}(0)}\right)$$

Similarly, $\Pr(g_{1\rightarrow 0})$ is given by

$$\Pr(g_{1\rightarrow 0}) = (1-\epsilon)\left(\frac{\mathcal{PW}(1)}{\mathcal{W}(1)}\right) + \epsilon\left(1 - \frac{\mathcal{PW}(1)}{\mathcal{W}(1)}\right)$$

Note that the two terms $(1 - \Pr(i_{1\rightarrow 0}))$ and $\Pr(j_{0\rightarrow 1}))$ are multiplied in the computation of the entries in the third column of Table 2.1. This implies that the events of i being correct and j failing are assumed independent. This assumption is valid if the gate is not a site for reconvergence of fanout. Since reconvergence causes the two events to be correlated, it is handled separately in Sec. 2.2.1.

Although the computation has been illustrated for an AND gate, the computation for an OR gate is symmetric, i.e., there are 3 rows for the probability of $1 \rightarrow 0$ error table and a single row for the probability of $0 \rightarrow 1$ error table. Inverters, nands, nors, and xors are all handled in a similar manner and the tables have been excluded for brevity.

Single-pass reliability analysis is illustrated for the circuit shown in Fig. 2.5. The weight vector, gate failure probability (ϵ), and probability of $0 \rightarrow 1$ and $1 \rightarrow 0$ error are indicated

for each gate. The gates are numbered in the order in which they are processed. Since all the gates in the circuit have only 2-inputs, the weight vector for each gate consists of 4 entries. All entries of the weight vector for gate 1 are 0.25 because the primary input vectors are equally likely. The fanout at gate 2 reconverges at gate 6 via gates 4 and 5. Thus, the event of $0 \rightarrow 1$ and $1 \rightarrow 0$ error at the outputs of gates 4 and 5 are correlated. However, independence is assumed and the probability of these events are used in the computation of $0 \rightarrow 1$ and $1 \rightarrow 0$ probability of error values for the output of gate 6.

2.2.1 Handling reconvergent fanout

The presence of reconvergent fanout renders the single-pass reliability analysis approximate because the events of $0 \rightarrow 1$ or $1 \rightarrow 0$ error for the inputs of a gate may not be independent at the point of reconvergence. Handling reconvergent fanout has been the subject of extensive research in signal probability computation. In this section, the theory of correlation factors used in signal probability computation [8], is extended to make single-pass reliability analysis more accurate in the presence of reconvergent fanout.

This approach relies on the propagation of the correlation factors for a pair of wires from the source of fanout to the point of reconvergence. Note that the word "wire" has been used as opposed to "node" because for a gate with fanout > 1, each fanout is treated as a separate wire, but they constitute the same node. The correlation factor for events on a pair of wires is defined as the joint probability of the events divided by the product of their marginal probabilities. For signal probability computation, an event on a wire is defined as the value of the wire being 1. Thus, for a pair of wires, a single correlation factor is sufficient to compute the joint probability of a 1 on both the wires.

In our analysis, an event is a defined as a $0 \rightarrow 1$ or $1 \rightarrow 0$ error on a wire. Hence, instead of a single correlation factor, 4 correlation factors for a pair of wires, one for every combination of events on the pair of wires. If v and w are two wires, the 4 correlation factors for this pair are denoted by C_{vw}, $C_{v\tilde{w}}$, $C_{\tilde{v}w}$, and $C_{\tilde{v}\tilde{w}}$, where v, w, $\tilde{v}$, and $\tilde{w}$ refers to the event of a $0 \rightarrow 1$, $0 \rightarrow 1$, $1 \rightarrow 0$, and $1 \rightarrow 0$ error at v and w respectively.

The correlation factors come into play at the the gates whose inputs are the site of reconvergence of fanout. At such gates, the events of $0 \rightarrow 1$ or $1 \rightarrow 0$ error at the inputs are not independent. Thus, the entries in the third column of Table 2.1 are weighted by the appropriate correlation factor, e.g., $\Pr(i_{0\rightarrow 1})(1 - \Pr(j_{1\rightarrow 0}))$ becomes $\Pr(i_{0\rightarrow 1})(1 - \Pr(j_{1\rightarrow 0})\mathcal{C}_{i\bar{j}})$.

Correlation factor computation: The correlation factor for a pair of wires can be calculated by first computing the correlation factors for the wires in the fanout source that cause the correlation, and then propagating these correlation factors along the appropriate paths leading to the pair of wires. Note that all four correlation factors for two independent wires are 1. The computation of correlation factors for the fanout source and the propagation of correlation factors at a 2-input AND gate are described below.

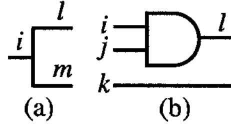


Figure 2.6: Computation/propagation of correlation factor

i) Computation at fanout source node: The fanout source node i is shown in Fig. 2.6(a). The correlation factor for the pair of wires $\{l, m\}$ is computed as follows:

$$\Pr(l_{0\rightarrow 1}) = \Pr(l_{0\rightarrow 1}, m_{0\rightarrow 1}) = \Pr(l_{0\rightarrow 1}) \Pr(m_{0\rightarrow 1})\mathcal{C}_{lm}$$

$$\text{i.e., } \mathcal{C}_{lm} = \frac{1}{\Pr(m_{0\rightarrow 1})}$$

$\mathcal{C}_{\bar{l}\bar{m}}$ can be computed in a similar manner. $\mathcal{C}_{\bar{l}m}$ and $\mathcal{C}_{l\bar{m}}$ are zero because it is not possible to have a $0 \rightarrow 1$ error on m and a $1 \rightarrow 0$ error on l, or vice-versa.

ii) Propagation at an AND gate: Propagation of correlation factors is illustrated for the AND gate in Fig. 2.6(b). Let i, j, k be three wires whose pairwise correlation factors are known. Computation of the correlation factors for the pair $\{l, k\}$ involves propagation of

$$\begin{aligned}
\Pr(l_{0\to1}|k_{0\to1}) = \epsilon + \frac{(1-2\epsilon)}{\mathcal{W}(0)} & (\mathcal{W}_{00}\Pr(i_{0\to1}|k_{0\to1})\Pr(j_{0\to1}|k_{0\to1})\mathcal{C}_{ij} \\
& + \mathcal{W}_{01}\Pr(i_{0\to1}|k_{0\to1})(1-\Pr(j_{1\to0}|k_{0\to1})\mathcal{C}_{i\bar{j}}) \\
& + \mathcal{W}_{10}(1-\Pr(i_{1\to0}|k_{0\to1})\mathcal{C}_{\bar{i}j})\Pr(j_{0\to1}|k_{0\to1})) \\
= \epsilon + \frac{(1-2\epsilon)}{\mathcal{W}(0)} & (\mathcal{W}_{00}\Pr(i_{0\to1})\mathcal{C}_{ik}\Pr(j_{0\to1})\mathcal{C}_{jk}\mathcal{C}_{ij} \\
& + \mathcal{W}_{01}\Pr(i_{0\to1})\mathcal{C}_{ik}(1-\Pr(j_{1\to0})\mathcal{C}_{\bar{j}k}\mathcal{C}_{i\bar{j}}) \\
& + \mathcal{W}_{10}(1-\Pr(i_{1\to0})\mathcal{C}_{\bar{i}k}\mathcal{C}_{\bar{i}j})\Pr(j_{1\to0})\mathcal{C}_{jk})
\end{aligned}$$

Figure 2.7: Derivation of $\Pr(l_{0\to1}|k_{0\to1})$ in terms of correlation factors of its inputs.

the correlation factors through the AND gate, using the correlation factors of i, j with k.

$$\mathcal{C}_{lk} = \frac{\Pr(l_{0\to1}|k_{0\to1})}{\Pr(l_{0\to1})}$$

The expression for $\Pr(l_{0\to1}|k_{0\to1})$ in terms of the correlation factors of the inputs i, j with k is shown in Fig. 2.7. The terms in the expression for $\Pr(l_{0\to1}|k_{0\to1})$ are similar to the terms in the third column of the upper part of Table 2.1. The only difference is that the probability of $0 \to 1$ and $1 \to 0$ errors have been multiplied by appropriate correlation factors. Note that the terms of the weight vector $\mathcal{W}$ include the signal probability of k. The expression for $\mathcal{C}_{\bar{l}k}$ is derived in a similar manner using the lower part of Table 2.1, and is left out for brevity. Expressions for $\mathcal{C}_{l\bar{k}}$ and $\mathcal{C}_{\bar{l}\bar{k}}$ are derived by replacing k by $\tilde{k}$ in the expressions for $\mathcal{C}_{lk}$ and $\mathcal{C}_{\bar{l}k}$ respectively. In Fig. 2.8, the consolidated probability of error at two correlated primary outputs of benchmark circuit b9 is used to illustrate the accuracy achieved with correlation factors.

2.2.2 Accuracy vs. computation complexity

As shown in Fig. 2.8 the degree of accuracy obtained using 4 correlation factors for a pair of wires is quite high. In the example shown in fig. 2.6 the correlation factors used in the

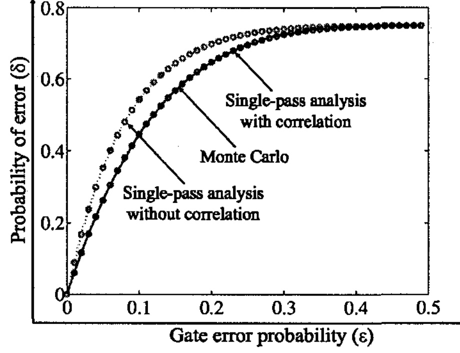


Figure 2.8: Handling reconvergent fanout in single-pass reliability analysis with 4 correlation factors.

simplest conditional probability terms like $\Pr(l_{0\to1}|k_{0\to1})C_{lk}$ are termed as first order correlation factors. Second order correlation factors are of the form $\Pr(i_{0\to1}|j_{0\to1}, k_{0\to1})C_{ijk}$ and so on for higher orders. correlation factors of order greater than 1 are approximated using the product of first order correlation factors, i.e. $C_{ijk} = C_{ij}C_{ik}$. When only 4 correlation factors are used there are some first order correlation factors that are not evaluated exactly, for example

$$\Pr(l_{0\to0}, k_{0\to1}) + \Pr(l_{0\to1}, k_{0\to1}) = \Pr(k_{0\to1}|l = 0) \tag{2.3}$$

Since $\Pr(l_{0\to1})$ is actually $\Pr(l_{0\to1}|l = 0)$. Using correlation factors the above expression can be rewritten as

$$\Pr(l_{0\to0})C + \Pr(l_{0\to1})C_{lk} = \frac{\Pr(k_{0\to1}|l = 0)}{\Pr(k_{0\to1})}$$

$$\Pr(l_{0\to0})C = \frac{\Pr(k_{0\to1}|l = 0)}{\Pr(k_{0\to1})} - \Pr(l_{0\to1})C_{lk}$$

Note that C is used to denote the correlation factor for $l_{0\to0}$ and $k_{0\to1}$ (which is different from the correlation factor for C_{lk} and $C_{\bar{l}k}$). When only 4 correlation factors are used C is approximated to $1 - \Pr(l_{0\to1})C_{l,k}$ because the value of $\Pr(k_{0\to1}|l = 0)$ is not known, and cannot be computed easily. This problem can be solved by using 16 correlation factors for a pair of wires. These 16 factors arise from the correlation among all combinations of

$\{0 \rightarrow 0, 0 \rightarrow 1, 1 \rightarrow 0, 1 \rightarrow 1\}$ failures for the two wires. The computation of these 16 correlation factors is done in the exact same manner as described for 4 correlation factors. Thus, using 16 correlation factors, higher accuracy can be obtained at the cost of higher computational complexity in propagating the 16 correlation factors.

2.3 Maximum-k gate failure reliability analysis

Recall the notation introduced in Sec. 2.1, $\Omega = \{0, 1, \cdots, N-1\}$ is the set of all gates in a circuit with N gates. The failure probability of the ith gate is denoted as ϵ_i. Both algorithms for reliability analysis proposed above - observability-based and single-pass, assume an independent gate failure model. Under an independent gate failure model, the probability that a subset $(F, F \subset \Omega)$ of gates fail simultaneously is given by the expression

$$\Pr(F) = \prod_{i \in F} \epsilon_i \prod_{i \notin F} (1 - \epsilon_i)$$

Thus, there is a non-zero probability that a large number of gates in the circuit fail simultaneously. For example, in a circuit with 10000 gates, if each gate has a failure probability of $10^{-}3$, then the probability of 10 gates failing simultaneously is 0.125. It is not unreasonable to expect such high failure rates for the emerging technologies like carbon nanotube transistors, single electron transistors, etc at least with the current fabrication technology for these devices. However, these failure rates are much beyond what has been predicted for future semiconductor technologies. A more plausible gate failure model for semiconductor technologies would be to have an upper bound on the number of simultaneous gate failures. Under this model the failures at gates are still independent of each other. However, an additional constraint, that only a maximum of k gates can fail simultaneously, is introduced. This global constraint causes the gate failures to be correlated with each other. The correlation between the gate failures can be explained mathematically as follows: Let X_i be a random variable that takes a value 1 when the ith gate has fails and a value 0 otherwise. In the maximum-k gate failure model, the X_is are independent of

each other, but the additional constraint is that $\sum X_i \leq k$. This constraint means that $\Pr(X_{k+1} = 1 | X_1 = 1, X_2 = 1, ..., X_k = 1) = 0$. But $\Pr(X_{k+1} = 1) = \epsilon_{k+1}$. Since the conditional probability is not equal to the marginal probability, the gate failures are correlated with each other. Note that for a single fault model k is equal to 1.

In this section, a reliability analysis algorithm that combines Monte Carlo sampling techniques with single-pass reliability analysis for reliability analysis under a maximum-k gate failure model is proposed. A variety of Monte Carlo sampling techniques exist in literature. For a comprehensive study of Monte Carlo sampling methods, the reader is referred to [10]. These techniques address the problem of sampling m (with given weights) out of a sample space of n. Under a maximum-k gate failure model, the sample space size is $\binom{N}{0} + \binom{N}{1} + ... + \binom{N}{k}$, ($N$ is the number of gates in the circuit). N is of the order of a 10^4, and thus the sample space becomes intractable even for $k = 3$. However, a special property of the sample space — the independence of gate failures — is exploited in the proposed sampling algorithm to reduce the computational complexity to $O(Nk^2)$.

2.3.1 Why fault injection and simulation fails ?

The simplest algorithm for reliability analysis under maximum-k gate failure model is based on Monte Carlo simulations. It is based on fault injection and random patter simulation. The idea is to generate a random number uniformly distributed in the interval [0,1] for each gate. By comparing the generated random number with the failure probability of the corresponding gate, it can be ascertained whether the gate has failed or not. Thus, a sample of failed gates which is the set of all the failed gates is generated. Since, the random numbers for each gate has been generated independently, there is no control over the number of gates that fail. A sample of failed gates generated by this algorithm has to be discarded if it does not obey the constraint on the maximum number of gate failures. If the constraint is satisfied, random pattern simulation is performed to evaluate the effect of the failed gates on the outputs. This procedure is repeated for a large number of samples and their effect on the outputs is averaged. A major drawback of this algorithm is that for high values gate

failure probabilities and low values of k, a large number of generated samples may not obey the constraint on maximum number of gate failures and have to be discarded. For instance, suppose that the total number of gates in the circuit is 10000, each gate fails independently with a probability 0.05, and $k = 3$. Since fault injection is performed independently at each gate, the average number of erroneous gates is $10000 \times 0.05 = 500 \gg 3$. Hence, a large number of samples will be discarded because they do not satisfy the constraint on the maximum number of gate failures. Thus, a very large number of runs are required to achieve convergence of the solution, and this makes the algorithm computationally intensive. In contrast, the algorithm proposed in this thesis is based on generating only correct samples of gate failures (which satisfy the constraint on the maximum number of gate failures). After generating a sample of erroneous gates, single-pass reliability analysis algorithm is used to evaluate the probability of error at the output of the circuit due to these gate failures. The probability of failure at each primary output is obtained by averaging over a large number of samples. Since, only correct samples are generated the solution converges much faster than the Monte Carlo approach.

2.3.2 Probability distribution of gate failures

When a cap on the maximum number of simultaneous gate failures (k) is imposed, the sample space of the possible combinations of simultaneous gate failures is reduced to sets G of cardinality $\leq k$. Thus, for the maximum-k gate failure model the probability that a subset of gates, G, fail simultaneously is given by the following expression

$$\begin{aligned} S &= \sum_{\{G \subset \Omega \mid |G| \leq k\}} \prod_{i \in G} \epsilon_i \prod_{i \notin G} (1 - \epsilon_i) \\ \Pr(G) &= \frac{\prod_{i \in G} \epsilon_i \prod_{i \notin G} (1 - \epsilon_i)}{S}, \quad \text{if } |G| \leq k \\ \Pr(G) &= 0, \quad \text{if } |G| > k \end{aligned} \tag{2.4}$$

Sample	Probability	In terms of $\alpha_i s$
{}	$(1-\epsilon_1)(1-\epsilon_2)(1-\epsilon_3)(1-\epsilon_4)/S$	$1/S_\alpha$
{1}	$\epsilon_1(1-\epsilon_2)(1-\epsilon_3)(1-\epsilon_4)/S$	α_1/S_α
{2}	$\epsilon_2(1-\epsilon_1)(1-\epsilon_3)(1-\epsilon_4)/S$	α_2/S_α
{3}	$\epsilon_3(1-\epsilon_1)(1-\epsilon_2)(1-\epsilon_4)/S$	α_3/S_α
{4}	$\epsilon_4(1-\epsilon_1)(1-\epsilon_2)(1-\epsilon_3)/S$	α_4/S_α
{1,2}	$\epsilon_1\epsilon_2(1-\epsilon_3)(1-\epsilon_4)/S$	$\alpha_1\alpha_2/S_\alpha$
{1,3}	$\epsilon_1\epsilon_3(1-\epsilon_2)(1-\epsilon_4)/S$	$\alpha_1\alpha_3/S_\alpha$
{1,4}	$\epsilon_1\epsilon_4(1-\epsilon_2)(1-\epsilon_3)/S$	$\alpha_1\alpha_4/S_\alpha$
{2,3}	$\epsilon_2\epsilon_3(1-\epsilon_1)(1-\epsilon_4)/S$	$\alpha_2\alpha_3/S_\alpha$
{2,4}	$\epsilon_1\epsilon_3(1-\epsilon_2)(1-\epsilon_4)/S$	$\alpha_2\alpha_4/S_\alpha$
{3,4}	$\epsilon_3\epsilon_4(1-\epsilon_1)(1-\epsilon_2)/S$	$\alpha_3\alpha_4/S_\alpha$

Table 2.2: Sample space of possible gate failures

Let $\alpha_i = \epsilon_i/(1-\epsilon_i)$. The equations in Eqn. 2.4 can be rewritten more succinctly in terms of α_is as

$$\begin{aligned} S_\alpha &= \sum_{\{G\subset\Omega \mid |G|\leq k\}} \prod_{i\in G} \alpha_i \\ \Pr(G) &= \frac{\prod_{i\in G}\alpha_i}{S}, \quad \text{if } |G| \leq k \\ \Pr(G) &= 0, \quad \text{if } |G| > k \end{aligned} \tag{2.5}$$

The sample space and associated probability of failure for $N = 4$ and $k = 2$ is shown in Table 2.2.

2.3.3 Generating a sample of erroneous gates

In this section, we describe two straight-forward techniques for generating samples of erroneous gates according to the the probability distribution given in Eqn. 2.4. It turns out that the first solution is incorrect because the generated samples have a slightly different probability distribution, and the second solution is computationally intractable due to its exponential complexity. This motivates the third solution, which is efficient and generates the samples from the correct probability distribution.

The first solution for generating a sample of $\leq k$ gates is based on generating the gates one at a time in k steps. In each step, the interval [0,1] is divided into sub-intervals proportional to $\{1, \alpha_1, \alpha_2, ...\}$. Each sub-interval corresponds to a gate in Ω. The sub-interval proportional to 1 corresponds to "no gate failure". A uniformly distributed random number is generated in the interval [0,1] and the sub-interval in which the generated random number lies indicates the erroneous gate. Note that if the random number lies in the sub-interval proportional to 1 then no gate is erroneous in that step. If the random number lies in the interval α_i, then the ith gate is designated as failed and it is removed from the Ω. The reduced Ω is then used for the next step. This procedure is repeated for k steps. For instance, suppose $k = 3$. If the 3 steps generate $\{1, 1, 1\}$, then no gate has failed (denoted as $\{\}$ in Table 2.2). If $\{1, \alpha_2, \alpha_4\}$ is generated, then gate 2 and gate 4 have failed (denoted as $\{2, 4\}$ in Table 2.2). Although this approach seems correct at first, a closer look reveals that the distribution from which the sample of failed gates is generated by this approach is incorrect. The reason is that the samples of gate failures shown in Table 2.2 are unordered sets, i.e. the sample $\{1, \alpha_2, \alpha_4\}$ is same as $\{\alpha_2, 1, \alpha_4\}$. However, the sample $\{\alpha_1, \alpha_2, \alpha_4\}$ is generated 6 times in this algorithm (differing only in the order in which the gates are generated in different steps, $\{\alpha_1, \alpha_2. \alpha_4\}$, $\{\alpha_1, \alpha_4, \alpha_2\}$, ...). However, the sample $\{1, 1, 1\}$ is generated only once, and the sample $\{1, 1, \alpha_2\}$ is generated only three times.

Another straight-forward but inefficient approach is to divide the interval [0,1] into sub-intervals of widths equal to the probabilities given in Eqn. 2.5. Thus, every subset of possible gate failures has a sub-interval in [0,1] with width equal to its probability of occurrence (given by Eqn. 2.5). Then, a uniform random number in [0,1] is generated and the sub-interval in which it lies indicates the generated sample of erroneous gates. Since, the number of subsets of erroneous gates is $\binom{N}{0} + \binom{N}{1} + ... + \binom{N}{k}$, the runtime complexity of this algorithm is $O(N^k)$. For large values of N, this algorithm becomes unusable even for small values of k.

The third algorithm for generating a sample of erroneous gates has a runtime complexity $O(Nk^2)$. The pseudo-code for the algorithm is shown in fig. 2.9. Table 2.3 is used

```
G – Set of failed gates
Ω – Set of all gates
k – Maximum number of simultaneous gate failures

G = {}
r = Uniform random number in [0,1]

BuildTable(Ω, k, r) {
  Construct Table 2.3.
  NextGateFailure(Ω, k, r)
}

NextGateFailure(Ω, k, r) {
  if (k = 0) return
  else {
   i = Sampled column in Table 2.3 based on r.
   if (i ∉ "Erroneous gate")     //0 ≤ i ≤ k − 1
    k = i
    Recompute r.
    NextGateFailure(Ω, k, r)
   else          //0 ≤ i ≤ N − 1
    G = G ∪ i.
    Ω = Ω \ i.
    Recompute r.
    BuildTable(Ω, k − 1, r)
   end
  end
}
```

Figure 2.9: Proposed sampling algorithm

for generating the sample of erroneous gates in the proposed algorithm. The columns of Table 2.3 are divided into two parts (i) Erroneous gate - which has N columns, $\{0, 1, \cdots, N-1\}$, and (ii) New sample size - which has k columns, $\{0, 1, \cdots,$ k-1$\}$. The probability distribution of the samples shown in Table 2.2 can be rewritten by grouping together samples that share a gate. This grouping can be done systematically as shown in the first part of Table 2.3 under the column "Erroneous gate". The jth column in the table is the sum of probability of all samples that contain gate j. The ith row is the probability distribution when the maximum number of gate failures is i. Thus, in the ith the probability of every sample of erroneous gates occurs i times, once each under the column corresponding to every gate in the erroneous gate sample.

The BuildTable() routine in the pseudo-code 2.9 constructs Table 2.3 for a given Ω and k. The core routine that generates the sample of erroneous gates is the NextGateFailure() routine. The possible outcomes of an execution of the NextGateFailure() routine are the following: (i) if k = 0, the algorithm ends, (ii) No new erroneous gate is generated and the sample size is reduced by at least 1, (iii) A new erroneous gate is generated and the sample size is reduced by 1. The random number r is used to decide between outcome (ii) and (iii) as follows. The entries in the kth row of Table 2.3 are used to divide the interval $[0, T_k]$ (T_k is the row total of the kth row) into sub-intervals. Let $\mathcal{I}$ be the sub-interval that contains the random number rT_k.

If $\mathcal{I}$ corresponds to an entry in the "Erroneous gate" part of Table 2.3, then the outcome is (iii). In this case, a new erroneous gate g has been generated. The column number corresponding to sub-interval $\mathcal{I}$ is used to determine the erroneous gate g. The gate g is added to the sample set ζ and removed from Ω. The value of k is reduced by 1 since one erroneous gate has been found. The fraction of overlap that rT_k had with the sub-interval I is used as the new value of r. The function BuildTable() is recursively called so that a new table (Table 2.3) can be generated for the reduced Ω.

If $\mathcal{I}$ corresponds to an entry in the "New sample size" part of Table 2.3, then the outcome is (ii). In this case, no new erroneous gate has been generated, but the sample size has

Table 2.3: Proposed sampling algorithm table.

	Erroneous gate				New sample size					
	1	2	...	N	$k-1$	$k-2$	...	1	0	Row total
1	α_1	α_2	...	α_N	0	0	...	0	1	T_1
2	$\alpha_1(T_1 - G_{1,1})$	$\alpha_2(T_1 - G_{1,2})$	...	$\alpha_N(T_1 - G_{1,N})$	T_1	0	...	0	1	T_2
3	$\alpha_1(\frac{T_2}{2} - G_{2,1})$	$\alpha_2(\frac{T_2}{2} - G_{2,2})$	...	$\alpha_N(\frac{T_2}{2} - G_{2,N})$	$\frac{T_2}{2}$	T_1	...	0	1	T_3
...	...	...	...	...	...	...	...	...	...	...
k	$\alpha_1(\frac{T_{k-1}}{k-1} - G_{k-1,1})$	$\alpha_2(\frac{T_{k-1}}{k-1} - G_{k-1,2})$	...	$\alpha_N(\frac{T_{k-1}}{k-1} - G_{k-1,N})$	$\frac{T_{k-1}}{k-1}$	$\frac{T_{k-2}}{k-2}$	...	T_1	1	T_k

been reduced by at least 1. This means that the size of the final sample will be less than k. The column number corresponding to the sub-interval $\mathcal{I}$ is used to determine the new value of k, k_{new}. The random number r is also recomputed as described earlier. Since, Ω has not changed, Table 2.3 can be reused for the row k_{new}. Hence, in this case NextGateFailure() is directly called instead of BuildTable().

2.3.4 Effect of multiple gate failures

After the sample of gate failures has been generated, the effect of these gate failures on the output of the circuit is computed. It is a well-known fact (and also mentioned in this thesis) that the effect of failures varies for different gates (depending on the observability of the gate at the output). Also, multiple gate failures may mask each other at the output of the circuit. The probability of error at the output of the circuit has to be computed given that the gates in the generated sample deviate from their correct values. This problem can be solved efficiently using the single-pass reliability analysis described in Sec. 2.2 by setting the failure probability of the gates in the sample equal to 1 and the failure probability of rest of the gates equal to 0. Note that the the flexibility of the single-pass reliability analysis comes handy here, because for different samples of gate failures, it is only the failure probability of the gates in the circuit that is changing. Since there is no change in the circuit structure, the weight vector $\mathcal{W}$ has to be computed only once. This makes the algorithm for reliability analysis for the maximum-k gate failure model very efficient, and thus scalable to large circuits. Since the single-pass reliability analysis is the core reliability

analysis engine, the results of this algorithm are also very accurate.

2.4 Results

The simulations were run on a 2.4 GHz Opteron-based system with 4 GB of memory. A 64-bit parallel pattern simulator was used to implement a Monte Carlo framework for reliability analysis based upon fault injection for comparison with single-pass reliability analysis. The sample size used for Monte Carlo was 6.4 million random patterns. The runtimes for the reliability analysis algorithm under the maximum-k gate failure model are shown Table 2.6. A total of 50K samples were used to evaluate the reliability under this model. The benchmark circuits used in [5] were synthesized using gates with a maximum of 3 inputs. However, the benchmark circuits used in this thesis are synthesized using only two input gates. This was done to eliminate second order correlations occurring due to multiple reconvergent paths at a 3-input gates. Also, reordering of BDDs using CUDD_REORDER_SIFT algorithm was used to lower the runtimes from those reported in [5].

2.4.1 Observability-based reliability analysis

A comparison of Monte Carlo and observability-based reliability analysis is presented in Table 2.4. The accuracy of the observability-based reliability analysis is quite low for most of the benchmarks circuits. The reasons for this have been discussed in Sec. 2.1.2. The runtimes for small/medium sized circuits are encouraging. However, for the large benchmark circuits (e.g. c3540) the runtime is high. This is because exact observability computation becomes extremely complex with a large number of reconvergent fanouts, which is the case with c3540 benchmark.

Table 2.4: Table shows a comparison between the observability-based reliability analysis and Monte Carlo for 10 benchmark circuits. Five values of ϵ have been used for comparison.

Benchmark	Size	Average percentage error over all outputs						Runtimes	
		$\epsilon = 0.05$	$\epsilon = 0.1$	$\epsilon = 0.15$	$\epsilon = 0.2$	$\epsilon = 0.25$	$\epsilon = 0.3$	Monte Carlo	Observability-based
b9	210	30.14	30.19	26.22	21.16	16.23	11.75	37m 15s	0.034s
cu	59	16.35	15.61	14.27	13.03	11.23	9.14	9m 50s	0.019s
x2	56	21.88	18.98	15.6	12.15	9.03	6.59	8m 41s	0.019s
frg2	1024	21.36	24.82	25.29	23.29	19.86	15.81	286m 38s	0.657s
c499	650	38.29	26.69	17.89	11.70	7.38	4.38	134m 55s	1m 12.8s
c1355	653	37.96	26.5	17.59	11.26	6.90	4.08	135m 7s	1m 35.9s
c1908	699	11.99	9.38	7.02	5.48	4.27	3.15	145m 5s	24.64s
c2670	756	11.45	11.14	9.43	7.62	5.94	4.42	208m 41s	93m 43s
c3540	1466	24.97	18.97	14.53	11.11	8.50	6.38	431m	16h
i10	2643	20.03	18.18	16.30	14.23	12.03	9.75	1668m 44s	11m 9.9s

2.4.2 Single-pass reliability analysis

Simulation results comparing single-pass reliability analysis with Monte Carlo simulations are reported in Table 2.5. In the table, columns 1 and 2 give the name and number of gates in the benchmark circuit. Both the Monte Carlo and single-pass reliability frameworks were used to compute $\delta(\vec{\epsilon})$ for 10 different values of ϵ over the range 0 to 0.5. Note that the same value of ϵ has been used for all the gates in the circuit, and hence $\vec{\epsilon}$ is replaced by ϵ. The third column reports the percentage error in single-pass reliability with 0, 4 and 16 correlation factors averaged over all the outputs and over 10 values of $\epsilon \in [0,0.5]$. The cumulative run-time for 10 runs is reported in the fourth column.

The maximum percentage error in $\delta(\vec{\epsilon})$ is less than 2% for the largest benchmark circuit, i10. For circuits with significant reconvergent fanout, e.g., c499, c1355 and c1908, the maximum percentage error in $\delta(\vec{\epsilon})$ is 13.1%, 13.5% and 6.5% respectively. The largest percentage error is observed when 0 correlation factors are used i.e. the correlations in failures introduced due to reconvergent fanout is ignored. The percentage error progressively improves as 4 and 16 correlation factors are used. Note that the error correcting benchmark circuits like c499, c1355 and c1908 have a lot of reconvergent fanouts that require the use of higher order correlation factors (of order > 1). The circuit structure that introduces higher order correlation factors is shown in Fig. 2.10. Since the higher order correlation factors

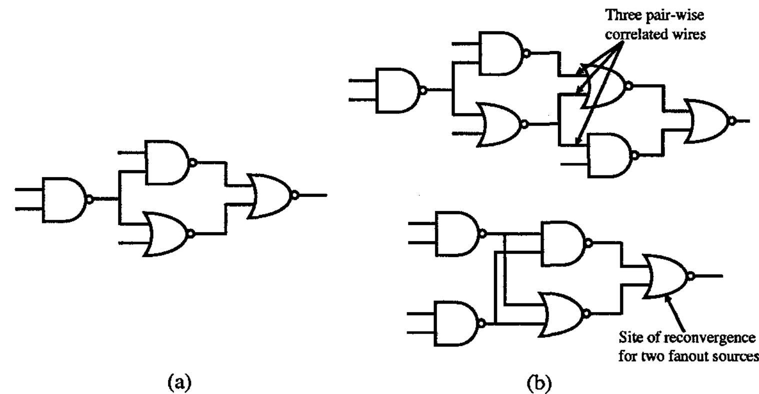


Figure 2.10: Figure shows the circuit structures that introduce (a) first and (b) higher order correlation factors.

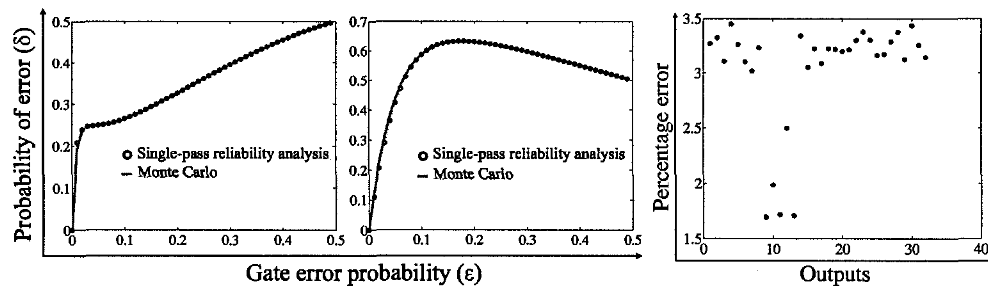


Figure 2.11: $\delta(\vec{\epsilon})$ curves for two outputs of i10. Average error in $\delta(\vec{\epsilon})$ per output of c499 over 1000 runs. On each run, $\epsilon_i \in \text{Uniform}(0, 0.5)$ for each gate.

are approximated in all three schemes (0, 4 and 16), only a slight much improvement in accuracy is observed when 4 and 16 correlation factors are used.

Fig. 2.11 presents $\delta(\vec{\epsilon})$ for two outputs of benchmark i10. The cone sizes of the two outputs are 662 and 1034 gates respectively. Each graph has two curves, one from Monte Carlo reliability analysis and one from single-pass reliability analysis using 0 correlation factors. The two curves are indistinguishable as seen in the figure. The diverse shapes of the curves illustrates not only the complexity of the relation between δ and $\vec{\epsilon}$, but also the accuracy of single-pass reliability analysis.

Fig. 2.11 also shows the percentage error in $\delta(\vec{\epsilon})$ for each of the 32 outputs of benchmark circuit c499. On each run, the ϵ for each gate was derived from a uniform random

distribution over the interval $[0, 0.5]$. Single-pass reliability analysis with 0 correlation factors was compared with Monte Carlo. The percentage error in $\delta(\vec{\epsilon})$ for each output, averaged over 1000 runs, is 1.5–3.5%. This illustrates that the single-pass reliability analysis is highly accurate even when the ϵ values are allowed to vary independently at every gate.

It is clear from the results that the proposed single-pass reliability analysis technique is highly accurate. Although a head-to-head performance comparison with approaches based on PTMs and Bayesian networks was not possible, it is our belief based on the results reported in [26] that the proposed technique affords at least a 500X speed-up over Bayesian networks on the largest circuit b9 (2.5s versus 0.005s (0.046/10s)) reported therein. Note also that results reported in [26] show that Bayesian networks afford a 1000X speed-up over PTMs. In summary, it is reasonable to conclude that the strengths of the proposed single-pass reliability analysis algorithm are its accuracy, scalability to large circuits, and speed-up in performance.

Benchmark	Size	Average error over all outputs and over 10 values of $\epsilon \in [0,0.1]$ (in %)			Runtimes (for 10 runs)			
		0 corr factors	4 corr factors	16 corr factors	Monte Carlo	0 corr factors	4 corr factors	16 corr factors
x2	69	0.86	0.83	0.68	1m47.568s	0.036s	0.029s	0.029s
cu	81	0.32	0.32	0.32	2m13.155s	0.048s	0.028s	0.032s
b9	167	0.42	0.41	0.41	5m25.876s	0.033s	0.048s	0.049s
c499	697	13.1	11.2	11.11	35m4.19s	11.2s	1m14.46s	2m27.78s
c1355	803	13.5	11.59	11.45	35m16.538s	15.8s	1m14.86s	2m17.91s
c1908	718	6.5	7.85	3.97	29m21.616s	0.99s	2m11.017s	5m8.93s
c2670	1073	1.34	1.62	0.95	71m0.280s	1.05s	11.89s	10.97s
frg2	1024	2.96	2.42	1.49	54m23.235s	0.22s	1.62s	2.88s
c3540	1737	4.2	5.27	2.89	118m23.627s	9.68s	17m46.55s	39m4.98s
i10	3442	1.7	1.14	1.12	687m56.562s	17.09s	12m8.68s	24m42.71s

Table 2.5: Comparison of accuracy and runtimes of single-pass reliability analysis with 0, 4 and 16 correlation factor.

2.4.3 Maximum-k gate failure model

The results for reliability analysis under maximum-k gate failure model for different benchmark circuits is shown in Table 2.6. The results have been presented for 3 values of k. Single-pass with 0 correlation factors was used as the core reliability analysis for evaluat-

Benchmark	Size	Runtime for different values of k		
		$k=1$	$k=2$	$k=3$
x2	69	0.29s	0.43s	0.60s
cu	81	0.33s	0.51s	0.71s
b9	167	0.66s	1.02s	1.46s
c499	697	55.15s	57.61s	59.64s
c1355	803	37.39s	49.81s	42.15s
c1908	718	7.99s	10.67s	12.91s
c2670	1073	7.61s	10.27s	13.13s
frg2	1024	5.86s	8.17s	10.75s
c3540	1737	32.68s	37.16s	41.59s
i10	3442	1m40.27s	1m48.28s	2m5.18s

Table 2.6: Runtimes for different benchmark circuits under maximum-k gate failure model. A total of 50K samples were used to evaluate the reliability.

ing the reliability of the 50K samples generated by the sampling algorithm. A significant fraction of the runtime can be attributed to the weight vector computation, ($\mathcal{W}$). For the maximum-k gate failure reliability analysis, $\mathcal{W}$ is computed only once and stored for use in all 50K runs. Thus, the runtime reported for the maximum-k gate failure reliability analysis shown in Table 2.6 is much less than 50K $\times$ the runtimes reported for the single-pass reliability analysis shown in Table 2.5. Thus, the proposed maximum-k gate failure reliability analysis is scalable to large circuits. Another important point to be noted is that the runtimes increase very slowly as k increases. This is because the sampling algorithm complexity is polynomial in k.

2.5 Applications

Single-pass reliability analysis can be used in redundancy-free design space exploration. It is called redundancy-free because no redundancy is used in the circuit to achieve improve-

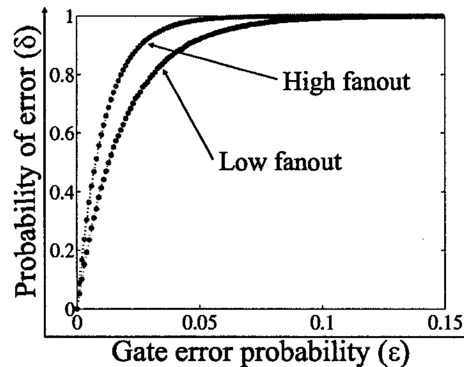


Figure 2.12: Redundancy-free improvements in reliability

ments in reliability. This is illustrated using two synthesized versions of benchmark b9: (i) a low fanout version with a maximum fanout of 2 and (ii) a high fanout version with a maximum fanout of 6. Fig. 2.12 compares the consolidated output error curves for the two versions of b9. The consolidated output error curve gives the probability that at least one of the outputs is in error, and is obtained by performing correlation-based analysis described in Sec. 2.2.1 on the individual $\delta(\vec{\epsilon})$ curves. In Fig. 2.12, $\epsilon \in [0, 0.15]$ because $\delta(\vec{\epsilon})$ for both circuits saturates at 1 for $\epsilon > 0.15$. Note that the same value of ϵ has been used for all the gates, and hence $\vec{\epsilon}$ is replaced by ϵ. It is clear from the figure that the low fanout version of b9 has higher reliability than the high fanout version. This can be explained by examining the levels of logic present in both circuits. The high (low) fanout version of b9 has a maximum of 12 (9) levels of logic and a total of 164 (111) levels of logic over all the outputs. As the number of levels of logic increase, the noise-free inputs have to pass through more levels of noise before they reach the primary outputs. This results in a higher consolidated output error probability.

Single-pass reliability analysis also provides $\delta(\vec{\epsilon})$ curves for each node in the circuit. This information can be used to introduce redundancy at selected gates, instead of introducing redundancy at every gate in the circuit. The proposed analysis technique also provides information about the $0 \rightarrow 1$ and $1 \rightarrow 0$ probability of error separately at each node in the circuit. This is valuable information for explicit introduction of asymmetric redundancy.

For instance, in quadded logic, the redundancy introduced for mitigating a $0 \rightarrow 1$ and $1 \rightarrow 0$ error are different by construction. Single-pass reliability analysis can be used to direct such fine-grained insertion of asymmetric redundancy to enhance reliability at a lower cost.

Observability-based reliability analysis is accurate when the probability of a single gate failure is significantly higher than the probability of multiple gate failures. This makes it directly applicable for soft-error rate estimation in logic circuits because failures due to single-event upsets are usually localized to the gate that is the site of the strike.

Chapter 3

Approximate logic functions

3.1 Concurrent error detection

Error detection has been an important technique for achieving reliability in unreliable systems. Circuits with concurrent error detection (CED) have the capability to detect both temporary and permanent faults and have historically been used in systems where dependability and data integrity are of importance. Systematic codes like parity and SEC/DED have been used to protect data in the main memory for a long time. Systematic codes are desirable because the check bits are appended to the normal bits, and hence no decoding is required to recover the normal bits. Coding techniques for error detection in logic circuits is also a well-researched topic.

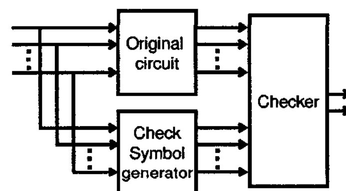


Figure 3.1: Basic design for concurrent error detection.

The basic model for concurrent error detection in logic circuits is shown in Fig. 3.1. It consists of a function logic that generates the normal outputs, the check symbol generator

that produces the check bits and the checker that indicates an error if the combination of normal bits and check bits do not form a codeword. The simplest form of error detection is duplication of the original circuit. However, the area overhead of this technique is unacceptable for main stream applications. An approach based on selective duplication of critical gates is proposed in [21]. This technique is intrusive, technology dependent and may incur a performance penalty on the original design. Time redundancy based techniques [23, 28] have also been explored. Although these techniques have very low area overhead, they incur a huge performance penalty. Another class of approaches explore non-intrusive techniques based on synthesis of an error detection function. However, the exact synthesis of these error detection functions is prohibitively complex. Heuristic synthesis and restrictive error models [2] have been used to simplify the synthesis of the error detection function. These techniques still have a significant area overhead, and do not provide fine-grained control over fault coverage. Systematic codes like parity-check codes for groups of outputs [3, 29], Berger codes [15], Bose-Lin codes [6] have been used to compact the outputs of a circuit. Efficient implementations for the check symbol generators using these codes are known for regular logic structures like PLAs, adders, multipliers [24]. However, for arbitrary multi-level logic the area overhead for the check symbol generators built using these codes is very high (often $>$ 100%). Some of these techniques are also intrusive because they impose constrains on the synthesis of the original circuit. Conventional techniques for synthesis of CED for multi-level logic circuits focus on guaranteeing 100% coverage of broad classes of errors and generally require very large area, power, and performance overhead (often in excess of 100%) [6, 12, 15, 18, 24, 28, 29]. For high-volume mainstream applications, most of these existing CED solutions will be overkill. Recent research has seen the emergence of CED techniques that try to meet coverage requirements at minimum cost instead of trying to guarantee coverage of broad classes of errors, e.g., [3, 21]. However, these techniques have one or more disadvantages that limit their integration into standard design flows. These include limited scalability and options to trade-off coverage for overhead, requiring modifications to or constraining synthesis of the original design

(intrusive CED), performance penalty due to longer critical path delay in CED logic, and technology-dependent synthesis.

In section 3.2, a scalable, technology-independent algorithm for the synthesis of a new class of circuits called approximate logic circuits is described. An approximate logic circuit is constructed by selectively converting minterms from the off-set or on-set of its Boolean function into don't cares. The idea is that by converting only a small fraction of minterms into don't cares, an approximation of a Boolean function can be synthesized with a significantly lower area-power-delay footprint than the exact Boolean function. Since brute-force enumeration and conversion of minterms into don't cares does not scale with circuit complexity, an efficient algorithm that manipulates a technology-independent multi-level network to realize approximate logic circuits is described in this thesis. The algorithm iteratively reduces the Boolean expressions of the nodes in the multi-level network using two cube selection techniques based on exact cubes and observability don't care cubes.

A low overhead, non-intrusive solution for CED based on approximate logic circuits is described in this thesis. The proposed synthesis algorithm for approximate logic circuits provides fine-grained trade-offs between area-power overhead and CED coverage. Since CED is non-intrusive, no modifications are necessary to the original design. A self-checking checker that is compatible with the proposed CED technique is also described. The checker produces two-rail encoded outputs, ensuring compatibility with other error detection techniques for error signal consolidation. Simulation results for 8 benchmark circuits indicate that CED coverage of 81% can be achieved for 25% and 34% area and power overhead on average.

CED based on approximate logic circuits incurs no performance penalty, since approximate logic circuits are significantly smaller than the original circuit. For the benchmarks studied in this thesis, the delay of the approximate logic circuit was 38% less than the delay of the original circuit on average. This can be leveraged to direct synthesis to obtain extensive CED coverage to errors caused by stuck-at as well as delay faults on speed-paths in the original logic circuit. By covering a significant fraction of the minterms that sensi-

tize the longest paths in the design, CED detects errors that arise from stuck-at as well as delay faults on such speed-paths. Simulation results for the top 5% (30%) of the longest paths indicate that CED achieves 82.4% (74%) coverage when these paths are sensitized, for 23.6% and 27.8% (29.8% and 36.5%) area and power overhead on average.

Finally, the synthesis algorithm is extended to include logic sharing between the original and approximate logic circuits to further reduce overhead. Results show that for the same coverage, CED using approximate logic circuits and logic sharing always requires a lower overhead in comparison to the best known intrusive technique for CED.

3.2 Synthesis of approximate logic functions

Given a Boolean function $\mathcal{F}$, a Boolean function $\mathcal{G}$ is a 0-approximation of $\mathcal{F}$ *iff* $\overline{\mathcal{G}} \Rightarrow \overline{\mathcal{F}}$. Similarly, $\mathcal{G}$ is a 1-approximation of $\mathcal{F}$ *iff* $\mathcal{G} \Rightarrow \mathcal{F}$. If all the input vectors are equally likely, the *approximation percentage* can be defined as the fraction of minterms that are covered by the approximate function in the 1(0)-minterm space of the exact function $\mathcal{F}$. If the input vectors are not equally likely, then the minterms that are covered by the approximate function must be appropriately weighted by its probability of occurrence to compute the approximation percentage.

Approximate logic functions are interesting because a very high approximation percentage can be achieved with a low area overhead. This can be illustrated with the following example. Consider the function $\mathcal{F} = a + b + \overline{c}\overline{d} + cd$. An optimal implementation of this function consists of 7 gates. A 1-approximation of this function is given by the function $\mathcal{G} = a + b$. The implementation of $\mathcal{G}$ requires only 1 gate. Assuming that all input vectors are equally likely, $\mathcal{G}$ covers 12 out of 14 minterms in $\mathcal{F}$. Thus, an approximation percentage of 85.72% can be achieved for an area overhead of 1/7 or 14.28%. The rest of this section describes an algorithm for synthesis of approximate logic functions.

The algorithm for approximate logic synthesis starts with a multi-level, technology-independent network of the original circuit [13]. The network is an interconnection of

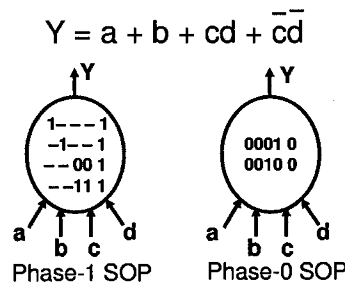


Figure 3.2: Representation of SOPs in the 0/1 phase.

nodes in a directed acyclic graph where each node has two kinds of Boolean functions associated with it: (i) a function of its local inputs referred to as the local Boolean function of the node, and (ii) a function of the primary inputs to the circuit referred to as the global Boolean function of the node. The local Boolean function of nodes in the network can be expressed as a sum-of-products (SOP) expression. Every SOP can be written either in the zero phase (denoting the off-set) or in the one phase (denoting the on-set) (ref. Fig. 3.2). A zero phase SOP can be converted to a one phase SOP and vice-versa using DeMorgan's law.

The approximate function is either a 0 or a 1-implication of the output of the original circuit. Although implication-based techniques for synthesis, verification, and test have been proposed in literature, e.g., [19], such techniques search for implication relations among Boolean functions implemented within the original circuit. In contrast, the synthesis algorithm proposed in this thesis is technology-independent and searches for implications that may not be a part of the original circuit. The algorithm for synthesis of approximate logic functions is divided into 2 stages: (i) Type assignment: Assigning type of approximation (0/1) to each node in the multi-level network, and (ii) Cube selection: Constrained approximation of each node in the multi-level network. The next two sections describe the two stages of the synthesis algorithm in detail.

3.2.1 Type assignment

The type assignment is done as a preprocessing step prior to cube selection. The aim of type assignment algorithm is to determine the type of approximation (0/1) that must be made at each node in the technology-independent network to maximize the approximation percentage at the primary output.

Type assignment uses the notion of the 0/1-observability of a node. The *0(1)-observability of a fanin node* is defined as the probability that a 0(1) value at the fanin is observable at the output of the node. The intuition behind assigning type to a node is that if the 0-observability is dominant, then a 0-approximation of the node would yield a high approximation percentage at the output of the node, and vice-versa. There are 4 kind of types that can be assigned to the nodes: 0, 1, EX, DC. Type 0(1) means that the 0(1)-minterm space of the global Boolean function at the node is crucial to the approximation of a primary output. Type EX means that both 0 and 1-minterm spaces are crucial to the approximation of a primary output. Finally, type DC means that neither 0 nor 1-minterm space is crucial to the approximation of any primary output.

The type assignment algorithm is initialized by ascertaining the type of approximation desired at each primary output of the circuit. The processing of a node involves assigning a type to the node and then analyzing the observabilities of the fanin nodes to *request* a type for each of its fanins. The algorithm processes the nodes in the network in a reverse topological order, i.e., a node is assigned a type after all its fanouts have been assigned a type. Nodes are assigned types in such an order because the type assignment of a node depends on the types requested by its fanout nodes. The type assignment of a node is done based on the following rules:

- Any fanout node requests type EX, node is assigned type EX.
- All fanout nodes request type DC, node is assigned type DC.
- All fanout nodes request type 0/DC, node is assigned type 0.
- All fanout nodes request type 1/DC, node is assigned type 1.

- Else, node is assigned type EX.

After assigning a type to the node, a request for the type of each fanin node is made based on the 0/1-observability of the fanin nodes. (i) If both 0/1-observability of a fanin node are small as compared to other fanin nodes, a type DC is requested for that fanin. (ii) If there is a big disparity between the 0-observability and 1-observability (based on the ratio of the two observabilities) of the fanin, then the type with the higher observability is requested for the fanin. (iii) If the 0-observability and 1-observability of the fanin are comparable, then a type EX is requested for that fanin.

3.2.2 Cube selection

In the preprocessing stage described above each node was assigned a type to maximize the approximation percentage. Approximation of the Boolean function at the fanins of a node may cause the bit(s) in the input vector to the node to change. A (1)0-approximation may be rendered incorrect due to such bit flips if an input vector in the 0(1)-minterm space transforms into an input vector in the 1(0)-minterm space. The 1(0)-minterm space of a type 1(type 0) node must be reduced so as to adapt for the bit flips and ensure correctness of approximation. This subset of the 1(0)-minterm space will be referred to as the feasible subspace of the node. Note that the feasible subspace of a node will be larger when lesser number of bit flips occur, i.e., the approximation at the fanin nodes are close to the exact functions. Since the feasible subspace depends on the approximations at the fanin nodes, explicit computation of the feasible subspace is prohibitively complex.

The goal of cube selection is two fold: (i) to ensure correctness of approximation and (ii) to achieve a high approximation percentage for low overhead. Two techniques for cube selection are described in this thesis. The first technique, exact cube selection, guarantees the correctness of the approximate function by computing a subset of the feasible subspace. However, this technique may affect the approximation percentage because it imposes strict constraints for selecting cubes to guarantee correctness. The second technique, observability don't care based cube selection, relaxes the constraints for cube selection using observ-

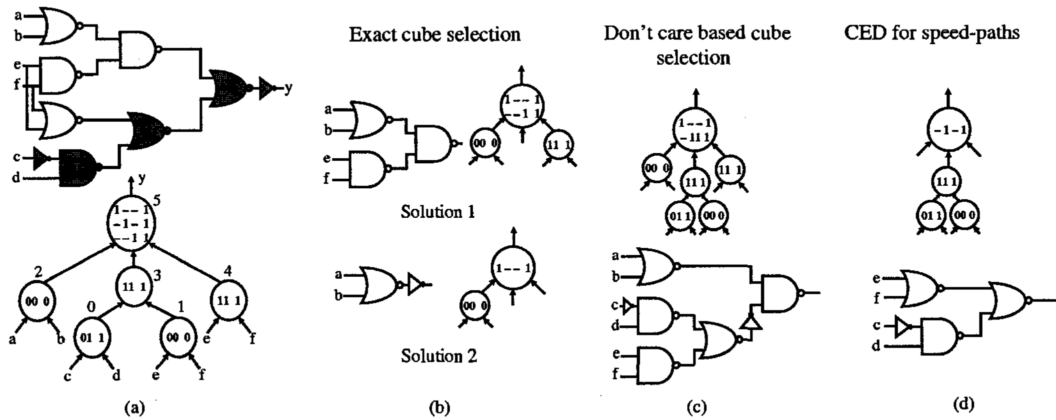


Figure 3.3: This example illustrates the two cube selection algorithms for synthesis of approximate logic circuits (Figs. b and c). It also illustrates synthesis driven towards high CED coverage on speed-paths (Fig. d).

ability don't cares. However, this relaxation may result in the inclusion of minterms outside the feasible subspace, and hence it does not guarantee correctness. To ensure correctness while maintaining a high approximation percentage, an algorithm for cube selection that uses these two techniques iteratively is proposed.

Theorem: Given Boolean functions X_1, X_2, $\mathcal{F} = X_1X_2$ and 1-approximate Boolean functions X_1', X_2' for X_1 and X_2, then $\mathcal{F}' = X_1'X_2'$ is a 1-approximation for $\mathcal{F}$.

Proof: Since X_1', X_2' are 1-approximations for X_1, X_2, $X_1' \Rightarrow X_1$ and $X_2' \Rightarrow X_2$. Thus,

$$\overline{X_1'} + X_1 = 1,\ \overline{X_2'} + X_2 = 1 \Leftrightarrow (\overline{X_1'} + X_1)(\overline{X_2'} + X_2) = 1$$
$$\Leftrightarrow \overline{X_1'}\,\overline{X_2'} + \overline{X_1'}X_2 + \overline{X_2'}X_1 + X_1X_2 = 1$$
$$\Leftrightarrow \overline{X_1'}(\overline{X_2'} + X_2) + \overline{X_2'}(\overline{X_1'} + X_1) + X_1X_2 = 1$$
$$\Leftrightarrow \overline{X_1'} + \overline{X_2'} + X_1X_2 = 1 \Leftrightarrow \overline{X_1'X_2'} + X_1X_2 = 1$$
$$\Leftrightarrow X_1'X_2' \Rightarrow X_1X_2 \Leftrightarrow \mathcal{F}' \Rightarrow \mathcal{F}$$

In other words, $\mathcal{F}'$ is a 1-approximation of $\mathcal{F}$. Similarly, we can prove that if $\mathcal{F} = X_1 + X_2$ then $\mathcal{F}' = X_1' + X_2'$ is a 1-approximation of $\mathcal{F}$. Note that both the above results also hold true for 0-approximations, i.e., if X_1', X_2' are 0-approximations of X_1, X_2 then (i) $\mathcal{F}' =$

$\overline{X_1'}\,\overline{X_2'}$ is a 0-approximation of $\mathcal{F} = X_1X_2$, and (ii) $\mathcal{F}' = \overline{X_1'} + \overline{X_2'}$ is a 0-approximation of $\mathcal{F} = X_1 + X_2$. The above theorems can be generalized to n variables using induction on n. Using this theorem, it can be proved that appropriate selection of cubes from the SOP of internal nodes in a technology-independent network gives rise to the correct approximation at the primary outputs. This result ensures correctness of exact cube selection.

Exact cube selection: This technique derives an approximate logic function by picking a subset of cubes from the SOP expression of type 0 and type 1 nodes. Type EX and type DC nodes are left untouched. The idea is to pick cubes to maximize the approximation percentage for low overhead. The phase of the SOP expression used for cube selection must match the node type for correctness of approximation. For instance, if the node type is 0, the SOP expression must be written in the zero phase before selecting the cubes.

A cube is said to conform to a fanin node of type 0(1) if the literal in the cube corresponding to the fanin node is a '0'('1') or '–' (don't care). A cube conforms to a fanin node of type DC if the corresponding literal in the cube is '–'. Every cube conforms to a fanin node of type EX. A cube is selected only if it conforms to the type assignment of every fanin node. It is proved using the theorem above that this method always generates a correct approximate function. This method uses only a subset of the feasible subspace because it does not exploit the observability don't care space to identify cubes in the feasible subspace that do not conform to the fanin node types. This is exploited in the cube selection method based on observability don't cares described below.

Observability don't care based selection: This technique uses local observability don't cares to expand the space from which cubes can be selected. Local observability don't cares refers to the observability don't care space with respect to the output of the node, as opposed to the primary output of the circuit (deriving which is computationally intensive). Eqn. 3.1 shows the Boolean expression for computing the feasible subspace. For the sake of clarity, the Boolean expression for a node with two fanin nodes, x_1 of type 1, x_2 of type

0 is shown.

$$\begin{array}{rl} \mathcal{F}(x_1 + \overline{\text{Obs}_{x_1}})(\overline{x_2} + \overline{\text{Obs}_{x_2}}) & \text{for a node of type 1} \\ \mathcal{F} + \left(\overline{(x_1 + \overline{\text{Obs}_{x_1}})(\overline{x_2} + \overline{\text{Obs}_{x_2}})}\right) & \text{for a node of type 0} \end{array} \tag{3.1}$$

$\mathcal{F}$ is the local boolean function of the node. The term $(x_1 + \overline{\text{Obs}_{x_1}})$ picks cubes that either conform to the type of the fanin node x_1 or those in which x_1 is not observable. For a node of type DC, only the observability don't care term is used. As long as only a single input bit flips in the input vector of a node, this technique guarantees correctness. However, multiple bit flips may render the approximation incorrect because the effect of multiple bit flips is not captured accurately by this technique. The Boolean AND of the observabilities of the bits is used instead of their joint observability (which is computationally demanding). Hence, minterms outside the feasible subspace may get included in this technique. The chance of multiple bit flips occurring at the inputs of a node is low when the Boolean function at the inputs have been well approximated. Note that the computational complexity of both cube selection algorithms is linear in the size of the technology-independent network.

Example: The cube selection algorithms are illustrated with an example. Fig. 3.3(a) shows a circuit and its technology-independent representation. Approximation of this network obtained using the exact cube selection algorithm is shown in Fig. 3.3(b). Two approximate circuits are shown that exhibit a trade-off between approximation percentage and area overhead. In one implementation node 2 and node 5 were assigned type 1 and the rest of the nodes were assigned type DC. Thus, only one cube was selected from the SOP of node 5 that conformed to the type assignment of its fanins (nodes 2, 3, 4). For the same type assignment, cube selection based on observability don't cares (Fig. 3.3(c)) discovered an additional cube -11. This minterm is acceptable in the observability don't care based algorithm because the type DC nodes (3 and 4) are not observable in this minterm. Note that cube selection based on observability don't cares gives the solution shown in Fig. 3.3(b) in addition to the solutions in Fig. 3.3(a).

3.2.3 Iterative cube selection algorithm

Both techniques for cube selection proposed above have their merits and demerits. The exact cube selection algorithm guarantees the correctness of the approximate function. However, this is accomplished at the cost of imposing strict constraints on cube selection, thus affecting the approximation percentage. The observability don't care based cube selection approach relaxes the constrains on cube selection but does not guarantee correctness of the approximate function. Here, we propose an algorithm that uses these techniques iteratively to achieve correctness while maintaining a high approximation percentage. The algorithm takes as input a network in which the type assignment of nodes has been done. The SOPs of type 0 and type 1 nodes are rewritten in the correct form so that the phase of the SOP matches the type of the node. The algorithm is split into two stages:

Approximation of SOPs: The SOP of every node (including type EX and type DC) is reduced by discarding cubes whose contribution to the Boolean function is insignificant. Usually, the cubes with a large support set fall into this category. For instance, in the Boolean function $a + \bar{b}c + b\bar{c}d\bar{e}$, the contribution of the cube $b\bar{c}d\bar{e}$ is least significant and is discarded. This cube selection is done freely without any constraint to ensure correctness. The aim of this stage is to lower area overhead while maintaining a high approximation percentage. The primary outputs of the circuit are then checked for correctness of approximate functions. This can be done very efficiently using SAT algorithms, or by checking the implication condition for correct approximation using BDDs of the original and approximated outputs. If all the outputs have been correctly approximated, then we are done. Otherwise, the outputs that have been incorrectly approximated are corrected in the second stage.

Ensuring correctness: The input to this stage is a primary output of the circuit that has been incorrectly approximated by the previous stage. The first step in correcting the approximation at the output is to perform a backward traversal of the circuit to identify a possible source of incorrect approximation. A node is a source of incorrect approximation if

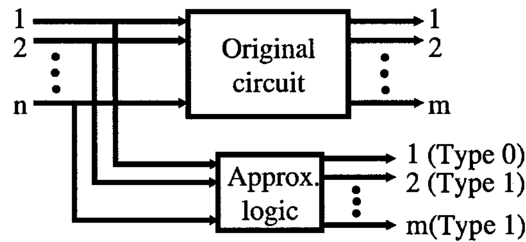


Figure 3.4: Figure illustrates the application of approximate logic functions to concurrent error detection.

the Boolean function of the node has been incorrectly approximated but all its fanin nodes have been correctly approximated. The first attempt at correcting the approximation at this node is made by selecting the cubes using observability don't cares. If it fails to correct the approximation at the node, the exact cube selection is used, which guarantees a correct approximation. The approximation at the primary output is checked again for correctness, and this stage is repeated for other sources of incorrect approximation until it is fixed.

Note that it is possible that the primary outputs of the circuit are correctly approximated, but there are internal nodes nodes that are not approximated correctly. This means that an error in approximation of an internal node is not observable at the primary output of the circuit. Thus, the algorithm is able to exploit the global observability don't care space of the internal nodes.

The computational complexity of the iterative algorithm depends on the amount of backtracking that needs to be performed in order to ensure correctness of approximation. For the benchmark circuits considered in this thesis, the primary outputs were correctly approximated after the first stage of the algorithm in most cases, i.e., there was no need for backtracking to fix the approximations at the nodes. This can be attributed to the type assignment done in the preprocessing stage. Thus, a good type assignment and a high approximation percentage at the internal nodes makes it possible to achieve correctness without any backtracking.

3.3 Approximate logic circuits for CED

3.3.1 Conventional CED

The basic flow for the application of approximate logic functions to CED in logic circuits is described in this section. A technology-independent network is given as input. For every primary output of the circuit, either a 1-approximate or a 0-approximate function is used for detection of $1 \rightarrow 0$ or $0 \rightarrow 1$ errors. The type of approximate function to be used for a primary output is decided by the type of error ($0 \rightarrow 1$ or $1 \rightarrow 0$) that dominates at that output. This can be done by running any reliability analysis algorithm, e.g., [5, 16]. In this thesis, we have used the reliability analysis technique described in [5] to compute the $0 \rightarrow 1$ and $1 \rightarrow 0$ errors at the outputs. The approximate logic function is then synthesized using the algorithm described in Sec. 3.2.3. The synthesized approximate function serves as error detection logic for the original circuit (ref Fig. 3.4).

The techniques for reliability analysis require a technology-mapped netlist as input. This is done using a quick synthesis and mapping pass on the multi-level technology-independent network. In Sec. 3.5.3, we show that CED coverage is mostly insensitive to the scripts used for synthesis of the original circuit as well as the specific library used for mapping prior to reliability analysis.

3.3.2 CED for speed-paths

The longest sensitizable paths in a logic circuit, also called speed-paths, are vulnerable to delay faults caused by delay defects and failure mechanisms such as transistor wearout. By injecting errors on only a set of speed-paths, the reliability analysis algorithm can identify the most vulnerable output cones in the logic circuit. The synthesis algorithm can then be biased to include minterms that sensitize these speed-paths on the selected output cones, which increases CED coverage to stuck-at and delay faults on such speed-paths. Fig. 3.3 illustrates the application of approximate logic functions to maximize CED coverage on speed-paths. The gates on the longest paths (5 gate delays) in the circuit are shaded in

Fig. 3.3(a). The minterms of the Boolean expression $\bar{c}d(e+f)$ sensitize the longest path in the logic circuit. Fig. 3.3(d) is a 1-approximate circuit that can be used to detect errors that occur on this speed-path in the original circuit. Since, the 1-approximate circuit has a longest path with only 3 gate delays, it is robust to errors arising due to delay defects or transistor wearout. Results for CED coverage on speed-paths using approximate logic circuits are presented in Sec. 3.5.2.

3.3.3 Low overhead intrusive CED

Although the discussion so far has focused on using approximate logic circuits for non-intrusive CED, further reductions in area-power overhead can be achieved as follows. By merging structurally or functionally equivalent nodes between the original and the approximate logic circuit, it is possible to trade-off CED coverage for overhead. This sharing of logic between the original and approximate logic circuits makes CED intrusive. Partial duplication for CED described in [21] can be viewed as a special case of approximate logic functions with logic sharing. In partial duplication, the approximate logic function has a 100% approximation percentage and non-critical nodes are shared between the original and the approximate logic circuits. Thus, results for CED using partial duplication are a lower bound for the proposed technique with logic sharing and are presented in Sec. 3.5.

3.4 Self-checking checker design

Checker design is an integral part of concurrent error detection. The function of the checker is to monitor the output of the circuit and the check symbol generator, and to signal an error when they do not form a valid codeword. A self-checking checker should be code-disjoint, fault-secure, and self-testing w.r.t a specified fault class. Code-disjointness ensures that the checker gives an invalid output codeword when an invalid codeword is presented at its inputs. Fault-secureness ensures that a fault within the fault class either gives the fault-free response or an invalid output codeword. Self-testing ensures that all the faults in

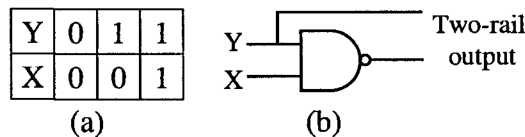


Figure 3.5: Self-checking checker design for the proposed error detection technique.

the specified fault class are testable under normal operation. Consider a circuit with an output Y protected by a 0-approximate logic function X. Denote the Boolean functions at Y and X by $\mathcal{F}_y$ and $\mathcal{F}_x$. The design of the self-checking checker is shown in Fig. 3.5. The fault class includes all single stuck-at faults at the outputs of the checker and at the inputs and output of the nand gate (except X stuck-at 1). This is because of implication relation between X and Y. The valid input codeword space for the checker is shown in Fig. 3.5(a). Since, X is a 0-approximation of Y, the input codeword space does not contain $X = 0, Y = 1$. The output codeword space is the two-rail code, i.e., $\{01, 10\}$. This is desirable because the outputs of the checkers can then be consolidated together using a two-rail code checker. It is evident from Fig. 3.5(a) that the checker is code-disjoint. The fault-secure and self-testing properties can be verified by enumeration of all single stuck-at faults in the checker in Fig. 3.5(b) and the input codewords. Note that since the checker is an irredundant logic circuit, it is completely testable for all single stuck-at faults during offline testing using all 4 input vectors to the checker.

3.5 Results

In this section we present simulation results for CED using approximate logic circuits. The framework for synthesizing approximate logic circuits was implemented in ABC, the logic synthesis tool developed at Berkeley [1]. The simulations were run on a 64-bit 2.4 GHz Opteron based system with 6 GB memory. The results for error detection in MCNC benchmark circuits are presented in this section. The inputs to the circuits are assumed to have an equal probability of occurrence, i.e., there is no input vector biasing.

3.5.1 Conventional CED

The results in Table 3.1 and Table 3.2 are reported for the single stuck-at fault model with all the gates in the circuit having the same probability of failure. Table 3.1 shows results for single output cones extracted from MCNC benchmarks. The first column is the name of the benchmark circuit from which the output cone was extracted. The second column is the number of gates in the output cone. The third column is the area overhead of the synthesized approximate logic circuit. The fourth column is the approximation percentage achieved by the approximate logic function. The fifth column is the maximum error detection percentage that can be achieved by protecting the dominant error ($0 \rightarrow 1$ or $1 \rightarrow 0$) at the output. The sixth column reports the error detection percentage, i.e., the CED coverage achieved by the synthesized approximate logic circuit. The results illustrate the effectiveness of the approximate logic functions in achieving a high approximation percentage for low area overhead. The disparity in the approximation percentage and CED coverage for circuits des and i8 is because CED coverage is limited by the amount of skew in the type of errors ($0 \rightarrow 1$ vs. $1 \rightarrow 0$) at the output. The results for error detection in complete

Name	Num. gates	Area (%)	Approx. (%)	CED coverage (%)	
				Max.	Achieved
i8	106	28	80	65	50
des	191	2.7	95.6	56	48
dalu	862	25	93.8	85	71
i10	1141	1.5	91	76	64

Table 3.1: Approximation percentage and CED coverage for output cones extracted from benchmark circuits.

MCNC benchmark circuits are shown in Table 3.2. The proposed synthesis algorithms for approximate logic functions were evaluated on logic benchmarks from this suite. Logic benchmarks with a reasonably large skew in the errors at the outputs have been chosen. Results for completely non-intrusive approximate logic functions are shown under the col-

umn "No logic sharing". Results with merging of non-critical nodes is shown under the column "With logic sharing". The results for error detection are compared with intrusive CED based on partial duplication. The columns of Table 3.2 have the same meaning as in Table 3.1. The proposed technique without logic sharing and partial duplication approach were tuned so as to match their CED coverage. The results show that the same CED coverage can be achieved with the proposed technique with an area overhead that is lower than that for partial duplication. Furthermore, the proposed technique is completely non-intrusive and incurs zero performance penalty because the approximate logic function always has a lower delay on the critical path. For the benchmarks studied in this thesis, the delay of the approximate logic circuit was 38% less than the delay of the original circuit on average.

Name	Gate count	Max. CED coverage	No logic sharing			With logic sharing		Partial duplication		
			Area	Power	CED coverage	Area	CED coverage	Area	Power	CED coverage
cmb	57	99.7	32	26	98	29	98	48	32	98
cordic	116	88	28	37	82	24	82	26	22	82
terml	260	82	15	25	71	13	70	17	19	70
x1	442	78	36	45	68	26	65	30	37	68
i2	440	89	5	6	84	3	83	6	4	82
frg2	1089	90	30	47	80	22	75	46	48	79
dalu	1166	92	21	35	80	15	77	44	44	77
i10	2866	85	36	56	81	30	77	54	49	81

Table 3.2: Approximate percentage and CED coverage for MCNC benchmark circuits.

3.5.2 CED for speed-paths

The results for CED coverage on speed-paths are shown in Table 3.3. The first two columns are the name and the number of gates in the benchmark circuit. CED coverage is reported in the third column as the coverage achieved on the minterms that sensitize the speed-paths (top 5% and top 30% longest paths) in the original circuit. The fourth column is the reduction in delay achieved by the synthesized approximate logic circuit as compared to the original circuit. The fifth and sixth columns report the area and power overheads for CED. The results show that a significant fraction of the minterms that sensitize the speed-paths can be covered for low area-power overhead. Moreover, since the approximate logic circuit

has a lower delay than the original circuit, it is not vulnerable to delay faults on its critical path.

Name	Gate count	Top 5 % longest paths				Top 30 % longest paths			
		Coverage	Delay improvement	Overhead		Coverage	Delay improvement	Overhead	
				Area	Power			Area	Power
cmb	57	85	50	32	26	82	50	32	26
cordic	116	73	11	35	40	76	39	35	38
terml	260	67	41	14	23	70	27	24	40
x1	442	82	11	16	24	55	21	25	33
i2	440	89.5	23	14	18	92	23	14	16
frg2	1089	85	11	45	55	68	39	40	52
dalu	1166	100	22	19	33	75	26	26	42
i10	2866	78	64	14	20	75	28	27	45

Table 3.3: Minterm coverage achieved for CED on speed-paths.

3.5.3 Technology-independence

In this section we present results in Table 3.4 to show that the CED coverage of the proposed technique is not significantly affected by the (i) synthesis scripts used to optimize and map the original and approximate logic circuits or (ii) the library used to map and perform reliability analysis on the original circuit. Initially, reliability analysis was performed on a netlist obtained by quick synthesis to determine the type of approximation for each output. The results of reliability analysis were used to perform synthesis of the approximate logic circuits. Five different technology-mapped implementations of the original circuit were generated using different optimization scripts in ABC and different technology libraries. The same approximate logic function (mapped with the technology library of the original circuit) was used to provide CED for each of the implementations. The area overhead was maintained nearly the same for the different implementations. Table 3.4 shows that the CED coverage remains fairly constant for different technology-mapped implementations of the original and approximate logic circuits. Thus, we can conclude that the effectiveness of CED achieved using the proposed technique depends mainly on the Boolean function being approximated, i.e., it is technology-independent.

Name	CED coverage %				
	Impln 1	Impln 2	Impln 3	Impln 4	Impln 5
cmb	95.8	96	96.6	95.1	96.7
cordic	74	74.5	74.1	74.6	73
term1	70	73	75	80	71
x1	67.8	68.6	64.1	64.5	68
i2	79	84	82	85	83
frg2	70	69	71.3	76.1	75.2
dalu	71.2	72.1	73	72.4	75
i10	70	71.2	70.5	71.7	72.2

Table 3.4: Technology-independence of CED coverage

Chapter 4

Conclusions and future work

Even as reliability gains wide acceptance as a significant design challenge, there is a lack of effective techniques for its analysis and optimization. This thesis described two accurate, scalable, and highly efficient techniques for reliability analysis of logic circuits under the independent gate failure model. The observability-based algorithm provides a closed-form expression for reliability using the observabilities of the gates. This closed-form expression is accurate in circuits where single gate failures are dominant. It is extremely efficient and flexible because there is only static processing in the form of closed-form expression computation of the observability information of gates. The single-pass reliability analysis algorithm described in this thesis uses a single topological walk through the circuit to compute the reliability of the circuit. The algorithm is accurate even for multiple gate failures. In fact, the single-pass reliability analysis is exact in the absence of reconvergent fanout. The correlations introduced due to the reconvergent fanout are handled using correlation factors. The accuracy vs. runtime complexity is analyzed for 0, 4 and 16 correlation factors.

Under an independent gate failure model, there is no control over the number of simultaneous gate failures that can occur. To address this problem, an improved gate failure model to limit the number of simultaneous gate failures is proposed. This is called the maximum-k gate failure model. An efficient reliability analysis algorithm that uses single-pass algorithm as its core reliability engine is also described for this model. These

techniques have several potential applications, including redundancy-free reliability optimization, asymmetric and fine-grained redundancy insertion, and reliability-driven design optimization.

This thesis also describes an efficient algorithm for the synthesis of approximate logic functions. Single-pass reliability analysis is used to guide the synthesis algorithm for approximate logic functions towards achieving a low cost, non-intrusive error detection technique. The proposed CED technique is technology-independent and incurs no performance penalty. A key advantage of this approach is the ability to detect errors caused by emerging failure mechanisms such as transistor wear-out on speed-paths in logic circuits.

Following are the directions for future research:

1. Generalize the approach for error detection using approximate logic functions by removing the implication restriction and having two approximate functions for every output. The minterms that are covered by the approximate logic function would be guided by the reliability analysis and input pattern simulation on the technology-independent network.

2. Explore reliability-aware design techniques like error detection, error masking at other stages of logic design including:

    (a) Decomposition of a technology-independent network

    (b) Technology mapping

    (c) Post-mapping transformations

3. Explore achievable bounds on circuit reliability to bridge the gap between theoretical limits to computation in the presence of noise and achievable limits using existing fault tolerant approaches.

Bibliography

[1] ABC Logic synthesis tool. Please visit the URL http://www.eecs.berkeley.edu/ alanmi/abc/ for further details.

[2] S. Almukhaizim, P. Drineas, and Y. Makris. Concurrent error detection for combinational and sequential logic via output compaction. In *Proc. Intl. Symposium on Quality Electronic Design*, pages 459–464, 2004.

[3] S. Almukhaizim, P. Drineas, and Y. Makris. Cost driven selection of parity trees. In *Proc. VLSI Test Symposium*, pages 319–324, 2004.

[4] S. Borkar. Designing reliable systems from unreliable components: The challenges of transistor variability and degradation. *IEEE Micro*, 25(6):10–16, 2005.

[5] M. Choudhury and K. Mohanram. Accurate and scalable reliability analysis of logic circuits. In *Proc. Design Automation and Test in Europe*, pages 1454–1459, 2007.

[6] D. Das and N.A. Touba. Synthesis of circuits with low cost concurrent error detection based on bose-lin codes. volume 15, pages 145–155, 1999.

[7] J. D.Meindl, Q. Chen, and J. A. Davis. Limits on silicon nanoelectronis for terascale integration. *Science*, 293:2044–2049, Sep. 2001.

[8] S. Ercolani et al. Estimate of signal probability in combinational logic networks. pages 132–138, 1989.

[9] W. Evans and N. Pippenger. On the maximum tolerable noise for reliable computation by formulas. In *Proceedings of IEE transactions on Information theory*, pages 1299–1305, 1998.

[10] G. Fishman. *Monte Carlo*. Springer series in operations research, 1995.

[11] J. Gao and J. Fortes. Bifurcations and fundamental error bounds for fault-tolerant computations. In *IEEE Transactions on Nanotechnology*, volume 4, pages 395–402, 2005.

[12] M. Gossel and S. Graf. *Error detection circuits*. McGraw-Hill Book Company, London, UK, 1993.

[13] G.D. Hachtel and F. Somenzi. *Logic synthesis and verification*. Kluwer Academic Publishers, 2000.

[14] R. Hegde and N.R. Shanbhag. Towards achieving energy-efficiency in presence of deep submicron noise. 8(4):379–391, Aug. 2000.

[15] N. K. Jha and S. Wang. Design and synthesis of self-checking VLSI circuits. *IEEE Trans. Computer-aided Design*, 2(1):878–887, 1993.

[16] S. Krishnaswamy et al. Accurate reliability evaluation and enhancement via probabilistic transfer matrices. In *Proc. Design Automation and Test in Europe*, pages 282–287, 2005.

[17] S. Krishnaswamy et al. Enhancing design robustness with reliability-aware resynthesis and logic simulation. In *Proc. Intl. Conference Computer-aided Design*, 2007. To appear.

[18] Sandip Kundu and Sudhakar M. Reddy. On symmetric error correcting and all unidirectional error detecting codes. *IEEE Trans. Computers*, 39(6):752–761, Jun. 1990.

[19] W. Kunz and D.K. Pradhan. Recursive learning: A new implication technique for efficient solutions to CAD problems — Test, verification, and optimization. In *IEEE Trans. Computer-aided Design*, volume 13, pages 1143–1158, 1994.

[20] T. Larrabee. Test pattern generation using Boolean satisfiability. *IEEE Trans. Computer-aided Design*, 11(1):4–15, Jan. 1992.

[21] K. Mohanram and N. A. Touba. Cost-effective approach for reducing soft error failure rate in logic circuits. In *Proc. Intl. Test Conference*, pages 893–901, 2003.

[22] K. Mohanram and N. A. Touba. Partial error masking to reduce soft error failure rate in logic circuits. In *Proc. Defect and Fault Tolerance Symposium*, pages 433–440, 2003.

[23] M. Nicolaidis. Time redundancy based soft error tolerance to rescue nanometer technologies. In *Proc. VLSI Test Symposium*, pages 86–94, 1999.

[24] D. K. Pradhan, editor. *Fault-tolerant computer system design*. Prentice-Hall, Inc., NJ, USA, 1996.

[25] Y. Qi, J. Gao, and J. Fortes. Markov chains and probabilistic computation – a general framework for multiplexed nanoelectronic systems. In *IEEE Transactions on Nanotechnology*, volume 4, pages 194–205, 2005.

[26] T. Rejimon and S. Bhanja. Scalable probabilistic computing models using Bayesian networks. pages 712–715, 2005.

[27] A. Sadek, K. Nikolić, and M. Forshaw. Parallel information and computation with restitution for noise-tolerant nanoscale logic networks. *Nanotechnology*, 15(1):192–210, Jan. 2004.

[28] Vl. V. Saposhnikov et al. Self-dual duplication for error detection. In *Proc. Asian Test Symposium*, pages 296–300, 1998.

[29] N. A. Touba and E. J. McCluskey. Logic synthesis of multilevel circuits with concurrent error detection. *IEEE Trans. Computer-aided Design*, 16(7):783–789, Jul. 1997.

[30] J. von Neumann. Probabilistic logics and the synthesis of reliable organisms from unreliable components. pages 43–98. Princeton University Press, 1956.