2010

Probabilistic error analysis models for nano-domain VLSI circuits

Karthikeyan Lingasubramanian
University of South Florida

Probabilistic Error Analysis Models for Nano-Domain VLSI Circuits

by

Karthikeyan Lingasubramanian

A dissertation submitted in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy in Electrical Engineering
Department of Electrical Engineering
College of Engineering
University of South Florida

Major Professor: Sanjukta Bhanja, Ph.D.
Nagarajan Ranganathan, Ph.D.
Syed M. Alam, Ph.D.
Wilfrido A. Moreno, Ph.D.
Paris H. Wiley, Ph.D.

Date of Approval:
March 3, 2010

Keywords: Reliability, Worst-case input, Sequential circuits, Redundancy models

DEDICATION

To my Family and Friends

ACKNOWLEDGEMENTS

I would like to thank my major professor Dr. Sanjukta Bhanja for believing in me and for giving me this opportunity. Without her support this dissertation wouldn't have been possible. She has trained me in every aspect of research and has helped me mold myself as a better researcher. Moreover she has also been a good friend to me.

My sincere thanks to Dr. Nagarajan Ranganathan, Dr. Syed M. Alam, Dr. Wilfredo A. Moreno and Dr. Paris H. Wiley for serving in my committee. I would like to thank them all for their valuable support and advice and most importantly for allotting their time in spite of their busy schedule.

I would like to thank all the faculties and staff of the Department of Electrical Engineering and College of Engineering.

I would like to thank all my present and former colleagues, Javier, Anitha, Srinath, Dinuka, Jose, Pruthvi, Thara, Shiva, Nirmal, Vivek, Satish, Praveen and Saket, for their unconditional support.

I am really very grateful for the invaluable support and motivation that I received from my family.

I would also like to thank all my friends that I ever had in my whole life.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

PROBABILISTIC ERROR ANALYSIS MODELS FOR NANO-DOMAIN VLSI CIRCUITS

Karthikeyan Lingasubramanian

ABSTRACT

Technology scaling to the nanometer levels has paved the way to realize multi-dimensional applications in a single product by increasing the density of the electronic devices on integrated chips. This has naturally attracted a wide variety of industries like medicine, communication, automobile, defense and even house-hold appliance, to use high speed multi-functional computing machines. Apart from the advantages of these nano-domain computing devices, their usage in safety-centric applications like implantable biomedical chips and automobile safety has immensely increased the need for comprehensive error analysis to enhance their reliability. Moreover, these nano-electronic devices have increased propensity to transient errors due to extremely small device dimensions and low switching energy. The nature of these transient errors is more probabilistic than deterministic, and so requires probabilistic models for estimation and analysis. In this dissertation, we present comprehensive analytic studies of error behavior in nano-level digital logic circuits using probabilistic reliability models. It comprises the design of exact probabilistic error models, to compute the *maximum* error over all possible input space in a circuit-specific manner; to study the behavior of transient errors in sequential circuits; and to achieve error mitigation through redundancy techniques. The model to compute maximum error, also provides the worst-case input vector, which has the highest probability to generate an erroneous output, for any given logic circuit. The model for sequential logic that can measure the expected output error probability, given a probabilis-

tic input space, can account for both spatial dependencies and temporal correlations across the logic, using a time evolving causal network. For comprehensive error reduction in logic circuits, temporal, spatial and hybrid redundancy models, are implemented. The temporal redundancy model uses the triple temporal redundancy technique that applies redundancy in the input space, spatial redundancy model uses the cascaded triple modular redundancy technique that applies redundancy in the intermediate signal space and the hybrid redundancy techniques encapsulates both temporal and spatial redundancy schemes. All the above studies are performed on standard benchmark circuits from ISCAS and MCNC suites and the subsequent experimental results are obtained. These results clearly encompasses the various aspects of error behavior in nano VLSI circuits and also shows the efficiency and versatility of the probabilistic error models.

CHAPTER 1

INTRODUCTION

Integrated Circuits are used in a wide range of important applications like automobile, aircraft, medicine, defense, communication and even house-hold appliances. Critical applications like medicine demand high accuracy and efficiency due to stringent safety requirements, while applications like automotive, defense demand more robustness due to extreme working conditions [41, 42, 39]. Also the demand for multi-dimensional applications in a single product has increased the density of the electronic devices on a chip eventually resulting in reduction of device feature size, pushing the technology to nanometer levels [60, 59]. Complementary Metal Oxide Semiconductor (CMOS) transistors, which are the current generation electronic devices, have been shrunk to sub-50nm dimensions [59]. This reduction in feature size results in variations in device and process parameters, which in turn leads to transient dynamic faults in digital circuits. In this dissertation, we present an error model that can handle these transient dynamic faults using probabilistic methods. Using this error model, we present a unique method to calculate maximum errors in digital circuits. Also, based on this error model, we present a time evolving probabilistic network that can calculate error in sequential circuits. Finally, we present temporal, spatial and hybrid redundancy techniques, which incorporates selective redundancy using the base error model, for error mitigation in digital circuits.

1.1 Motivation

Why use probabilistic models? Nano-domain computing devices are likely to have higher error rates (both in terms of defect and transient faults) as they operate near the thermal limit and information processing occurs at extremely small volume [61, 47]. Nano-CMOS, beyond 22nm, is not an exception in this regard as the frequency scales up and voltage and geometry scales down. The resulting errors, due to uncontrollable variations in device and process parameters like temperature and threshold voltage, are highly intractable for deterministic testing tools used to detect permanent faults. A fresh look at reliability in a technology independent fashion is both timely and necessary. Given the inherent stochastic nature of the devices in the nano-regime, instead of deterministic logic models probabilistic models would be more appropriate. This requires a significant shift in the design and testing paradigm, with reliability adopting a central role in design of electronic devices.

Why model maximum error? Industries like automotive and health care have traditionally addressed high reliability requirements by employing redundancy, error corrections, and choice of proper assembly and packaging technology. In addition, rigorous product testing at extended stress conditions filters out even an entire lot in the presence of a small number of failures [39]. Another rapidly growing class of electronic chips where reliability is very critical is implantable biomedical chips [41, 42]. More interestingly, some of the safety approaches, such as redundancy and complex packaging, are not readily applicable to implantable biomedical applications because of low voltage and low power operation and small form factor requirements. Authors in [41] identified that conventional approaches in device and parasitic modeling, circuit techniques, and manufacturing and test need to improve due to extreme low power and high reliability requirements, since these constraints pose serious complexities in circuit design through unpredictable design environment. In addition, we believe our method of calculating maximum probability of error and the proposed maximum

error probability aware design is well suited for implantable biomedical IC design. While two design implementation choices can have different average probabilities of failures, the lower average choice may in fact have higher maximum probability of failure leading to lower yield in manufacturing and more rejects during chip burn-in and extended screening. Also, when the input space for a circuit is completely random and equally probable, calculation of average error will suffice. But, as in some cases, when the input space gets biased, the average error information will not be comprehensive enough to understand the error behavior in the circuit. Therefore, using maximum probability of failure as a critical design metric along with average case would be required in design of safety critical electronic chips.

Why model error in sequential circuits? Most of the real-time applications of electronic devices, like random access memories, needs them to be sequential in nature. Sequential circuits consist of a combinational logic block, set of inputs, set of state bits where the values of the next state bit is fed back to the present state in the next clock cycle through latches. At a given time instance t_i, the state signals s_{t_i} are uniquely identified as a function of primary input signals i_{t_i} and state signals $s_{t_{i-1}}$ of the previous time instance giving rise to temporal correlations. Due to this, error occurring in the combinational part of the circuit at one time instance might propagate towards several consecutive time instances making the device more vulnerable [54, 51]. The static reliability models used for combinational circuits are not adequate to model the temporal dependencies between the circuit nodes, at the combinational part of the sequential circuit, at different time instances [52, 53, 54, 51, 55]. In order to handle this a more dynamic model which can evolve through consecutive time instances is needed.

1.2 Significance

The errors that can occur in nano-domain VLSI circuits can be widely divided into two categories, *hard faults* and *soft errors*. Hard faults refer to any permanent faults that can

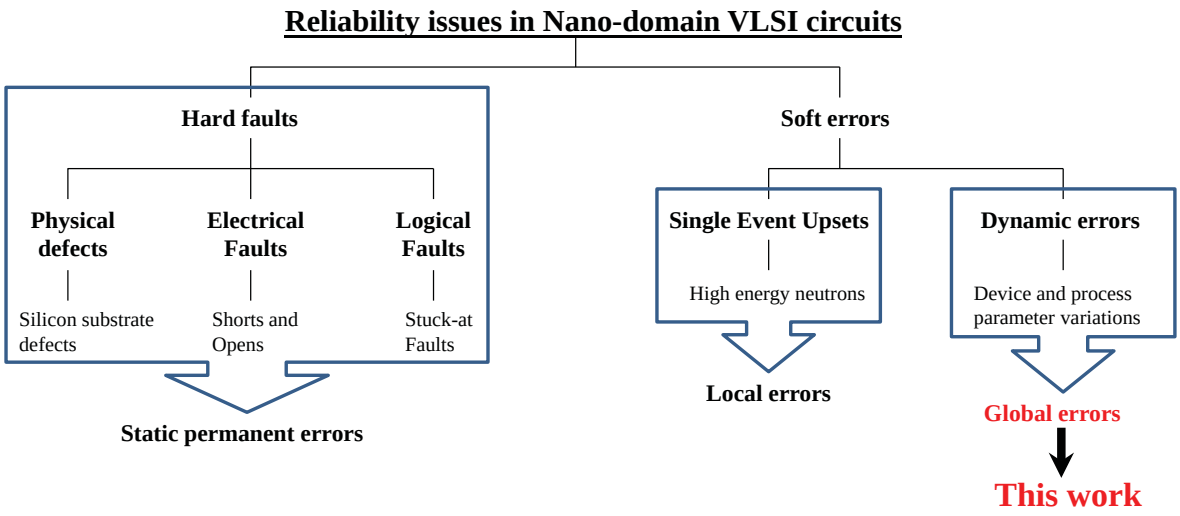


Figure 1.1. Significance of this dissertation

occur in a circuit component due to physical defects like oxide abnormalities in the transistor, electrical defects like shorts and opens, logical defects like stuck-at and delay faults. Soft errors refer to the failures in circuit components due to external conditions like high energy neutron interaction or device parameter variations. Out of these categories, soft errors are the toughest to model due to their transient nature, since the external conditions responsible for these errors are highly unpredictable at the nanometer levels. So, the reliability models used to address hard faults cannot be used to model soft errors, since they are completely deterministic. Therefore, comprehensive probabilistic models, like our model, are well suited to handle the transient soft errors.

The most prevalent soft errors in nano-domain VLSI circuits are widely categorized into Single Event Upsets (SEUs) due to external particle interaction, and dynamic errors due to device and process variabilities. While failures due to SEUs are more localized, in the sense, they occur in a particular component in the circuit and gets propagated, dynamic errors are more global, in the sense that, they can occur on multiple components of the circuit at the same time. So, the models that address failures due to SEUs are not enough to model dynamic

errors. Our model, presented in this work, targets the dynamic errors by giving provisions to address error behavior in multiple circuit components at the same time.

Also our model can be considered as a complete and comprehensive model that can accurately calculate *both maximum and average errors* in digital circuits. This versatility offers a wider diagnostic application space, which aids the collection of a variety of information sets that are highly essential for IC testing.

1.3 Contribution

The contributions of this dissertation are as follows,

- A method to calculate *maximum* output error in digital circuits using a probabilistic model is presented.
  - Given a circuit with a fixed gate error probability ε, this error model can provide the maximum output error probability and the worst-case input vector, which can be very useful testing parameters. Its also shown that these worst-case input vectors not only depend on the circuit structure but could dynamically change with ε.
  - It is shown that the maximum output error probabilities are much larger than average output error probabilities, for comparatively lower values of individual gate error probability ε, thereby signifying the importance of maximum error as a design parameter.
  - The circuit-specific error bounds for fault-tolerant computation are presented and it is shown that maximum output errors provide a tighter bound. Also, it is shown that the error bound for an individual gate placed in a circuit can be dependent on the circuit structure.

- Through this work, an efficient design framework that employs inference in binary join trees using *Shenoy-Shafer* algorithm, to perform MAP hypothesis accurately, is being applied for the first time in the context of digital computing machines.
- The validity of the error model is tested through comparison with circuit simulations using HSpice and the results showed that the highest % difference of the error model over HSpice is just 1.23%, signifying its accuracy.
- The possibility of efficient error incorporation in this model is presented by providing variable ε values to different gates of a circuit, instead of providing the same ε value to all gates. This formation of the error model can help in useful diagnostic studies like error sensitivity analysis.

- An exact probabilistic error model that can study transient error behavior in *sequential* logic is presented.
  - This model can accurately calculate the average output error probability in any given sequential circuit.
  - A *minimal* time evolving probabilistic network, namely, the Temporal Dependency Model (TDM), that can handle both spatial dependencies between nodes in a single time slice and temporal dependencies between nodes in different time slices, is presented.
  - It is shown that the increase in output error probabilities is more than 2 folds, even for a slight increase in ε value, thereby indicating the vulnerability of sequential circuits to transient errors.
  - The crucial study of error propagation across different time instances, in a sequential circuit, can be performed using this model. This study is important to understand error behavior in sequential circuits.

- It is shown that the number of time slices needed by the model, to converge to a final average output error value, is completely dependent on the circuit structure.
- The flexibility of the error model is shown by incorporating unequal gate error probability values, ε_0 and ε_1, to study the effect of $0 \rightarrow 1$ and $1 \rightarrow 0$ errors on the output of a circuit. Given a gate output signal, ε_0 represents the probability of error occurrence when the ideal value of the signal is '0', and ε_1 represents the probability of error occurrence when the ideal value of the signal is '1'.
- The validity of the error model is tested through comparison with circuit simulations using HSpice and the results showed that the highest percentage difference of the error model over HSpice is only 6.25%, signifying its accuracy.

- Using the probabilistic error model, temporal, spatial and hybrid redundancy techniques are performed, to achieve error mitigation in digital logic circuits.
  - Efficient error reduction is achieved through *selective* redundancy, which is established by applying redundancy only to the most influential input combinations and the most sensitive nodes.
  - Through experimental results, the relative benefits of the temporal, spatial and hybrid redundancy schemes are presented and hybrid redundancy is shown to be the best scheme for error mitigation in digital logic circuits.
  - It is shown that increasing the amount of redundancy results in better error mitigation in all the three schemes.
  - It is shown that the error mitigation percentage for 15% temporal redundancy, is more than 10% for all circuits, while for 15% spatial redundancy, it is more than 20% for all circuits and for 15% hybrid redundancy, it is more than 30% for all circuits, thereby showing the high yield of hybrid redundancy scheme.

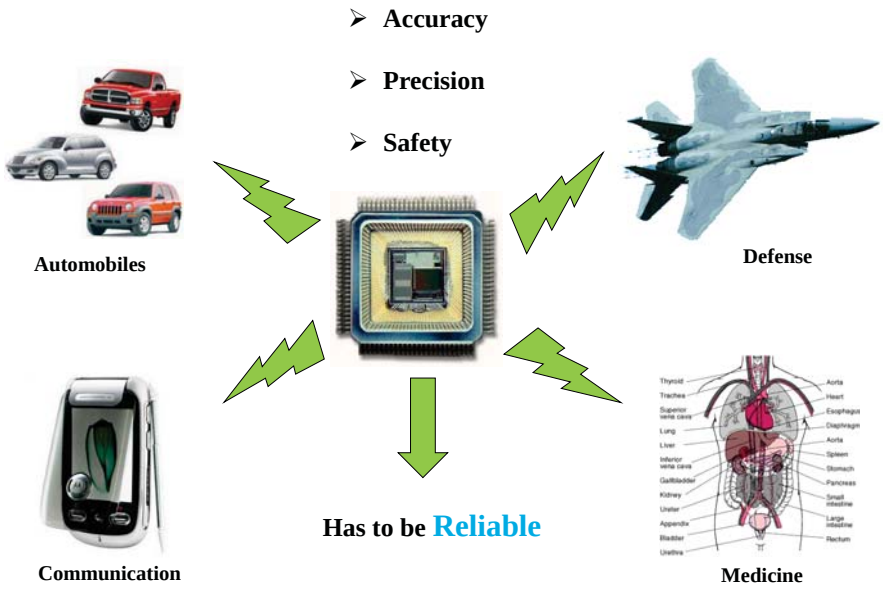


Figure 1.2. Scope of application

– Delay and area penalties in temporal and spatial redundancies respectively are presented and its is shown that the area penalty is much higher than the delay penalty.

1.4 Scope of Application

Digital VLSI circuits are widely used in critical and essential applications like automobiles, defense, medicine and communication. The need for reliable computation in these circuits are of the utmost importance due to the nature of its applications. VLSI circuits are used in the automobile brake system, implantable bio-medical devices like pacemaker, aircraft control system and multi-functional smart phones like iPhone. The scaled computational devices in current generation nano-domain VLSI circuits has immensely improved its application space. The advent of smart phones and implantable bio-medical chips are made possible primarily by this scaling trend. At the same time VLSI circuits at nano-domain suffer from various reliability issues that should be addressed during the design process.

In nano-domain digital VLSI circuits, affected by multiple errors, there can be one maximum error that can even breakdown the entire device. While primarily all of the current reliability studies estimate the overall adverse effect by considering the average of all errors, estimating the worst-case maximum error has proved to be tedious and cumbersome. Our reliability model, presented in this dissertation, can efficiently estimate this worst-case maximum error and the input vector associated with this error, through intelligent diagnostic studies.

In IC testing, the usage of a probabilistic error model and the information about the worst-case input vector can help to improve testing techniques like scan chains, burn-in test and hierarchical testing. Scan chains are widely used in Design for Test (DFT) methodologies for IC testing. The basic idea is to form a chain of flip-flops that are made scan-able and the desired test pattern can be serially inserted into the flip-flop chain. The test pattern is applied to the logic circuits driven by the flip-flop chain after which the logic circuit outputs can also be captured into the same or different flip-flop chain for serial shift-out. In such a setup, including the worst-case input vector in the test patterns can speed up the testing process, since the most hazardous behavior of the circuit-under-test can be detected with the worst-case input vector. Burn-in tests are performed to find out devices with inherent defects or manufacturing defects [44]. These devices will go faulty when subjected to high stress. The IC is subjected to long test time and stress conditions, such as extreme Vdd and temperatures, during a burn-in test. To aid the burn-in test, a probabilistic error model that can target and exercise individual device fault modes would help to expedite the failure mechanisms and to screen for inherent faults in a shorter test time. More specifically, the worst case input vectors generated according to our method is well suited for application during the burn-in test. Finally, in hierarchical testing, the entire circuit-under-test is divided into several internal modules where these modules can be tested individually. Such a hierarchical division reduces the size of circuit-under-test facilitating rigorous probabilistic error analysis and the application of worst input vectors to the targeted internal modules.

The reliability model for error estimation in sequential circuits, presented in this dissertation, can be used to perform efficient diagnostic studies in essential real-time applications like computer memories. In these sequential circuits, for a fixed input vector, the intermediate signals can get stuck at a wrong value due to the presence of error. This could propagate across several time instances and this behavior can happen to any input vector. Such deterministic approaches provides inaccurate estimation of the error behavior in sequential circuits. Our model, which is a probabilistic reliability model, takes care of this discrepancy by treating both input and signal space in a probabilistic manner, thereby ensuring efficient diagnostic studies for reliability.

While the three error mitigation schemes, temporal, spatial and hybrid, presented in this dissertation, can be used for error optimization in any nano-domain VLSI circuit, the trade-off studies between them can provide essential application-specific information for circuit designers. If the application demands lesser area, then more importance should be given to temporal redundancy than spatial redundancy. If the application has high probability of error occurrence in the signal space than the input space, then more importance should be given to spatial redundancy than temporal redundancy. The trade-off studies presented in this dissertation, can provide information related to the above scenarios, which are crucial for circuit design.

1.5 Organization

This dissertation is organized as follows,

- Chapter 2 provides the related research works done in the field of probabilistic reliability analysis for VLSI circuits.
- Chapter 3 provides the fundamental design concepts of the probabilistic error model.

- Chapter 4 explains, in detail, about the modeling of maximum errors in logic circuits using a probabilistic error model.
- Chapter 5 explains, in detail, about modeling of errors in sequential circuits using a dynamic time-evolving probabilistic error model.
- Chapter 6 explains, in detail, about the temporal, spatial and hybrid redundancy schemes used for error mitigation in digital logic circuits.
- Chapter 7 provides the conclusion and future directions of this work.

CHAPTER 2

RELATED WORK

In nanometer level circuits, due to device scaling, the most prevalent and detrimental errors are soft errors that are caused mainly by external particle interactions and variations in device and process parameters. While the former results in localized failures like Single Event Upsets (SEUs), the latter leads to more global dynamic errors.

2.1 SEU Modeling

The modeling of device failures due to SEUs are done in different levels of design abstraction, like device level, circuit level and gate level [62, 69, 70, 71, 73, 85, 74]. Initial work on external radiation interaction on semiconductors was done as early as 1967 [62], in which the authors proposed one dimensional drift diffusion models to study the radiation effects on semiconductor devices used widely in space applications. This work was followed by a number of significant device level models for memory elements, using numerical simulation [63, 64]. In order to handle more complex situations, which are intractable by numerical simulation models, analytic and empirical models were proposed [67]. The study of external particle interaction with semiconductor devices, which is more of a multi dimensional phenomenon, was enhanced through the advent of two dimensional and three dimensional models [65, 66], which accurately measured the charge particle drift and diffusion mechanisms. At the circuit level, SEU modeling is done by addressing circuit parameters like supply voltage, threshold voltage and clock period; and circuit characteristics like electrical masking, logical mask-

ing and latching window effects. Simulation based models like SEMM [69] and SERA [56] encapsulates these circuit aspects to provide soft error rate analysis in digital logic circuits. While optimization techniques using dual-Vdd and gate sizing are used to model SEU [70], its effects on interconnects are also modeled at the placement level [71], using simulated annealing. At the gate level, SEU modeling is based primarily on the detection of the probability of error occurrence at the gate outputs. Logical abstraction tools like binary decision diagrams are used to perform soft error rate analysis in both combinational [72] and sequential [51] circuits, while a completely probabilistic model based on Bayesian networks was used in [85] to detect SEUs in digital logic circuits. While practical experiments like injecting SEUs in chips using laser pulses to verify fault tolerance [74, 75] were performed, popular testing techniques like built-in self-test mechanism [76] were also used to study soft errors.

2.2 Dynamic Error Modeling

Dynamic errors are transient soft errors caused by the uncontrollable and unpredictable fluctuations in device and process parameters due to scaling. These global errors can coexist with the local SEUs and static hard faults, and they can happen randomly at any node in the circuit, making them untraceable. The basic concept of dynamic error modeling is the assumption that every circuit component will have a finite propensity to be erroneous. Based on this idea, researchers approached dynamic error modeling problem in three broad categories, calculation of error bounds, calculation of average error, and error reduction through redundancy.

2.2.1 Calculation of Error Bounds

The study of reliable computation using unreliable components was initiated by Von Neumann [1] who showed that erroneous components with some small error probability can pro-

vide reliable outputs and this is possible only when the error probability of each component is less than $1/6$. In this heuristic study, Neumann represented the logic gates as automatons which are governed by logic functions. It was stated that the probability of error in the automaton and its output cannot exceed $1/2$, since the system will become irrelevant at that bound. Keeping this as the basic upper bound for the probability of error in the output, the error probability of the automaton was studied through a majority organ, in which three copies of the same automaton were created and the majority of the three outputs was considered true. This arrangement was proven to reduce the error probability of the base system, and through this it was shown that the error probability of the automaton cannot be $\geq 1/6$, since at this upper bound the system becomes unsustainable.

This work was later enhanced by Pippenger [3] who realized Von Neumann's model using formulas for boolean functions. Here the digital logic components are realized using functions whose number of arguments relate to the number of inputs in the component. Through this arrangement, it was shown that for a function controlled by k-arguments, the error probability of each component should be less than $(k-1)/2k$ to achieve reliable computation. Through this, an interesting result was shown for 3-input components, whose error probability bound for reliable computation was $1/3$, which is greater than the Von Neumann bound of $1/6$, thereby creating curiosity. This work was later extended by using networks instead of formulas to realize the reliability model [4]. In [5], Hajek and Weller used the concept of formulas to show that for 3-input gates the error probability should be less than $1/6$, thereby reiterating Von Neumann's bound. Later this work was extended for k-input gates [6] where k was chosen to be odd. The authors claimed that since $k+1$ input gates can simulate k input gates, their model can be easily used to compute bounds for gates with even number of inputs. For a specific even case, Evans and Pippenger [7] showed that the maximum tolerable noise level for 2-input NAND gate should be less than $(3-\sqrt{7})/4 = 0.08856\cdots$.

Later this result was reiterated by Gao et al. [8] for 2-input NAND gate, along with other results for *k*-input NAND gate and majority gate, using bifurcation analysis that involves repeated iterations on a function relating to the specific computational component. The probability of the output line of a NAND gate, given by Z, was associated with the probabilities of the input lines X and Y using the equation,

$$Z = (1-\varepsilon)(1-XY) + \varepsilon XY = (1-\varepsilon)(2\varepsilon-1)XY \tag{2.1}$$

where ε is the probability of error in the NAND gate. In order to study the error behavior, a network of NAND gates, where the output of each gate is connected to the input of at least one other gate, was created and the inputs X and Y are considered to be equally probable to be at logic '1'. The corresponding equation for this network was written as,

$$X_{i+1} = (1-\varepsilon) + (2\varepsilon-1)X_i^2 \tag{2.2}$$

The initial value X_0 was arbitrarily chosen and an iterative process was performed to obtain consequent X_i values. After the solution has converged, values from the last few iterations are plotted against the corresponding ε values to obtain the bi-modal graph for bifurcation analysis. This bi-modal graph clearly showed that reliable computing using erroneous 2-input NAND gates is not possible when its error probability $\varepsilon = 0.08856\cdots$.

While there exist studies of circuit-specific bounds for circuit characteristics like switching activity [9], the study of circuit-specific error bounds would be highly informative and useful for designing high-end computing machines.

2.2.2 Calculation of Average Error

Many researchers are currently focusing on computing the average error from a circuit and also on the expected error to conduct reliability-redundancy trade-off studies. In [45], a Probabilistic Transfer Matrix (PTM) based model for reliability studies was proposed. In this method each circuit signal is represented using random variables and the functionality of each erroneous gate is represented in a matrix form using the PTMs (Fig. 2.1.(a)). Each gate in the underlying digital circuit was represented by an individual PTM. To calculate the error probability of the circuit, a PTM for the entire circuit is formed by multiplying the individual gate PTMs. If gates g_1 and g_2 are connected in series, under the condition that when g_1 gets an input g_1^I it results in g_2 giving an output g_2^O, the combined PTM can be written as

$$p(g_2^O|g_1^I) = \sum_{all\, j} p(g_2^O|j)p(j|g_1^I) \tag{2.3}$$

If gates g_1 and g_2 are connected in parallel, under the condition that when g_1 gets an input g_1^I it results in output g_1^O and when g_2 gets an input g_2^I it results in output g_2^O, the combined PTM can be written as

$$p(g_2^O|g_1^I) = p(g_2^O|g_2^I)p(g_1^O|g_1^I) \tag{2.4}$$

This is an exact method but it is computationally expensive.

An approximate method based on Probabilistic Gate Model (PGM) is discussed by Han et al. in [15]. Here the PGMs are formed using the sum of product equations governing the functionality between an input and an output. For any gate, with an output Z_i and with error probability ε, its PGM can be written as,

$$Z_i = E_i(1-\varepsilon) + (1-E_i)\varepsilon \tag{2.5}$$

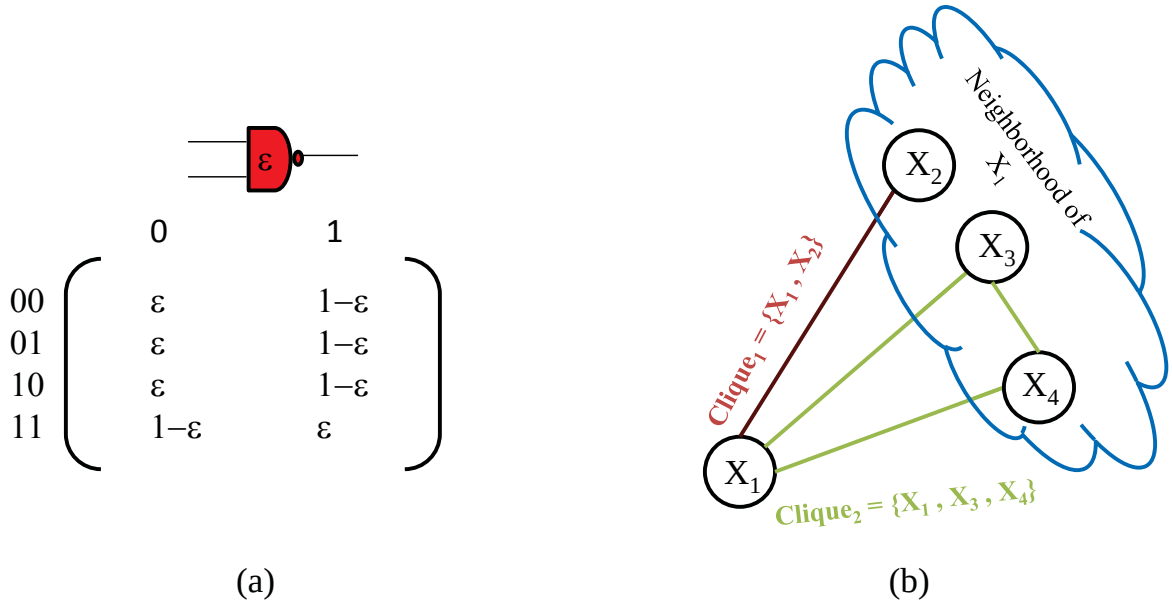


Figure 2.1. (a) Probabilistic transfer matrix for erroneous NAND gate with error probability ε [45] (b) Markov random field [47]

where E_i is the sum of product equation. For a 2-input AND gate with inputs I_1 and I_{12}, $E_i = I_1 I_2$. So the corresponding Z_i can be written as,

$$Z_i = (I_1 I_2)(1-\varepsilon) + (1-(I_1 I_2))\varepsilon \tag{2.6}$$

All the gates in the circuit were represented with individual PGMs and the overall reliability of the circuit was calculated by multiplying the individual gate reliabilities, which were assumed to be independent. This approximate model was proved to be faster than the exact PTM model.

A Markov Random Field (MRF) based probabilistic model for reliability studies was proposed in [47], which concentrated more on hard errors than soft errors. Here, the circuit signals were represented as random variables in a Markov random network, where every node is dependent only with the directly connected nodes that are called its neighbors (Fig. 2.1.(b)). Given a set of random variables $\Gamma = \{X_1, \cdots, X_n\}$ forming a Markov network, the probability

of any random variable, X_i, in the Markov network was described using Gibbs distribution as follows,

$$P(X_i|\{\Gamma - X_i\}) = \frac{1}{Z} e^{-\frac{1}{kT}\sum_{c\in\phi} U_c(X)} \tag{2.7}$$

where Z is a normalizing constant that bounds the probability value to [0,1], kT is the thermal energy, c is clique in the set of cliques ϕ associated with X_i and U_c is the clique energy. A typical clique in the circuit representation of Markov network will comprise of the nodes representing the inputs and output of a gate. In this sense, every gate will have its own clique and clique energy. The logic gates were represented using their sum of products term and the clique energy for each gate was derived. For an inverter with input x_0 and output x_1, the clique energy was derived as follows,

$$\begin{aligned} U &= -((1-x_0)x_1 + x_0(1-x_1)) \\ &= -(x_1 - x_0x_1 + x_0 - x_0x_1) \\ &= 2x_0x_1 - x_0 - x_1 \end{aligned} \tag{2.8}$$

The negative sign in the clique energy signified the design condition that clique energies of valid states should be lower than those of invalid states. The corresponding Gibbs distribution was given as,

$$P(x_0,x_1) = \frac{1}{Z} e^{-\frac{1}{kT}(2x_0x_1 - x_0 - x_1)} \tag{2.9}$$

The probability of output $x_1 = 1$ was calculated by marginalizing $P(x_0,x_1)$ over all possible values of x_0.

$$\begin{aligned} P(x_1) &= \frac{1}{Z} \sum_{x_0=\{0,1\}} e^{-\frac{1}{kT}(2x_0x_1 - x_0 - x_1)} \\ &= \frac{e^{\frac{x_1}{kT}} + e^{\frac{(1-x_1)}{kT}}}{2(1+e^{\frac{1}{kT}})} \end{aligned} \tag{2.10}$$

Likewise, the probability distribution of every signal in the circuit was represented using Gibbs distribution. Corresponding probability distributions for the primary outputs of the circuit was determined by propagating the marginalized distributions across various cliques using belief propagation algorithm. Since these distributions were associated with thermal energy kT, comprehensive reliability studies on nanoarchitectures working under critical thermal limits, were performed by altering the kT values and examining the signal probability distributions. Although, this work provided some much needed insight on thermal behavior of nano-domain circuits, it was performed on error free devices instead of erroneous ones.

Another work on reliability studies using probabilistic model checking was proposed in [58]. This method employed discrete-time Markov Chains for probabilistic model checking. In another significant work [99], the average output error in digital circuits was calculated using a probabilistic reliability model that employed Bayesian Networks.

2.2.3 Error Reduction Through Redundancy

The term 'redundancy' means the usage of multiple redundant copies of the same erroneous component in order to test or improve its reliability. Von Neumann, in his legendary work, was one of the first to propose one such methodology called *multiplexing* and he used it to study the reliability of NAND logic [1]. This model was created by taking multiple copies of the same erroneous NAND gate and supplying them input signals randomly from various bundles of input lines. This setup ensures effective duplication of all possible signals at the outputs. To obtain better error tolerance, two more NAND multiplexing setups are cascaded with the previous one. While the first NAND multiplexing setup called the "Executive Unit" performed the logic computation, the following two units called the "Restorative Unit" restored the correct computation values. (Fig. 2.2.)

Von Neumann also introduced the widely used redundancy technique called Triple Modular Redundancy (TMR) [1]. In TMR, three copies of the same erroneous logic component

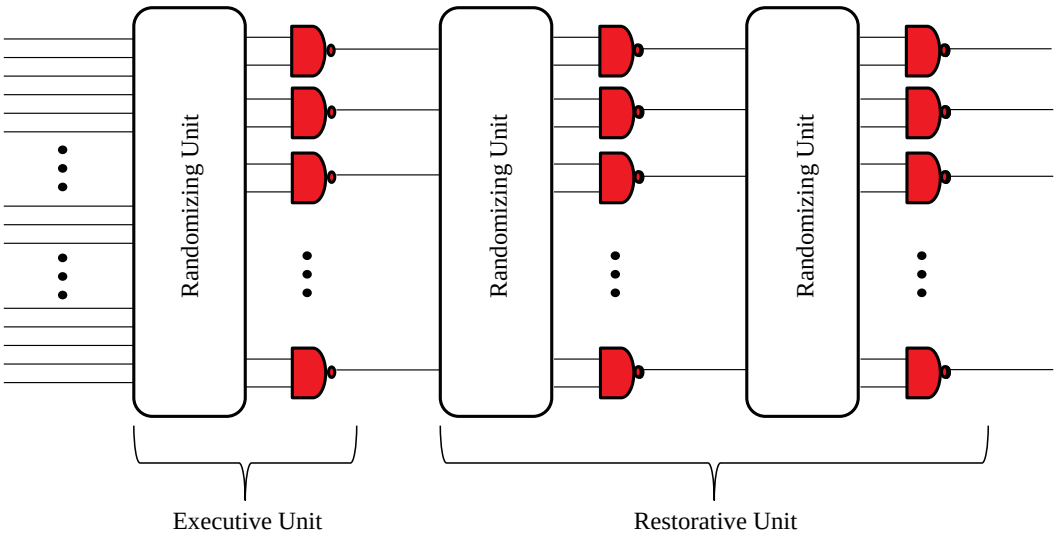


Figure 2.2. NAND multiplexing scheme introduced by Von Neumann [1]

was created and the correct value was determined by performing the majority voting out of the three outputs. Given three different signals X, Y and Z, the majority voting could be performed using the function, $XY + YZ + XZ$. Using this, Von Neumann showed significant reduction in the probability of error occurrence in logic devices. As an extension of TMR, a more general model called N-Modular Redundancy (NMR) [2] was proposed, where N is chosen to be odd to facilitate majority voting. If TMR was used to choose the majority of 2 out of 3 inputs, NMR was used to choose the majority of $n+1$ out of $2n+1$ inputs. Also, given an erroneous system with error probability ε, the reliability R through performing TMR was given by,

$$R(TMR) = \varepsilon^3 + 3\varepsilon^2(1-\varepsilon) \tag{2.11}$$

and the corresponding reliability through performing NMR was given by,

$$R(NMR) = \sum_{i=0}^{n} \frac{N!}{(N-i)!i!}(1-\varepsilon)^i \varepsilon^{N-i} \tag{2.12}$$

where $N = 2n+1$. These base models for hardware redundancy were later applied in essential applications like fault-tolerant microprocessor design [10], and also paved the way to a variety of techniques for software, data and time redundancies. Apart from being used in the circuit level, they were also used in different levels of design abstractions like in [73], where Selective TMR (STMR) was used in FPGA's to minimize error behavior due to SEUs.

From the initial works of Von Neumann, the study of fault-tolerant computation expanded its barriers into fields like nano-computing architectures. An expansion of the TMR technique called Cascaded Triple Modular Redundancy (CTMR) [11] was used for reliability studies of nanochips using single-electron devices and quantum cellular automata gates. While TMR is referred to as single level redundancy technique, CTMR is referred to as multilevel redundancy technique, where outputs from three different TMR units were supplied to another majority gate to perform multiple levels of voting in order to obtain better error reduction. A generalized CTMR technique, called Cascaded General Modular Redundancy (CGMR) was also proposed in this work [11].

In [12], the reliability of reconfigurable architectures was obtained using NAND multiplexing technique. The processors in the architecture were implemented with NAND multiplexing system with a redundancy factor of 3. In the design, redundant spare circuitries were also developed to enhance error correction and minimize error detection. In [13], majority multiplexing was used to achieve fault-tolerant designs for nanoarchitectures. They further enhanced the majority multiplexing model for small input error probabilities, by removing the restorative stage, since effective restoration is possible without that stage. A recent comparative study of some of these methods [14], indicates that a 1000-fold redundancy would be required for a device error (or failure) rate of 0.01[1].

[1]Note that this does *not* mean 1 out of 100 devices will fail, it indicates the devices will generate erroneous output 1 out of 100 times.

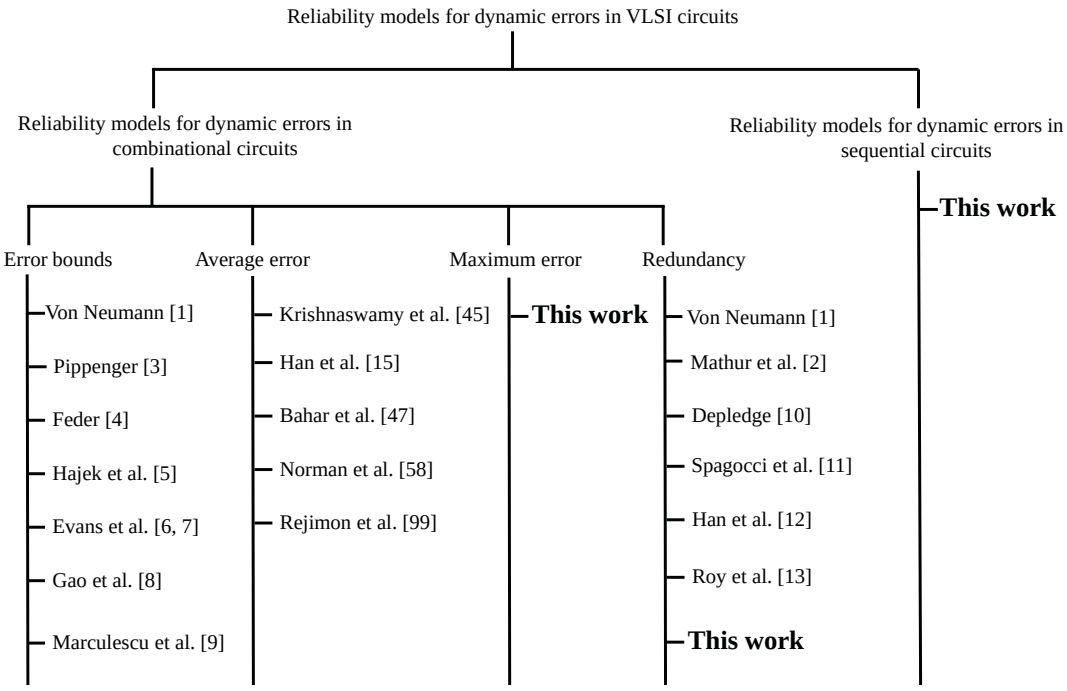


Figure 2.3. Some of the related works on reliability models for dynamic errors in VLSI circuits

2.3 Relation to State-of-the-Art

This work concentrates on the following,

- Modeling *dynamic errors*, which are global, as opposed to localized SEUs. This is done using a probabilistic error model, where efficient error incorporation in multiple nodes is possible. Also in this model, the error injection and probability of error for each gate can be modified easily. Moreover, both fixed and variable gate errors can be accommodated in a single circuit without affecting computational complexity.

- Estimation of *maximum error as opposed to average error*, since for higher design levels it is important to account for maximum error behavior, especially if this behavior is far worse than the average case behavior. This estimation is performed as a diagnostic study in our error model, using the Maximum *a posteriori* (MAP) hypothesis, where the

output nodes are forced to be erroneous and the information is propagated towards the input nodes to estimate the possible input configuration, that can provide a maximum error in the output.

- Estimation of output error in *sequential circuits as opposed to combinational circuits*, since the transient errors that occurs in a particular time frame, of a sequential circuit, will propagate to consecutive time frames thereby making the device more vulnerable. This estimation is performed using a *minimal* time evolving probabilistic network, namely, the Temporal Dependency Model (TDM), that can handle both spatial dependencies between nodes in a single time slice and temporal dependencies between nodes in different time slices.

- Designing *temporal, spatial and hybrid redundancy schemes*, using our probabilistic error model, to achieve error mitigation. We perform temporal redundancy using Triple Temporal Redundancy (TTR) technique and spatial redundancy using CTMR technique. Also efficient error reduction is achieved through *selective* redundancy, by applying redundancy only to the most influential input combinations and the most sensitive nodes.

CHAPTER 3

DESIGN FUNDAMENTALS

3.1 Probabilistic Representation of Digital Circuits

A digital circuit is basically a network of digital signals connected together through gates whose functionalities are based on boolean logic. This network can be represented accurately using a graphical model, where the nodes represent the digital signals and the edges represent the boolean logic functionality of the gates. Also these edges should be unidirectional, since information flow in digital circuits is unidirectional from input to output. In order to assist efficient diagnostic studies on digital circuits, their graphical representation can be modeled as probabilistic graphical models where each node is a random variable with two possible states, 'logic 0' and 'logic 1' or simply '0' and '1'. To represent the digital functionalities, each random variable should be associated with a probability distribution function (pdf). Consider the example in Fig 3.1., where a digital circuit and its probabilistic graphical model are given. As discussed, each node from $N1$ to $N8$ is a random variable whose value will be either '0' or '1'. In a network representing any digital circuit, the nodes corresponding to the primary inputs (i.e.,$N1$, $N2$, $N3$ in our example) will always be completely independent and every other *child* node will be dependent on at least one *parent* node. This kind of interdependency between nodes gives rise to conditional probability distribution, and so the pdf's are represented as Conditional Probabilistic Tables (CPTs). Fig 3.1. provides the CPTs for all the nodes. Since $N1$, $N2$, $N3$ are primary inputs, their pdf's can be controlled by the user. The child node $N4$ is dependent on its parent nodes $N1$ and $N2$ through AND logic, and the corresponding CPT

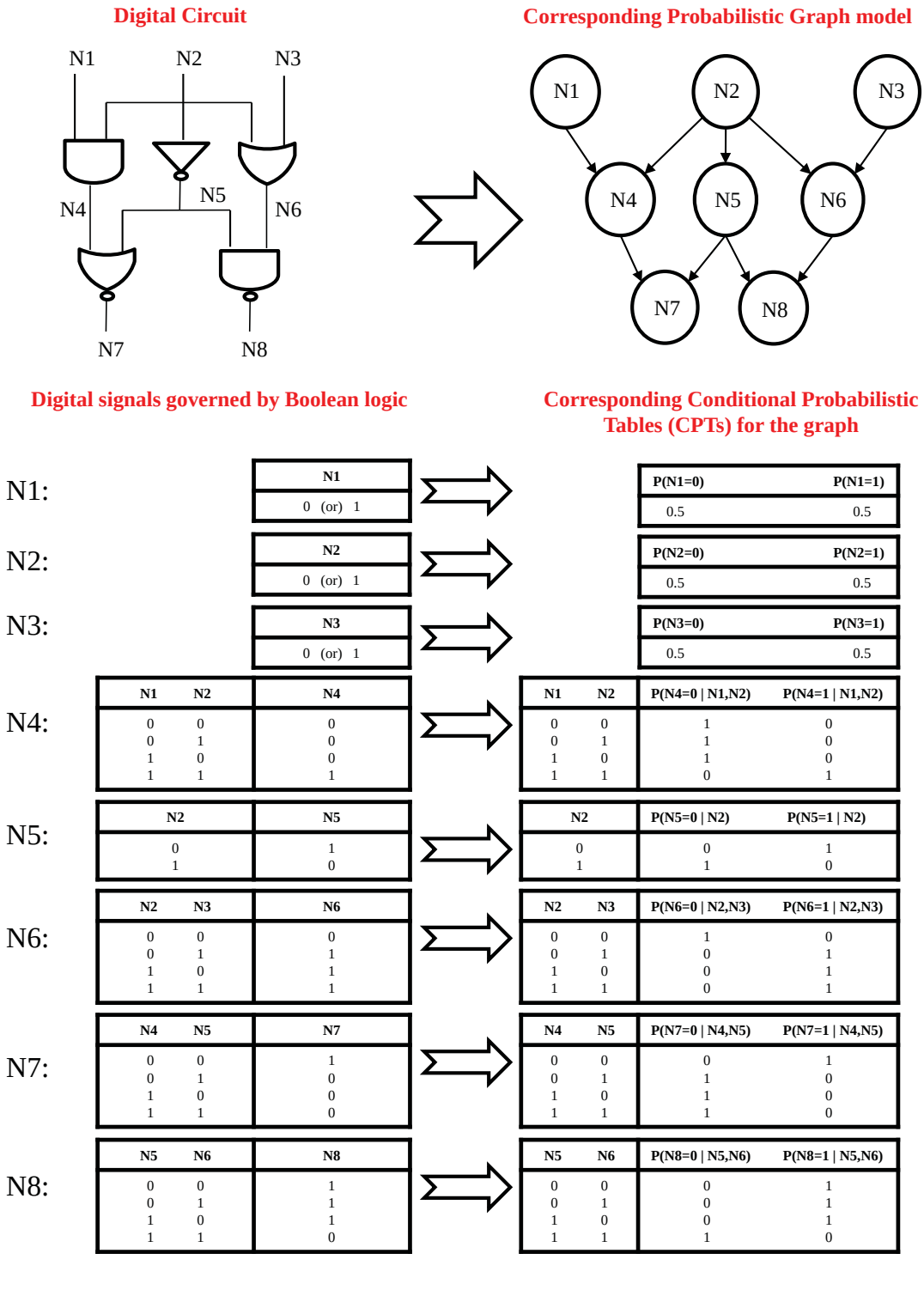


Figure 3.1. Representation of a digital circuit as a probabilistic graph

should reflect this functionality. This can be achieved by providing the pdf as follows,

$$P(N4 = 0|N1, N2) = \begin{cases} 0 & \text{if } N1\text{=1 and } N2\text{=1} \\ 1 & \text{otherwise} \end{cases}$$
$$P(N4 = 1|N1, N2) = \begin{cases} 1 & \text{if } N1\text{=1 and } N2\text{=1} \\ 0 & \text{otherwise} \end{cases} \tag{3.1}$$

Similarly the CPTs for $N5$ should obey NOT logic, $N6$ should obey OR logic, $N7$ should obey NOR logic, and $N8$ should obey NAND logic.

Once the graph model for a digital circuit is ready, the next obvious question is whether the model captures all the interdependencies between the nodes. For example, in the probabilistic graph given in Fig 3.1., the node $N7$ is directly dependent on nodes $N4$, $N5$ and indirectly dependent on nodes $N1$, $N2$. Also, node $N8$ is directly dependent on nodes $N5$, $N6$ and indirectly dependent on nodes $N2$, $N3$. If we add edges representing these indirect dependencies, then the resulting probabilistic graph will be as seen in Fig 3.2.(a). But are these edges necessary? In the given digital circuit, it can be seen that the relation of the signal $N7$ towards the signals $N1$, $N2$ is taken care by the signals $N4$, $N5$, i.e. any change in signals $N1$, $N2$ will be captured by their direct output signals $N4$, $N5$ and the same changes will be translated to signal $N7$ through $N4$ and $N5$. So, in the corresponding probabilistic graph model, we can comfortably say that node $N7$ is independent of nodes $N1$, $N2$ given nodes $N4$, $N5$. In a similar fashion, we can also say that node $N8$ is independent of nodes $N2$, $N3$ given nodes $N5$, $N6$. In other words, we can say that all the indirect dependencies are taken care by the direct dependencies. As a result all the extra edges representing indirect dependencies can be removed from the probabilistic graph model given in Fig 3.2.(a) resulting in Fig 3.2.(b), which is similar to the initial model given in Fig 3.1. This representation is the absolute *minimal*, in the sense that

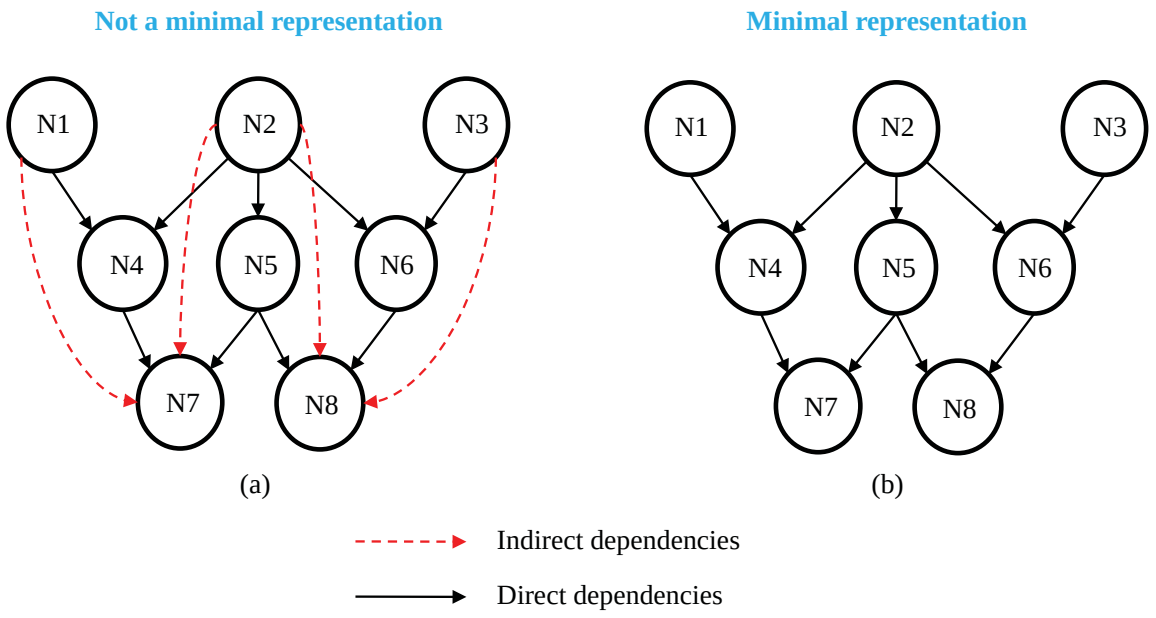


Figure 3.2. Minimal representation of the probabilistic graph

removing even one edge will collapse the interdependencies between the nodes and eventually results in an incomplete representation of the given digital circuit.

The probabilistic graph model can be represented mathematically as the conditional factoring of a joint probability distribution. Any probability function $P(y_1, y_2, \cdots, y_N)$ can be written as,

$$\begin{aligned} P(y_1, \cdots, y_N) &= P(y_N | y_{N-1}, y_{N-2}, \cdots, y_1) \\ & \quad P(y_{N-1} | y_{N-2}, y_{N-3}, \cdots, y_1) \\ & \quad \cdots P(y_1) \end{aligned} \tag{3.2}$$

where $y_1, y_2, \cdots, y_N$ are random variables. This expression holds for any ordering of these random variables. For the example probabilistic graph model in Fig 3.1., this probability

function can be written as,

$$\begin{aligned} P(n1,\cdots,n8) &= P(n8|n7,n6,n5,n4,n3,n2,n1) \\ & \quad P(n7|n6,n5,n4,n3,n2,n1) \\ & \quad P(n6|n5,n4,n3,n2,n1) \\ & \quad P(n5|n4,n3,n2,n1) \\ & \quad P(n4|n3,n2,n1) \\ & \quad P(n3)P(n2)P(n1) \end{aligned} \tag{3.3}$$

where $n1,\cdots,n8$ are the random variables represented by the nodes $N1,\cdots,N8$ respectively. But this equation does not perfectly represent the structure of the corresponding probabilistic graph model. As discussed earlier, in the minimal representation of the probabilistic graph model, every child node is connected only to its parent nodes. So Eqn. 3.2 can be restructured as follows,

$$P(y_1,\cdots,y_N) = \prod_v P(y_v|Pa(Y_v)) \tag{3.4}$$

where $Pa(Y_v)$ are the parents of the node Y_v, representing its direct causes. For the example probabilistic graph model in Fig 3.1., this restructured joint probability function can be written as,

$$\begin{aligned} P(n1,\cdots,n8) &= P(n8|n6,n5)P(n7|n5,n4) \\ & \quad P(n6|n3,n2)P(n5|n2) \\ & \quad P(n4|n2,n1)P(n3)P(n2)P(n1) \end{aligned} \tag{3.5}$$

3.2 Modeling Error in Digital Circuits

Any unexpected change in the logic state of the digital signals gives rise to error in digital circuits. In order to understand and study these errors, we need a model that can detect these unexpected changes. One such way of doing that is to compare the erroneous circuit with its ideal error-free counterpart. Consider the circuit in Fig 3.3.(a), where each signal other than the primary input signals can be erroneous through the faulty gates. Note that we assume that primary input signals are error-free. In order to create the error detection model, two copies of the circuit is created, where one copy represents the circuit in its normal erroneous form and the other copy represents the circuit in its ideal form. When the primary outputs of these two copies are compared, any error occurrence will become evident through the possible presence of dissimilar logic states. The appropriate logic gate to do this operation is the XOR gate, which produces a '1' in its output when its inputs have dissimilar logic states and provides a '0' in its output when its inputs have similar logic states. Fig 3.3.(b) illustrates the error detection model for digital circuits based on the above mentioned concept. $N4^e, N5^e, \cdots, N8^e$ are the erroneous signals and $N4, N5, \cdots, N8$ are the ideal signals. Signal $C1$ gives the comparison between the erroneous and ideal primary outputs $N7$ and $N7^e$; signal $C2$ gives the comparison between the erroneous and ideal primary outputs $N8$ and $N8^e$. It should be noted that the ideal error-free portion and the comparator portion are fictitious and used only for studying the given circuit.

The corresponding probabilistic graph model for error detection can be created as shown in Fig 3.3.(d). Lets say that each gate in the digital circuit has ε % chance of being faulty. ε can be termed as the *gate error probability*. This can be accommodated in the corresponding

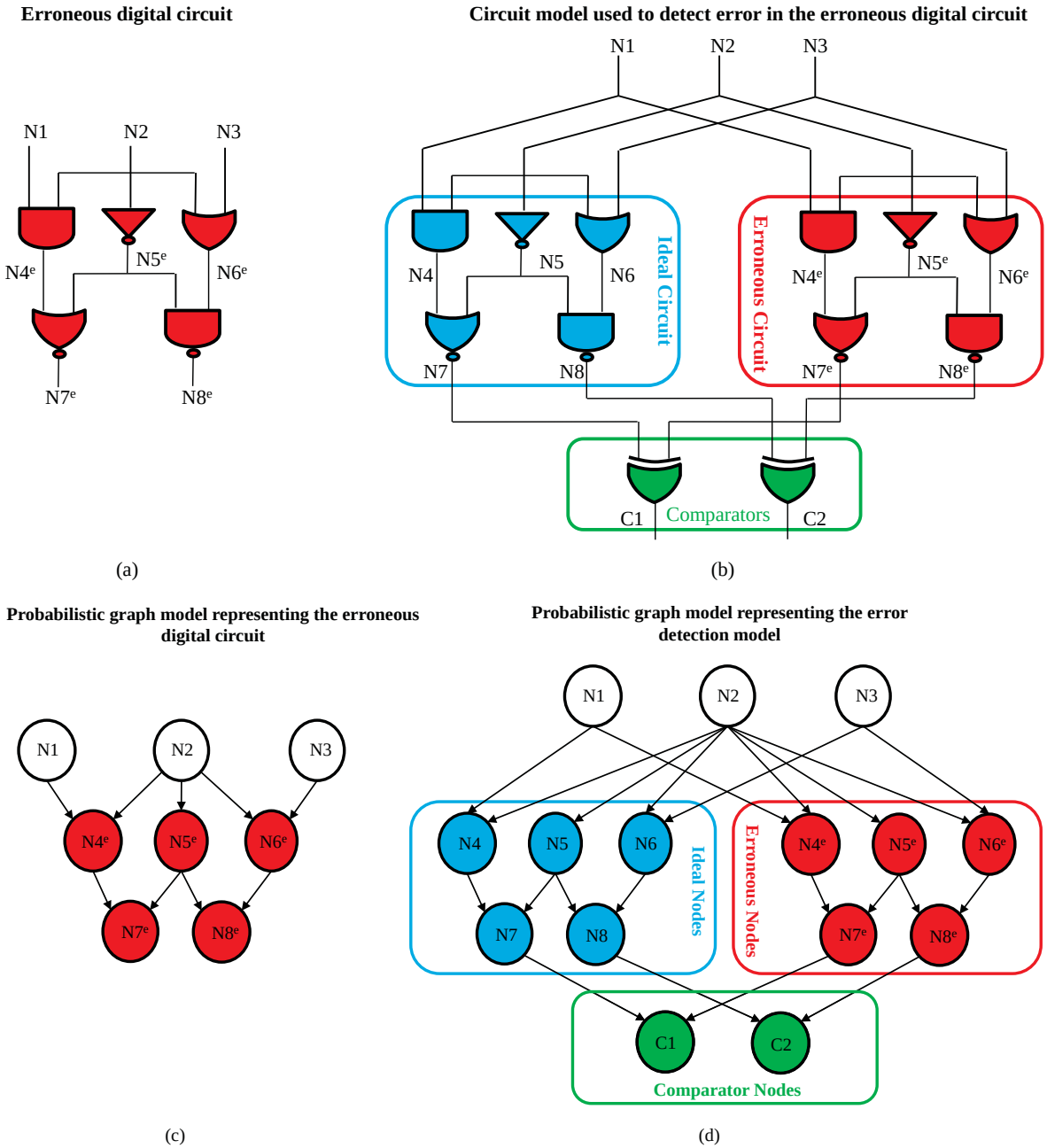


Figure 3.3. Error model

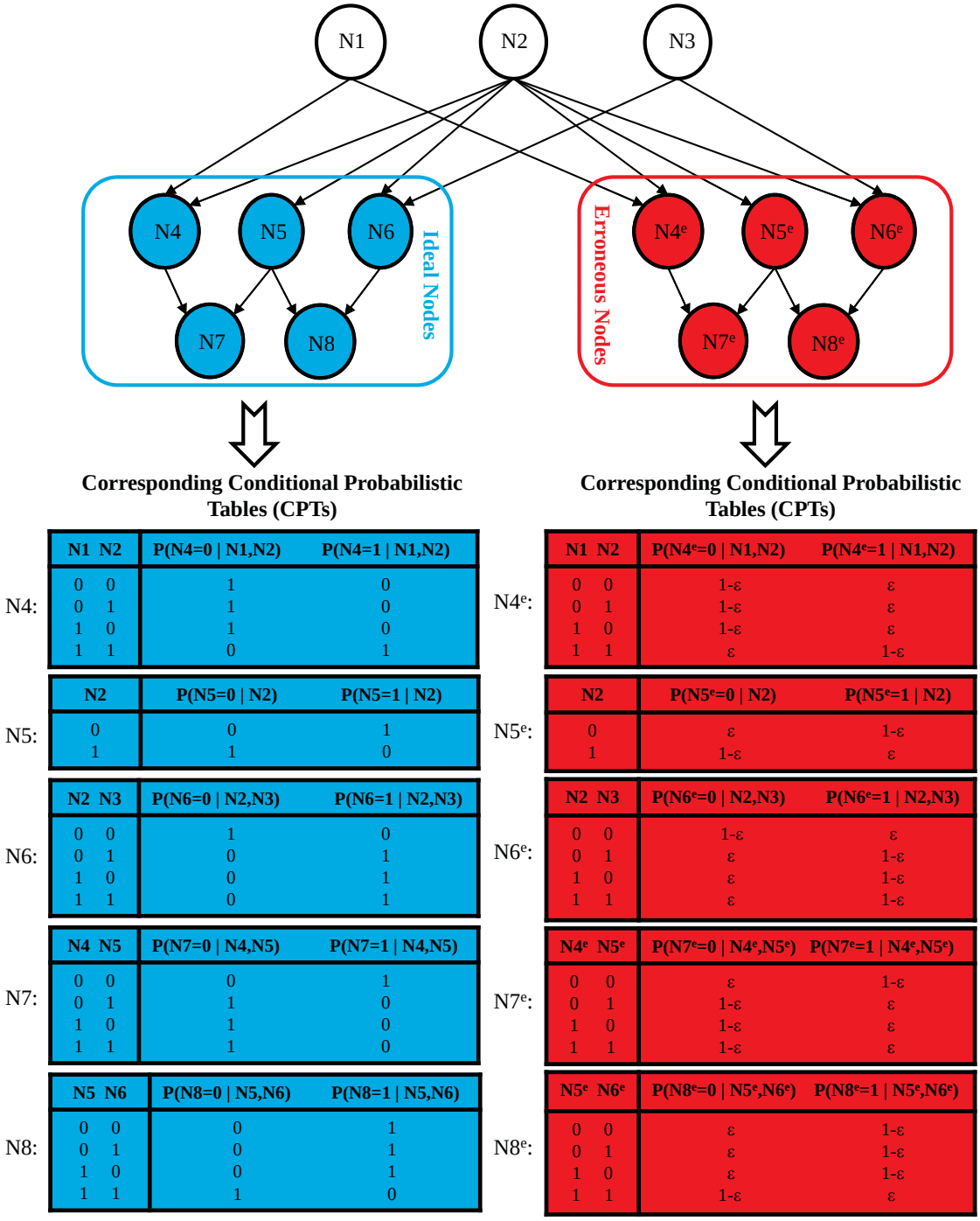


Figure 3.4. Conditional probabilistic tables for the ideal and erroneous nodes in the error model

probabilistic graph model by changing the CPTs as follows,

$$P(N4^e = 0|N1, N2) = \begin{cases} \varepsilon & \text{if } N1\text{=1 and } N2\text{=1} \\ 1-\varepsilon & \text{otherwise} \end{cases}$$

$$P(N4^e = 1|N1, N2) = \begin{cases} 1-\varepsilon & \text{if } N1\text{=1 and } N2\text{=1} \\ \varepsilon & \text{otherwise} \end{cases} \tag{3.6}$$

where $N4^e$ is the erroneous output signal of a faulty AND gate as shown in Fig 3.3.(a). Accordingly, the corresponding CPTs for rest of the erroneous nodes are provided in Fig 3.4.

CHAPTER 4

MAXIMUM ERROR MODELING

In this chapter, we present a probabilistic model to study the *maximum* output error over all possible input space for a given logic circuit. We present a method to find out the worst-case input vector, i.e., the input vector that has the highest probability to give an error at the output. In the first step of our model, we convert the circuit into a corresponding *edge-minimal* probabilistic network that represents the basic logic function of the circuit by handling the interdependencies between the signals using random variables of interest in a composite joint probability distribution function $P(y_1, y_2, \cdots, y_N)$. Each node in this network corresponds to a random variable representing a signal in the digital circuit, and each edge corresponds to the logic governing the connected signals. The individual probability distribution for each node is given using conditional probability tables.

From this probabilistic network we obtain our probabilistic error model that consists of three blocks, (i) ideal error free logic, (ii) error prone logic where every gate has a gate error probability ε i.e., each gate can go wrong individually by a probabilistic factor ε and (iii) a detection unit that uses comparators to compare the error free and erroneous outputs. The error prone logic represents the real time circuit under test, whereas the ideal logic and the detection unit are fictitious elements used to study the circuit. Both the ideal logic and error prone logic would be fed by the primary inputs **I**. We denote all the internal nodes, both in the error free and erroneous portions, by **X** and the comparator outputs as **O**. The comparators are based on XOR logic and hence a state "1" would signify error at the output. An evidence set **o** is created by evidencing one or more of the variables in the comparator set **O** to state "1" ($P(O_i = 1) =$

1). Then performing MAP hypothesis on the probabilistic error model provides the worst-case input vector $\mathbf{i}_{MAP}$ which gives $\max_{\forall \mathbf{i}} P(\mathbf{i}, \mathbf{o})$. The maximum output error probability can be obtained from $P(O_i = 1)$ after instantiating the input nodes of probabilistic error model with $\mathbf{i}_{MAP}$ and inferencing. The process is repeated for increasing ε values and finally the ε value that makes at least one of the output signals completely random ($P(O_i = 0) = 0.5, P(O_i = 1) = 0.5$) is taken as the error bound for the given circuit.

It is obvious that we can arrive at MAP estimate by enumerating all possible input instantiations and compute the maximum $P(\mathbf{i}, \mathbf{o})$ by any probabilistic computing tool. The attractive feature of this MAP algorithm lies on eliminating a significant part of the input search-subtree based on an easily available upper-bound of $P(\mathbf{i}, \mathbf{o})$ by using probabilistic traversal of a binary Join tree with *Shenoy-Shafer* algorithm [23, 24]. The actual computation is divided into two theoretical components. First, we convert the circuit structure into a binary Join tree and employ Shenoy-Shafer algorithm, which is a two-pass probabilistic message-passing algorithm, to obtain multitude of upper bounds of $P(\mathbf{i}, \mathbf{o})$ with partial input instantiations. Next, we construct a Binary tree of the input vector space where each path from the root node to the leaf node represents an input vector. At every node, we traverse the search tree if the upper bound, obtained by Shenoy-Shafer inference on the binary join tree, is greater than the maximum probability already achieved; otherwise we prune the entire sub-tree. Experimental results on a few standard benchmark show that the worst-case errors significantly deviate from the average ones and also provides tighter bounds for the ones that use homogeneous gate-type (c17 with NAND-only). Salient features and deliverables are itemized below:

- We have proposed a method to calculate *maximum* output error using a probabilistic model. Through experimental results, we show the importance of modeling maximum output error. (Fig. 4.10.)

- Given a circuit with a fixed gate error probability ε, our model can provide the maximum output error probability and the *worst-case* input vector, which can be very useful testing parameters.

- We present the circuit-specific error bounds for fault-tolerant computation and we show that maximum output errors provide a tighter bound.

- We have used an efficient design framework that employs inference in binary join trees using Shenoy-Shafer algorithm to perform MAP hypothesis accurately.

- We give a probabilistic error model, where efficient error incorporation is possible, for useful reliability studies. Using our model the error injection and probability of error for each gate can be modified easily. Moreover, we can accommodate both fixed and variable gate errors in a single circuit without affecting computational complexity.

We would like the readers to note that we will be representing a set of variables by bold capital letters, set of instantiations by bold small letters, any single variable by capital letters. Also probability of the event $Y_i = y_i$ will be denoted simply by $P(y_i)$ or by $P(Y_i = y_i)$.

4.1 Maximum a Posteriori (MAP) Estimate

Let us define the random variables in our probabilistic error model as $\mathbf{Y} = \mathbf{I} \cup \mathbf{X} \cup \mathbf{O}$, composed of the three disjoint subsets **I**, **X** and **O** where

- $I_1, \cdots, I_k \in \mathbf{I}$ are the set of k primary inputs.

- $X_1, \cdots, X_m \in \mathbf{X}$ are the m internal logic signals for both the erroneous (every gate has a failure probability ε) and error-free ideal logic elements.

- $O_1, \cdots, O_n \in \mathbf{O}$ are the n comparator outputs, each one signifying the error in one of the primary outputs of the logic block.

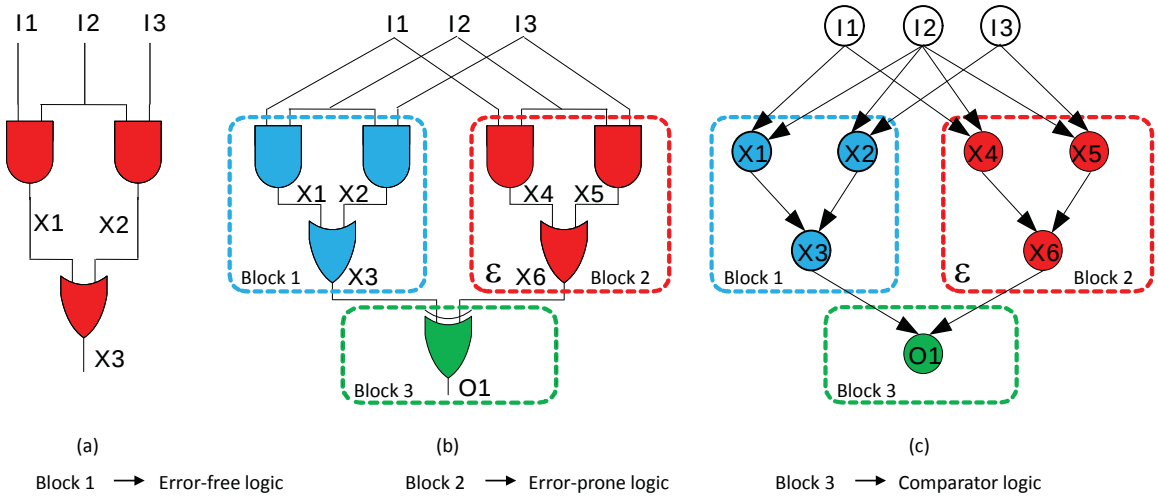


Figure 4.1. (a) Digital logic circuit (b) Error model (c) Probabilistic error model

- $N = k + m + n$ is the total number of network random variables.

Any primary output node can be forced to be erroneous by fixing the corresponding comparator output to logic "1", that is providing an *evidence* $\mathbf{o} = \{P(O_i = 1) = 1\}$ to a comparator output O_i. Given some evidence $\mathbf{o}$, the objective of the Maximum *a posteriori* estimate is to find a complete instantiation $\mathbf{i}_{MAP}$ of the variables in $\mathbf{I}$ that gives the following joint probability,

$$MAP(\mathbf{i}_{MAP}, \mathbf{o}) = \max_{\forall \mathbf{i}} P(\mathbf{i}, \mathbf{o}) \tag{4.1}$$

The probability $MAP(\mathbf{i}_{MAP}, \mathbf{o})$ is termed as the *MAP probability* and the variables in $\mathbf{I}$ are termed as *MAP variables* and the instantiation $\mathbf{i}_{MAP}$ which gives the maximum $P(\mathbf{i}, \mathbf{o})$ is termed as the *MAP instantiation*.

For example, consider Fig 4.1. In the probabilistic model shown in Fig 4.1.(c), we have $\{I1, I2, I3\} \in \mathbf{I}$; $\{X1, X2, X3, X4, X5, X6\} \in \mathbf{X}$; $\{O1\} \in \mathbf{O}$. $X3$ is the ideal error-free primary output node and $X6$ is the corresponding error-prone primary output node. Giving an evidence $\mathbf{o} = \{P(O1 = 1) = 1\}$ to $O1$ indicates that $X6$ has produced an erroneous output. The MAP hypothesis uses this information and finds the input instantiation, $\mathbf{i}_{MAP}$, that would give the

maximum $P(\mathbf{i},\mathbf{o})$. This indicates that $\mathbf{i}_{MAP}$ is the most probable input instantiation that would give an error in the error-prone primary output signal $X6$. In this case, $\mathbf{i}_{MAP} = \{I1 = 0, I2 = 0, I3 = 0\}$. This means that the input instantiation $\{I1 = 0, I2 = 0, I3 = 0\}$ will most probably provide a wrong output, $X6 = 1$ (since the correct output is $X6 = 0$).

We arrive at the exact Maximum *a posteriori* (MAP) estimate using the algorithms by Park and Darwiche [29] [30]. It is obvious that we could arrive at MAP estimate by enumerating all possible input instantiations and compute the maximum output error. To make it more efficient, our MAP estimates rely on eliminating some part of the input search-subtree based on an easily available upper-bound of MAP probability by using a probabilistic traversal of a binary Join tree using *Shenoy-Shafer* algorithm [23, 24]. The actual computation is divided into two theoretical components.

- First, we convert the circuit structure into a binary Join tree and employ Shenoy-Shafer algorithm, which is a two-pass probabilistic message-passing algorithm, to obtain multitude of upper bounds of MAP probability with partial input instantiations (discussed in Section. 4.1.1). The reader familiar with Shenoy-Shafer algorithm can skip the above section. To our knowledge, Shenoy-Shafer algorithm is not commonly used in VLSI context, so we elaborate most steps of join tree creation, two-pass join tree traversal and computation of upper bounds with partial input instantiations.

- Next, we construct a Binary tree of the input vector space where each path from the root node to the leaf node represents an input vector. At every node, we traverse the search tree if the upper bound, obtained by Shenoy-Shafer inference on the binary join tree, is greater than the maximum probability already achieved; otherwise we prune the entire sub-tree. The depth-first traversal in the binary input instantiation tree is discussed in Section. 4.1.2 where we detail the search process, pruning and heuristics used for better

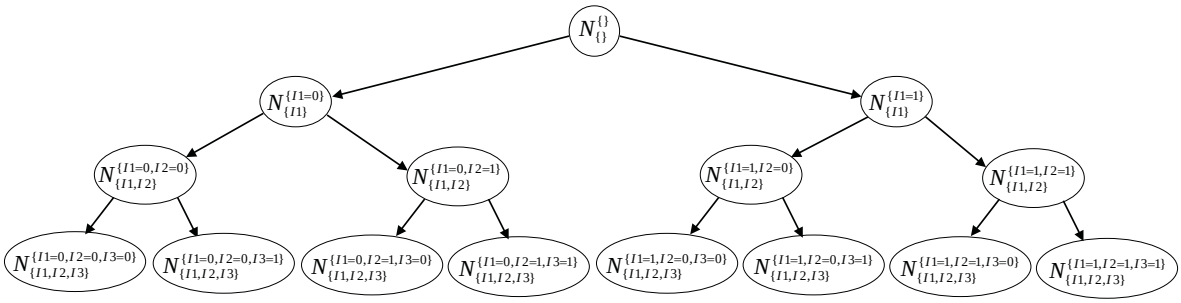


Figure 4.2. Search tree where depth first branch and bound search performed

pruning. Note that the pruning is key to the significantly improved efficiency of the MAP estimates.

4.1.1 Calculation of MAP Upper Bounds Using Shenoy-Shafer Algorithm

To clearly understand the various MAP probabilities that are calculated during MAP hypothesis, let us see the binary search tree formed using the MAP variables. A complete search through the MAP variables can be illustrated as shown in Fig. 4.2. which gives the corresponding search tree for the probabilistic error model given in Fig. 4.1.(c). In this search tree, the root node N will have an empty instantiation; every intermediate node $N^{\mathbf{i}_{inter}}_{\mathbf{I}_{inter}}$ will be associated with a subset $\mathbf{I}_{inter}$ of MAP variables $\mathbf{I}$ and the corresponding partial instantiation $\mathbf{i}_{inter}$; and every leaf node $N^{\mathbf{i}}_{\mathbf{I}}$ will be associated with the entire set $\mathbf{I}$ and the corresponding complete instantiation $\mathbf{i}$. Also each node will have v children where v is the number of values or states that can be assigned to each variable I_i. Since we are dealing with digital signals, every node in the search tree will have two children. Since the MAP variables represent the primary input signals of the given digital circuit, one path from the root to the leaf node of this search tree gives one input vector choice. In Fig. 4.2., at node $N^{01}_{\{I1,I2\}}$, $\mathbf{I}_{inter} = \{I1, I2\}$ and $\mathbf{i}_{inter} = \{I1 = 0, I2 = 1\}$. The basic idea of the search process is to find the MAP probability $MAP(\mathbf{i}, \mathbf{o})$ by finding the upper bounds of the intermediate MAP probabilities $MAP(\mathbf{i}_{inter}, \mathbf{o})$.

MAP hypothesis can be categorized into two portions. The first portion involves finding intermediate *upper bounds* of MAP probability, $MAP(\mathbf{i}_{inter}, \mathbf{o})$, and the second portion involves *improving* these bounds to arrive at the exact MAP solution, $MAP(\mathbf{i}_{MAP}, \mathbf{o})$. These two portions are intertwined and performed alternatively to effectively improve on the intermediate MAP upper bounds. These upper bounds and final solution are calculated by performing inference on the probabilistic error model using Shenoy-Shafer algorithm [23, 24].

Shenoy-Shafer algorithm is based on local computation mechanism. The probability distributions of the locally connected variables are propagated to get the joint probability distribution of the entire network from which any individual or joint probability distributions can be calculated. The Shenoy-shafer algorithm involves the following crucial information and calculations.

- *Valuations*: The valuations are functions based on the prior probabilities of the variables in the network. A valuation for a variable Y_i can be given as $\phi_{Y_i} = P(Y_i, Pa(Y_i))$ where $Pa(Y_i)$ are the parents of Y_i. For variables without parents, the valuations can be given as $\phi_{Y_i} = P(Y_i)$. These valuations can be derived from the CPTs as shown in Table 4.1.

- *Combination*: Combination is a pointwise multiplication mechanism conducted to combine the information provided by the operand functions. A combination of two given functions f_a and f_b can be written as $f_{a\cup b} = f_a \otimes f_b$, where a and b are set of variables. Table 4.2. provides an example.

- *Marginalization*: Given a function $f_{a\cup b}$, where a and b are set of variables, marginalizing over b provides a function of a and that can be given as $f_a = f_{a\cup b}^{mar(b)}$. This process provides the marginals of a single variable or a set of variables. Generally the process can be done by summing or maximizing or minimizing over the *marginalizing variables* in b. Normally the summation operator is used to calculate the probability distributions. In MAP hypothesis both summation and maximization operators are involved.

Table 4.1. Valuations of the variables derived from corresponding CPTs

CPT **Valuation**

Error-free AND		
$P(X1=1\|I1,I2)$	$P(I2=0)=1$	$P(I2=1)=1$
$P(I1=0)=1$	0	0
$P(I1=1)=1$	0	1

Error-free AND			
$X1$	$I1$	$I2$	ϕ_{X1}
0	0	0	1
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	1

Error-prone AND		
$P(X4=1\|I1,I2)$	$P(I2=0)=1$	$P(I2=1)=1$
$P(I1=0)=1$	ε	ε
$P(I1=1)=1$	ε	1-ε

Error-prone AND			
$X4$	$I1$	$I2$	ϕ_{X4}
0	0	0	1-ε
0	0	1	1-ε
0	1	0	1-ε
0	1	1	ε
1	0	0	ε
1	0	1	ε
1	1	0	ε
1	1	1	1-ε

Input	
$P(I1=0)$	0.5
$P(I1=1)$	0.5

Input	
$I1$	ϕ_{I1}
0	0.5
1	0.5

Table 4.2. Combination

x	y	f_{xy}
0	0	1
0	1	1
1	0	1
1	1	0

y	z	f_{yz}
0	0	1
0	1	0
1	0	0
1	1	0

x	y	z	$f_{xyz} = f_{xy} \otimes f_{yz}$
0	0	0	1x1
0	0	1	1x0
0	1	0	1x0
0	1	1	1x0
1	0	0	1x1
1	0	1	1x0
1	1	0	0x0
1	1	1	0x0

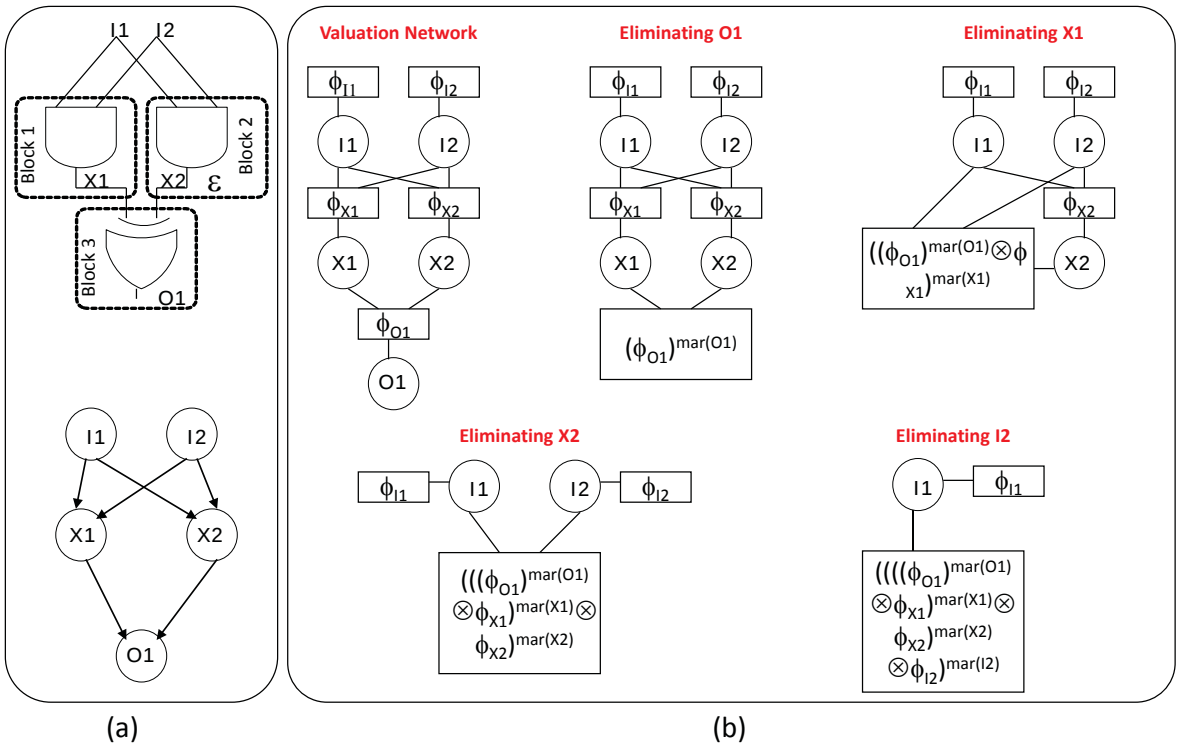


Figure 4.3. Illustration of the fusion algorithm

The computational scheme of the Shenoy-Shafer algorithm is based on *fusion* algorithm proposed by Shenoy in [25]. Given a probabilistic network, like our probabilistic error model in Fig. 4.3.(a), the *fusion* method can be explained as follows,

- The valuations provided are associated with the corresponding variables forming a valuation network as shown in Fig. 4.3.(b). In our example, the valuations are ϕ_{I1} for $\{I1\}$, ϕ_{I2} for $\{I2\}$, ϕ_{X1} for $\{X1, I1, I2\}$, ϕ_{X2} for $\{X2, I1, I2\}$, ϕ_{O1} for $\{O1, X1, X2\}$.

- A variable $Y_i \in \mathbf{Y}$ for which the probability distribution has to be found out is selected. In our example let us say we select $I1$.

- Choose an arbitrary variable elimination order. For the example network let us choose the order as O1,X1,X2,I2. When a variable Y_i is eliminated, the functions associated with that variable $f_{Y_i}^1, \cdots f_{Y_i}^j$ are combined and the resulting function is marginalized over Y_i. It can be represented as, $(f_{Y_i}^1 \otimes \cdots \otimes f_{Y_i}^j)^{mar(Y_i)}$. This function is then associated with the neighbors of Y_i. This process is repeated until all the variables in the elimination order are removed. Fig. 4.3. illustrates the fusion process.

  - Eliminating $O1$ yields the function $(\phi_{O1})^{mar(O1)}$ associated to neighbors $X1, X2$.
  - Eliminating $X1$ yields the function $((\phi_{O1})^{mar(O1)} \otimes \phi_{X1})^{mar(X1)}$ associated to neighbors $X2, I1, I2$.
  - Eliminating $X2$ yields the function $(((\phi_{O1})^{mar(O1)} \otimes \phi_{X1})^{mar(X1)} \otimes \phi_{X2})^{mar(X2)}$ associated to neighbors $I1, I2$.
  - Eliminating $I2$ yields the function $((((\phi_{O1})^{mar(O1)} \otimes \phi_{X1})^{mar(X1)} \otimes \phi_{X2})^{mar(X2)} \otimes \phi_{I2})^{mar(I2)}$ associated to neighbor $I1$.
  - According to a theorem presented in [24], combining the functions associated with $I1$ yields the probability distribution of $I1$. $\phi_{I1} \otimes ((((\phi_{O1})^{mar(O1)} \otimes \phi_{X1})^{mar(X1)} \otimes$

$\phi_{X2})^{mar(X2)} \otimes \phi_{I2})^{mar(I2)} = (\phi_{I1} \otimes \phi_{O1} \otimes \phi_{X1} \otimes \phi_{X2} \otimes \phi_{I2})^{mar(O1,X1,X2,I2)}$ = Probability distribution of I1 [24]. Note that the function $\phi_{I1} \otimes \phi_{O1} \otimes \phi_{X1} \otimes \phi_{X2} \otimes \phi_{I2}$ represents the joint probability of the entire probabilistic error model.

- The above process is repeated for all the other variables individually.

To perform efficient computation, an additional undirected network called *join tree* is formed from the original probabilistic network. The nodes of the join tree contains *clusters* of nodes from the original probabilistic network. The information of locally connected variables, provided through valuations, is propagated in the join tree by *message passing* mechanism. To increase the computational efficiency of the Shenoy-Shafer algorithm, a special kind of join tree named *binary join tree* is used. In a binary join tree, every node is connected to no more than three neighbors. In this framework only two functions are combined at an instance, thereby reducing the computational complexity. We will first explain the method to construct a binary join tree, as proposed by Shenoy in [24], and then we will explain the inference scheme using message passing mechanism.

The binary join tree is constructed using the fusion algorithm. The construction of binary join tree can be explained as follows,

- To begin with we have,
  - $\Lambda \Longrightarrow$ A set that contains all the variables from the original probabilistic network. In our example, $\Lambda = \{I1, I2, X1, X2, O1\}$.
  - $\Gamma \Longrightarrow$ A set that contains the subsets of variables, that should be present in the binary join tree. i.e., the subsets that denote the valuations and the subsets whose probability distributions are needed to be calculated. In our example, let us say that we need to calculate the individual probability distributions of all the variables. Then we have, $\Gamma = \{\{$I1$\}, \{$I2$\}, \{$X1,I1,I2$\}, \{$X2,I1,I2$\}, \{$O1,X1,X2$\}, \{$X1$\}, \{$X2$\}, \{$O1$\}\}$.

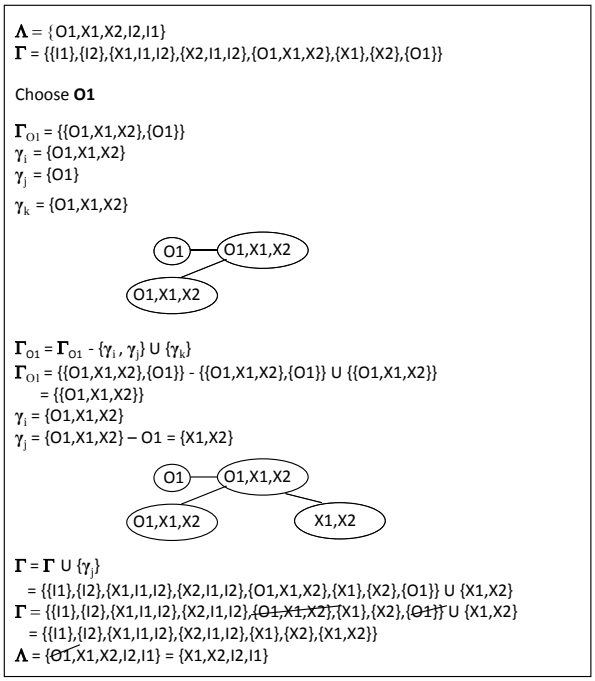


Figure 4.4. Partial illustration of binary join tree construction method for the first chosen variable

- $N \Longrightarrow$ A set that contains the nodes of the binary join tree and it is initially null.
- $E \Longrightarrow$ A set that contains the edges of the binary join tree and it is initially null.
- We also need an order in which we can choose the variables to form the binary join tree. In our example, since the goal is to find out the probability distribution of I1, this order should reflect the variable elimination order (O1,X1,X2,I2,I1) used in fusion algorithm .

- 1: **while** $|\Gamma| > 1$ **do**
  - 2: Choose a variable $Y \in \Lambda$
  - 3: $\Gamma_Y = \{\gamma_i \in \Gamma | Y \in \gamma_i\}$
  - 4: **while** $|\Gamma_Y| > 1$ **do**
    - 5: Choose $\gamma_i \in \Gamma_Y$ and $\gamma_j \in \Gamma_Y$ such that $||\gamma_i \cup \gamma_j|| \leq ||\gamma_m \cup \gamma_n||$ for all $\gamma_m, \gamma_n \in \Gamma_Y$
    - 6: $\gamma_k = \gamma_i \cup \gamma_j$

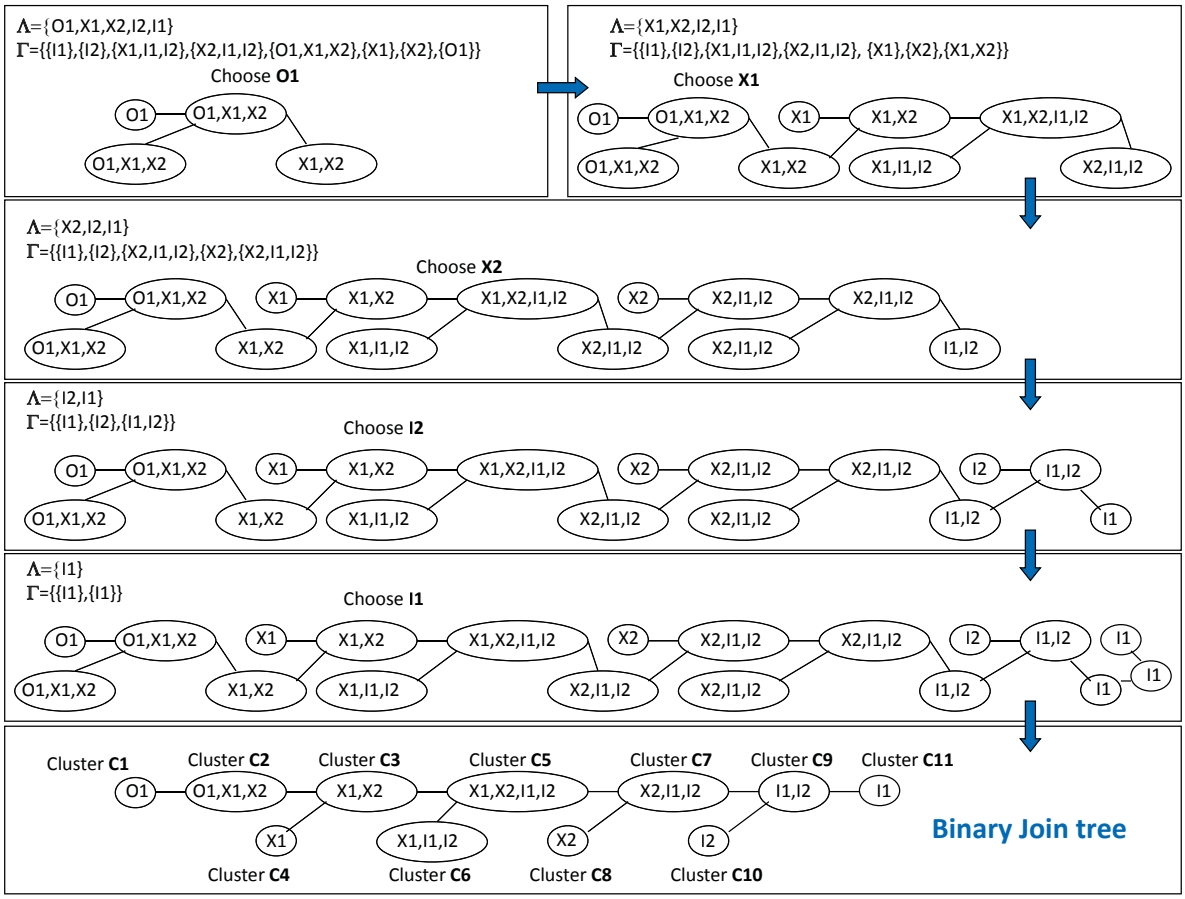


Figure 4.5. Complete illustration of binary join tree construction method

7: $N = N \cup \{\gamma_i\} \cup \{\gamma_j\} \cup \{\gamma_k\}$

8: $E = E \cup \{\{\gamma_i, \gamma_k\}, \{\gamma_j, \gamma_k\}\}$

9: $\Gamma_Y = \Gamma_Y - \{\gamma_i, \gamma_j\}$

10: $\Gamma_Y = \Gamma_Y \cup \{\gamma_k\}$

11: **end while**

12: **if** $|\Lambda| > 1$ **then**

13: Take γ_i where $\gamma_i = \Gamma_Y$

14: $\gamma_j = \gamma_i - \{Y\}$

15: $N = N \cup \{\gamma_i\} \cup \{\gamma_j\}$

16: $E = E \cup \{\{\gamma_i, \gamma_j\}\}$

17: $\Gamma = \Gamma \cup \{\gamma_j\}$

18: **end if**

19: $\Gamma = \Gamma - \{\gamma_i \in \Gamma | Y \in \gamma_i\}$

20: $\Lambda = \Lambda - \{Y\}$

21: **end while**

- The final structure will have some duplicate clusters. Two neighboring duplicate clusters can be merged into one, if the merged node does not end up having more than three neighbors. After merging the duplicate nodes we get the binary join tree.

Fig. 4.4. and Fig. 4.5. illustrate the binary join tree construction method for the probabilistic error model in Fig. 4.3.(a). Fig. 4.4. explains a portion of the construction method for the first chosen variable, here it is $O1$. Fig. 4.5. illustrates the entire method. Note that, even though the binary join tree is constructed with a specific variable elimination order for finding out the probability distribution of I1, it can be used to find out the probability distributions of other variables too.

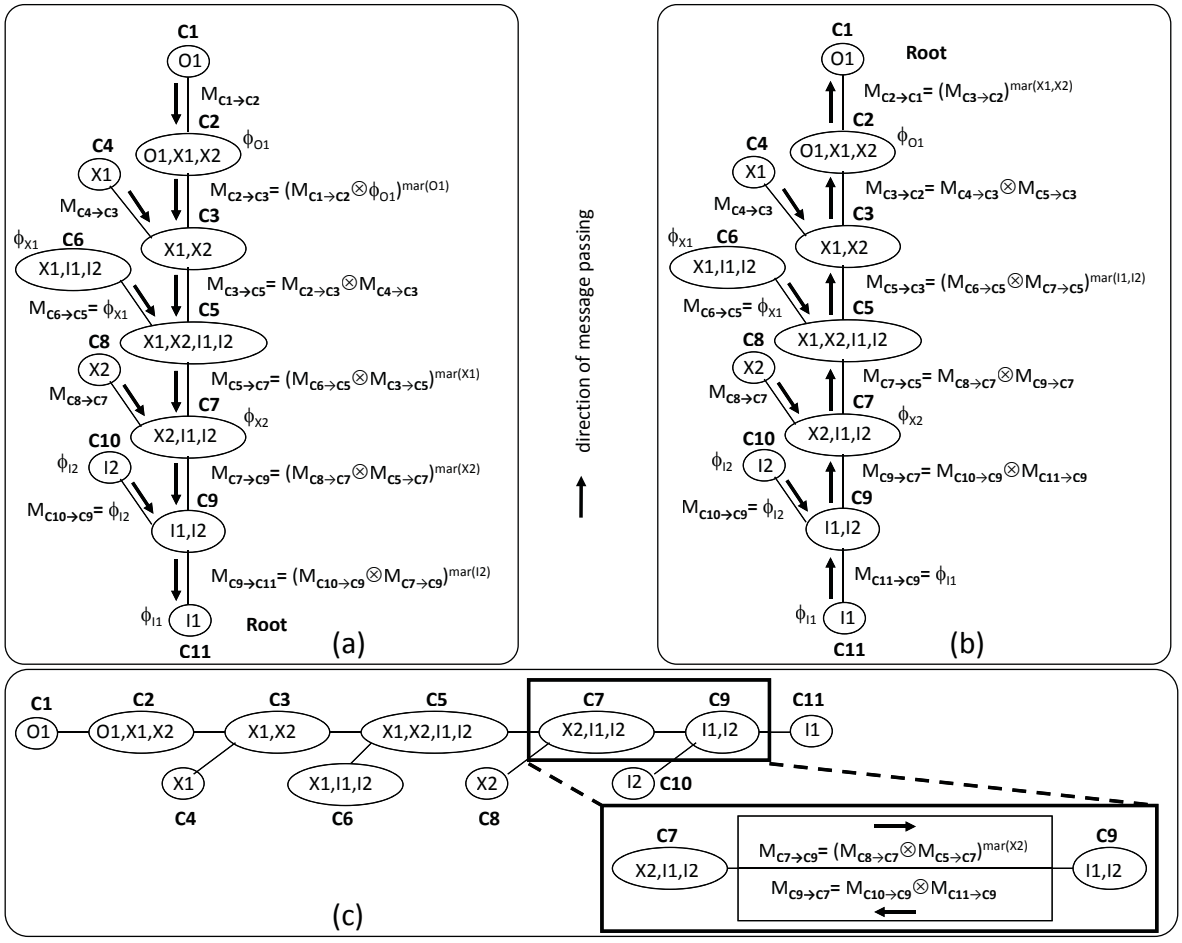


Figure 4.6. (a) Message passing with cluster C11 as root (b) Message passing with cluster C1 as root (c) Message storage mechanism

Inference in a binary join tree is performed using message passing mechanism. Initially all the valuations are associated to the appropriate clusters. In our example, at Fig. 4.6., the valuations are associated to these following clusters,

- ϕ_{I1} associated to cluster **C11**
- ϕ_{I2} associated to cluster **C10**
- ϕ_{X1} associated to cluster **C6**
- ϕ_{X2} associated to cluster **C7**
- ϕ_{O1} associated to cluster **C2**

A message passed from cluster b, containing a variable set **B**, to cluster c, containing a variable set **C** can be given as,

$$M_{b\to c} = (\phi_b \prod_{a\neq c} M_{a\to b})^{mar(\mathbf{B}\setminus\mathbf{C})} \tag{4.2}$$

where ϕ_b is the valuation associated with cluster b. If cluster b is not associated with any valuation, then this function is omitted from the equation. The message from cluster b can be sent to cluster c only after cluster b receives messages from all its neighbors other than c. The resulting function is marginalized over the variables in cluster b that are not in cluster c. To calculate the probability distribution of a variable Y_i, the cluster having that variable alone is taken as root and the messages are passed towards this root. Probability of Y_i, $P(Y_i)$, is calculated at the root. In our example, at Fig. 4.6.(a), to find the probability distribution of I1, the cluster **C11** is chosen as the root. The messages from all the leaf clusters are sent towards **C11** and finally the probability distribution of I1 can be calculated as, $P(I1) = M_{\mathbf{C9}\to\mathbf{C11}} \otimes \phi_{I1}$. Also note that the *order of the marginalizing variables* is O1,X1,X2,I2 which exactly reflects the elimination order used to construct the binary join tree. As we mentioned before, this binary join tree can be used to calculate probability distributions of other variables

also. In our example, at Fig. 4.6.(b), to find out the probability distribution of O1, cluster **C1** is chosen as root and the messages from the leaf clusters are passed towards **C1** and finally the probability distribution of O1 can be calculated as, $P(O1) = M_{\mathbf{C2}\rightarrow\mathbf{C1}}$. Note that the *order of the marginalizing variables* changes to I1,I2,X1,X2. We can also calculate joint probability distributions of the set of variables that forms a cluster in the binary join tree. In our example, the joint probability $P(I1, I2)$ can be calculated by assigning cluster **C9** as root. In this fashion, the probability distributions of any individual variable or a set of variables can be calculated by choosing appropriate root cluster and sending the messages towards this root. During these operations some of the calculations are not modified and so performing them again will prove inefficient. Using the binary join tree structure these calculations can be stored thereby eliminating the redundant recalculation. In the binary join tree, between any two clusters b and c, both the messages $M_{b\rightarrow c}$ and $M_{c\rightarrow b}$ are stored. Fig. 4.6.(c) illustrates this phenomenon using our example.

If an evidence set **e** is provided, then the additional valuations $\{e_{Y_i} | Y_i \in \mathbf{e}\}$ provided by the evidences has to be associated with the appropriate clusters. A valuation e_{Y_i} for a variable Y_i can be associated with a cluster having Y_i alone. In our example, if the variable O1 is evidenced, then the corresponding valuation e_{O1} can be associated with cluster **C1**. While finding the probability distribution of a variable Y_i, the inference mechanism (as explained before) with an evidence set **e** will give the probability $P(Y_i, \mathbf{e})$ instead of $P(Y_i)$. From $P(Y_i, \mathbf{e})$, $P(\mathbf{e})$ is calculated as, $P(\mathbf{e}) = \sum_{Y_i} P(Y_i, \mathbf{e})$. Calculation of the probability of evidence $P(\mathbf{e})$ is crucial for MAP calculation.

The MAP probabilities $MAP(\mathbf{i}_{inter}, \mathbf{o})$ are calculated by performing inference on the binary join tree with evidences $\mathbf{i}_{inter}$ and **o**. Let us say that we have an evidence set $\mathbf{e} = \{\mathbf{i}_{inter}, \mathbf{o}\}$, then $MAP(\mathbf{i}_{inter}, \mathbf{o}) = P(\mathbf{e})$. For a given partial instantiation $\mathbf{i}_{inter}$, $MAP(\mathbf{i}_{inter}, \mathbf{o})$ is calculated by maximizing over the MAP variables which are not evidenced. This calculation can be done by modifying the message passing scheme to accommodate maximization over unevidenced

MAP variables. So for MAP calculation, the marginalization operation involves both maximization and summation functions. The maximization is performed over the unevidenced MAP variables in **I** and the summation is performed over all the other variables in **X** and **O**. For MAP, a message passed from cluster *b* to cluster *c* is calculated as,

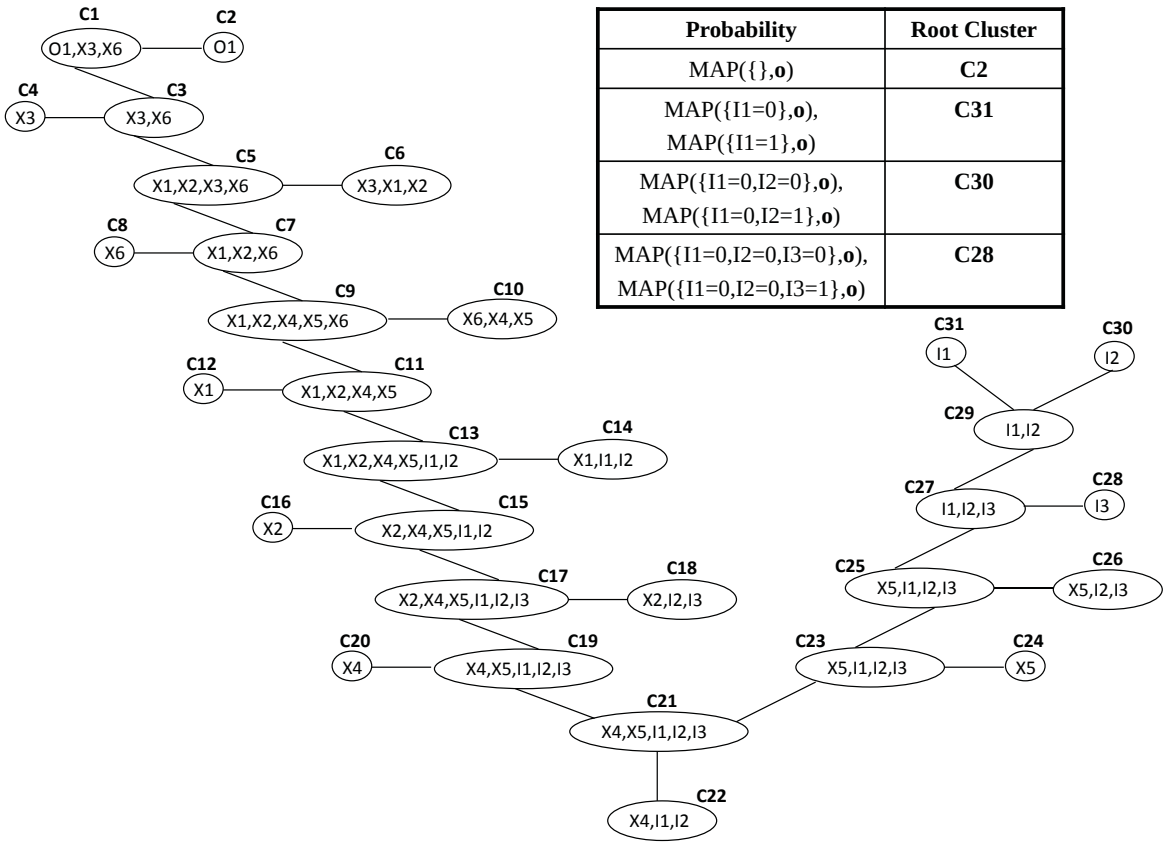
$$M_{b\to c} = \max_{\{\mathbf{I}_b\}\in\{\mathbf{B}\setminus\mathbf{C}\}} \sum_{\{\mathbf{X}_b\cup\mathbf{O}_b\}\in\{\mathbf{B}\setminus\mathbf{C}\}} \phi_b \prod_{a\neq c} M_{a\to b} \tag{4.3}$$

where $\mathbf{I}_b \subseteq \mathbf{I}\setminus\mathbf{I}_{inter}$, $\mathbf{X}_b \subseteq \mathbf{X}$, $\mathbf{O}_b \subseteq \mathbf{O}$ and $\{\mathbf{I}_b, \mathbf{X}_b, \mathbf{O}_b\} \in \mathbf{B}$.

Here the most important aspect is that the maximization and summation operators in Eqn. 4.3 are non-commutative.

$$[\sum_{\mathbf{X}} \max_{\mathbf{I}} P](\mathbf{y}) \geq [\max_{\mathbf{I}} \sum_{\mathbf{X}} P](\mathbf{y}) \tag{4.4}$$

So during message passing in the binary join tree, the *valid order of the marginalizing variables* or the *valid variable elimination order* should have the summation variables in **X** and **O** before the maximization variables in **I**. A message pass through an invalid variable elimination order can result in a bad upper bound that is stuck at a local maxima and it eventually results in the elimination of some probable instantiations of the MAP variables **I** during the search process. But an invalid elimination order can provide us an initial upper bound of the MAP probability to start with. The closer the invalid variable elimination order to the valid one, the tighter will be the upper bound. In the binary join tree, any cluster can be chosen as root to get this initial upper bound. For example, in Fig. 4.6.(b) choosing cluster **C1** as root results in an invalid variable elimination order (I1, I2, X1, X2) and message pass towards this root can give the initial upper bound. Also it is essential to use a valid variable elimination order during the construction of the binary join tree so that there is at least one path that can provide a good upper bound.



Probability	Root Cluster
MAP({},**o**)	**C2**
MAP({I1=0},**o**), MAP({I1=1},**o**)	**C31**
MAP({I1=0,I2=0},**o**), MAP({I1=0,I2=1},**o**)	**C30**
MAP({I1=0,I2=0,I3=0},**o**), MAP({I1=0,I2=0,I3=1},**o**)	**C28**

Figure 4.7. Binary join tree for the probabilistic error model in Fig. 4.1.(c)

Fig. 4.7. gives the corresponding binary join tree, for the probabilistic error model given in Fig. 4.1.(c), constructed with a valid variable elimination order (O1, X3, X6, X1, X2, X4, X5, I3, I2, I1). In this model, there are three MAP variables I1, I2, I3. The MAP hypothesis on this model results in $\mathbf{i}_{MAP} = \{I1 = 0, I2 = 0, I3 = 0\}$.

The initial upper bound $MAP(\{\}, \mathbf{o})$ is calculated by choosing cluster **C2** as root and passing messages towards **C2**. As specified earlier this upper bound can be calculated with any cluster as root. With **C2** as root, an upper bound will most certainly be obtained since the variable elimination order (I3, I2, I1, X4, X5, X1, X2, X3, X6) is an invalid one. But since the maximization variables are at the very beginning of the order, having **C2** as root will yield a looser upper bound. Instead, if **C16** is chosen as root, the elimination order (O1, X3, X6, X1, I3, X4, X5, I2, I1) will be closer to a valid order. So a much tighter upper bound can be achieved. To calculate an intermediate upper bound $MAP(\mathbf{i}_{inter}, \mathbf{o})$, the MAP variable I_i newly added to form $\mathbf{i}_{inter}$ is recognized and the cluster having the variable I_i alone is selected as root. By doing this a valid elimination order and proper upper bound can be achieved. For example, to calculate the intermediate upper bound $MAP(\{I1 = 0\}, \mathbf{o})$ where the instantiation $\{I1 = 0\}$ is newly added to the initially empty set $\mathbf{i}_{inter}$, a valid elimination order should have the maximization variables I2,I3 at the end. To achieve this, cluster **C31** is chosen as root thereby yielding a valid elimination order (O1, X3, X6, X1, X2, X4, X5, I3, I2).

4.1.2 Calculation of the Exact MAP Solution

The calculation of the exact MAP solution $MAP(\mathbf{i}_{MAP}, \mathbf{o})$ can be explained as follows,

- To start with we have the following,
  - $\mathbf{I}_{inter} \rightarrow$ subset of MAP variables $\mathbf{I}$. Initially empty.
  - $\mathbf{i}_{inter} \rightarrow$ partial instantiation set of MAP variables $\mathbf{I}_{inter}$. Initially empty.
  - $\mathbf{i}_{d_1}, \mathbf{i}_{d_2} \rightarrow$ partial instantiation sets used to store $\mathbf{i}_{inter}$. Initially empty.

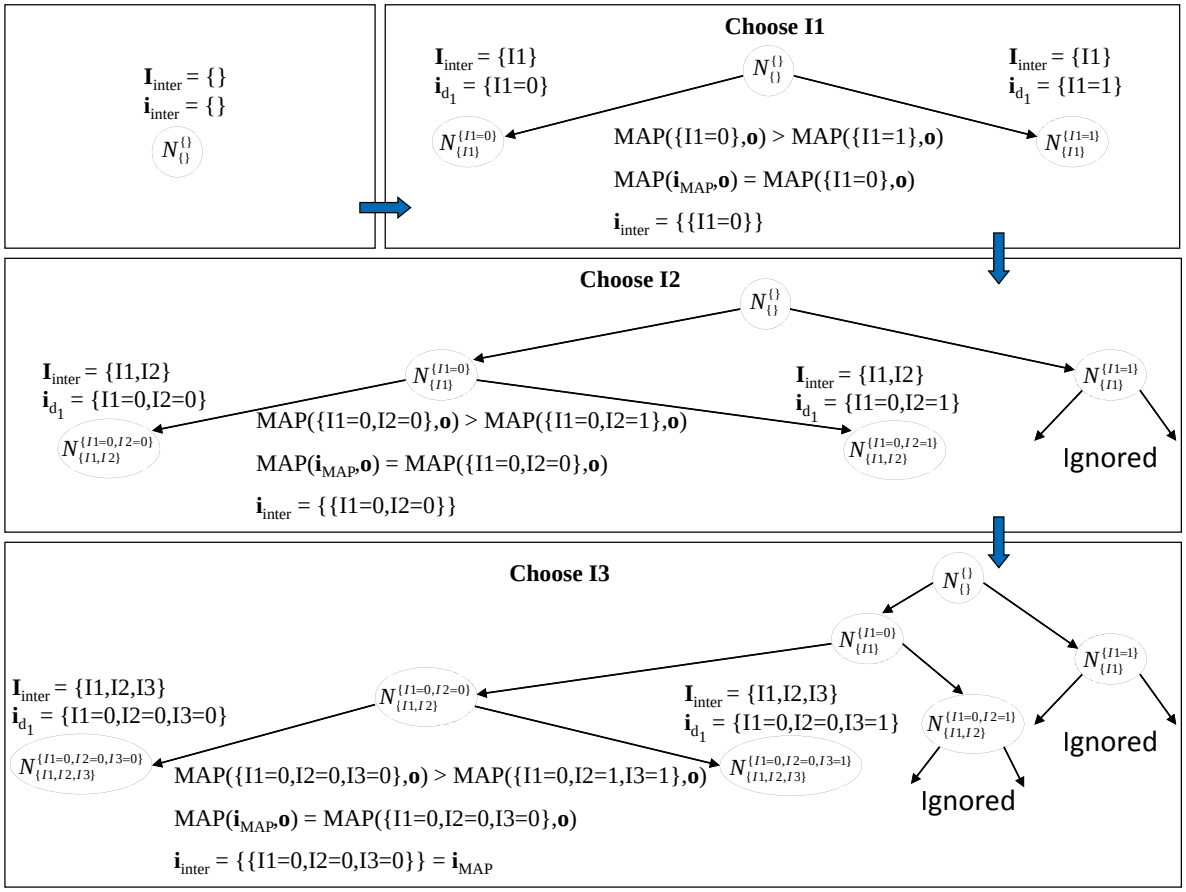


Figure 4.8. Search process for MAP computation

- $\mathbf{i}_{MAP} \rightarrow$ MAP instantiation. At first, $\mathbf{i}_{MAP} = \mathbf{i}_{init}$, where $\mathbf{i}_{init}$ is calculated by *sequentially* initializing the MAP variables to a particular instantiation and performing local *taboo search* around the neighbors of that instantiation [30].
- $MAP(\mathbf{i}_{MAP}, \mathbf{o}) \rightarrow$ MAP probability. Initially $MAP(\mathbf{i}_{MAP}, \mathbf{o}) = MAP(\mathbf{i}_{init}, \mathbf{o})$ calculated by inferencing the probabilistic error model.
- $v(I_i) \rightarrow$ number of values or states that can be assigned to a variable I_i. Since we are dealing with digital signals, $v(I_i) = 2$ for all i.

1: Calculate $MAP(\mathbf{i}_{inter}, \mathbf{o})$. /**This is the initial upper bound of MAP probability.*/

2: **if** $MAP(\mathbf{i}_{inter}, \mathbf{o}) \geq MAP(\mathbf{i}_{MAP}, \mathbf{o})$ **then**

3: $MAP(\mathbf{i}_{MAP}, \mathbf{o}) = MAP(\mathbf{i}_{inter}, \mathbf{o})$

4: **else**

5: $MAP(\mathbf{i}_{MAP}, \mathbf{o}) = MAP(\mathbf{i}_{MAP}, \mathbf{o})$

6: $\mathbf{i}_{MAP} = \mathbf{i}_{MAP}$

7: **end if**

8: **while** $|\mathbf{I}| > 0$ **do**

9: Choose a variable $I_i \in \mathbf{I}$.

10: $\mathbf{I}_{inter} = \mathbf{I}_{inter} \cup \{I_i\}$.

11: **while** $v(I_i) > 0$ **do**

12: Choose a value $i_{v(I_i)}$ of I_i

13: $\mathbf{i}_{d_1} = \mathbf{i}_{inter} \cup \{I_i = i_{v(I_i)}\}$.

14: Calculate $MAP(\mathbf{i}_{d_1}, \mathbf{o})$ from binary join tree.

15: **if** $MAP(\mathbf{i}_{d_1}, \mathbf{o}) \geq MAP(\mathbf{i}_{MAP}, \mathbf{o})$ **then**

16: $MAP(\mathbf{i}_{MAP}, \mathbf{o}) = MAP(\mathbf{i}_{d_1}, \mathbf{o})$

17: $\mathbf{i}_{d_2} = \mathbf{i}_{d_1}$

18: **else**

19: $MAP(\mathbf{i}_{MAP}, \mathbf{o}) = MAP(\mathbf{i}_{MAP}, \mathbf{o})$

20: **end if**

21: $v(I_i) = v(I_i) - 1$

22: **end while**

23: $\mathbf{i}_{inter} = \mathbf{i}_{d_2}$

24: **if** $|\mathbf{i}_{inter}| = 0$ **then**

25: goto line 29

26: **end if**

27: $\mathbf{I} = \mathbf{I} - \{I_i\}$

28: **end while**

29: **if** $|\mathbf{i}_{inter}| = 0$ **then**

30: $\quad \mathbf{i}_{MAP} = \mathbf{i}_{MAP}$

31: **else**

32: $\quad \mathbf{i}_{MAP} = \mathbf{i}_{inter}$

33: **end if**

The pruning of the search process is handled in lines 11-23. After choosing a MAP variable I_i, the partial instantiation set $\mathbf{i}_{inter}$ is updated by adding the best instantiation $I_i = i_{v(I_i)}$ thereby ignoring the other instantiations of I_i. This can be seen in Fig. 4.8. which illustrates the search process for MAP computation using the probabilistic error model given in Fig. 4.1.(c) as example.

4.1.3 Calculating the Maximum Output Error Probability

According to our error model, the MAP variables represent the primary input signals of the underlying digital logic circuit. So after MAP hypothesis, we will have the input vector which has the highest probability to give an error on the output. The random variables **I** that represent the primary input signals are then instantiated with $\mathbf{i}_{MAP}$ and inferenced. So the evidence set for this inference calculation will be $\mathbf{e} = \{\mathbf{i}_{MAP}\}$. The output error probability is obtained by observing the probability distributions of the comparator logic variables **O**. After inference, the probability distribution $P(O_i, \mathbf{e})$ will be obtained. From this $P(O_i|\mathbf{e})$ can be obtained as, $P(O_i|\mathbf{e}) = \frac{P(O_i,\mathbf{e})}{P(\mathbf{e})} = \frac{P(O_i,\mathbf{e})}{\sum_{O_i} P(O_i,\mathbf{e})}$. Finally the maximum output error probability is given by, $\max_i P(O_i = 1|\mathbf{e})$.

4.1.4 Computational Complexity of MAP Estimate

The time complexity of MAP depends on that of the depth first branch and bound search on the *input instantiation search tree* and also on that of *inference in binary join tree*. The

former depends on the number of MAP variables and the number of states assigned to each variable. In our case each variable is assigned two states and so the time complexity can be given as $O(2^k)$ where k is the number of MAP variables. This is the worst case time complexity assuming that the search tree is not pruned. If the search tree is pruned, then the time complexity will be $< O(2^k)$.

The time complexity of inference in the binary join tree depends on the number of cliques q and the size Z of the biggest clique. It can be represented as $q.2^Z$ and the worst case time complexity can be given as $O(2^Z)$. In any given probabilistic model with N variables, representing a joint probability $P(x_1, \cdots x_N)$, the corresponding join tree will have $Z < N$ always [27]. Also depending on the underlying circuit structure, the join tree of the corresponding probabilistic error model can have $Z << N$ or Z close to N, which in turn determines the time complexity.

Since for every pass in the search tree inference has to be performed in the join tree to get the upper bound of MAP probability, the worst case time complexity for MAP can be given as $O(2^{k+Z})$. The space complexity of MAP depends on the number of MAP variables for the search tree and on the number of variables N in the probabilistic error model and the size of the largest clique. It can be given by $2^k + N.2^Z$.

4.2 Experimental Results

The experiments are performed on ISCAS85 and MCNC benchmark circuits. The computing device used is a Sun server with 8 CPUs where each CPU consists of 1.5GHz UltraSPARC IV processor with at least 32GB of RAM.

4.2.1 Experimental Procedure for Calculating Maximum Output Error Probability

Our main goal is to provide the maximum output error probabilities for different gate error probabilities ε. To get the maximum output error probabilities every output signal of a circuit

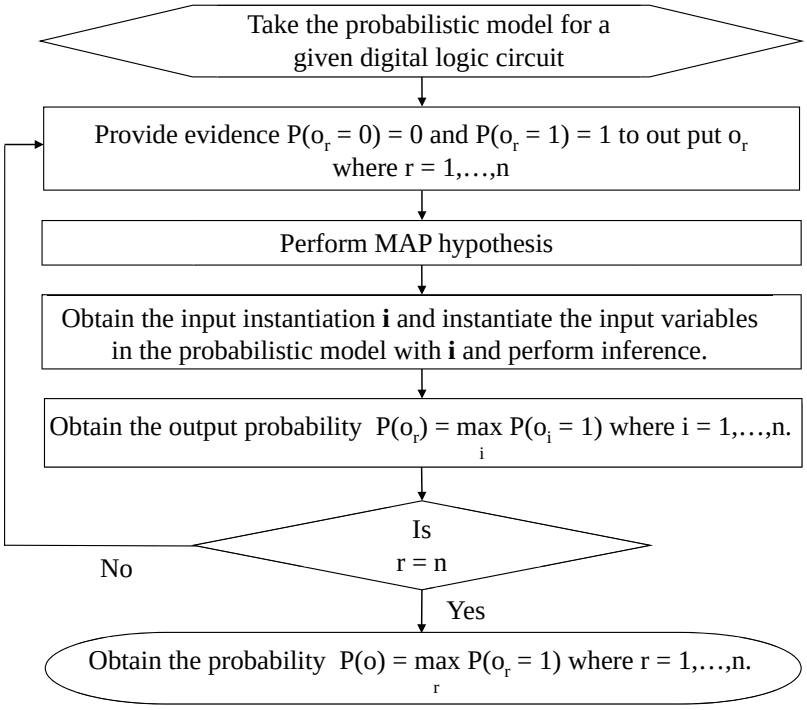


Figure 4.9. Flow chart describing the experimental setup and process

has to be examined through MAP estimation, which is performed through algorithms provided in [31]. The experimental procedure is illustrated as a flow chart in Fig. 4.9. The steps are as follows,

- First, an evidence has to be provided to one of the comparator output signal variables in set **O** such that $P(O_i = 0) = 0$ and $P(O_i = 1) = 1$. Recall that these variables have a probability distribution based on XOR logic and so giving evidence like this is similar to forcing the output to be wrong.

- The comparator outputs are evidenced individually and the corresponding input instantiations **i** are obtained by performing MAP.

- Then the primary input variables in the probabilistic error model are instantiated with each instantiation **i** and inferenced to get the output probabilities.

Table 4.3. Worst-case input vectors from MAP

Circuits	No. of Inputs	Input vector	Gate error probability ε
c17	5	01111	0.005 - 0.2
max_flat	8	00010011	0.005 - 0.025
		11101000	0.03 - 0.05
		11110001	0.055 - 0.2
voter	12	000100110110	0.01 - 0.19
		111011100010	0.2

- $P(O_i = 1)$ is noted from all the comparator outputs for each **i** and the maximum value gives the maximum output error probability.
- The entire operation is repeated for different ε values.

4.2.2 Worst-case Input Vectors

Table 4.3. gives the worst-case input vectors got from MAP i.e., the input vectors that gives maximum output error probability. The notable results are as follows,

- In *max_flat* and *voter* the worst-case input vectors from MAP changes with ε, while in *c*17 it does not change.
- In the range $\{0.005\text{-}0.2\}$ for ε, *max_flat* has three different worst-case input vectors while *voter* has two.
- It implies that these worst-case input vectors not only depend on the circuit structure but could dynamically change with ε. This could be of concern for designers as the worst-case inputs might change after gate error probabilities reduce due to error mitigation schemes. Hence, explicit MAP computation would be necessary to judge the maximum error probabilities and worst-case vectors after every redundancy schemes are applied.

4.2.3 Circuit-Specific Error Bounds for Fault-Tolerant Computation

The error bound for a circuit can be obtained by calculating the gate error probability ε that drives the output error probability of at least one output to a hard bound beyond which the output does not depend on the input signals or the circuit structure. When the output error probability reaches $0.5(50\%)$, it essentially means that the output signal behaves as a non-functional random number generator for at least one input vector and so 0.5 can be treated as a hard bound.

Fig. 4.10. gives the error bounds for various benchmark circuits. It also shows the comparison between maximum and average output error probabilities with reference to the change in gate error probability ε. These graphs are obtained by performing the experiment for different ε values ranging from 0.005 to 0.1. The average error probabilities are obtained from our previous work by Rejimon et al. [86]. The notable results are as follows,

- The $c17$ circuit consists of 6 NAND gates. The error bound for each NAND gate in $c17$ is $\varepsilon = 0.1055$, which is greater than the conventional error bound for NAND gate, which is 0.08856 [7, 8]. The error bound of the same NAND gate in *voter* circuit (contains 10 NAND gates, 16 NOT gates, 8 NOR gates, 15 OR gates and 10 AND gates) is $\varepsilon = 0.0292$, which is lesser than the conventional error bound. This indicates that the error bound for an individual *NAND gate placed in a circuit* can be dependent on the circuit structure. The same can be true for all other logics.

- The maximum output error probabilities are much larger than average output error probabilities, thereby reaching the hard bound for comparatively lower values of ε, making them a very crucial design parameter to achieve tighter error bounds. Only for $alu4$ and $malu4$, the average output error probability reaches the hard bound within $\varepsilon = 0.1 (\varepsilon = 0.095 for\ alu4, \varepsilon = 0.08 for\ malu4)$, while the maximum output error prob-

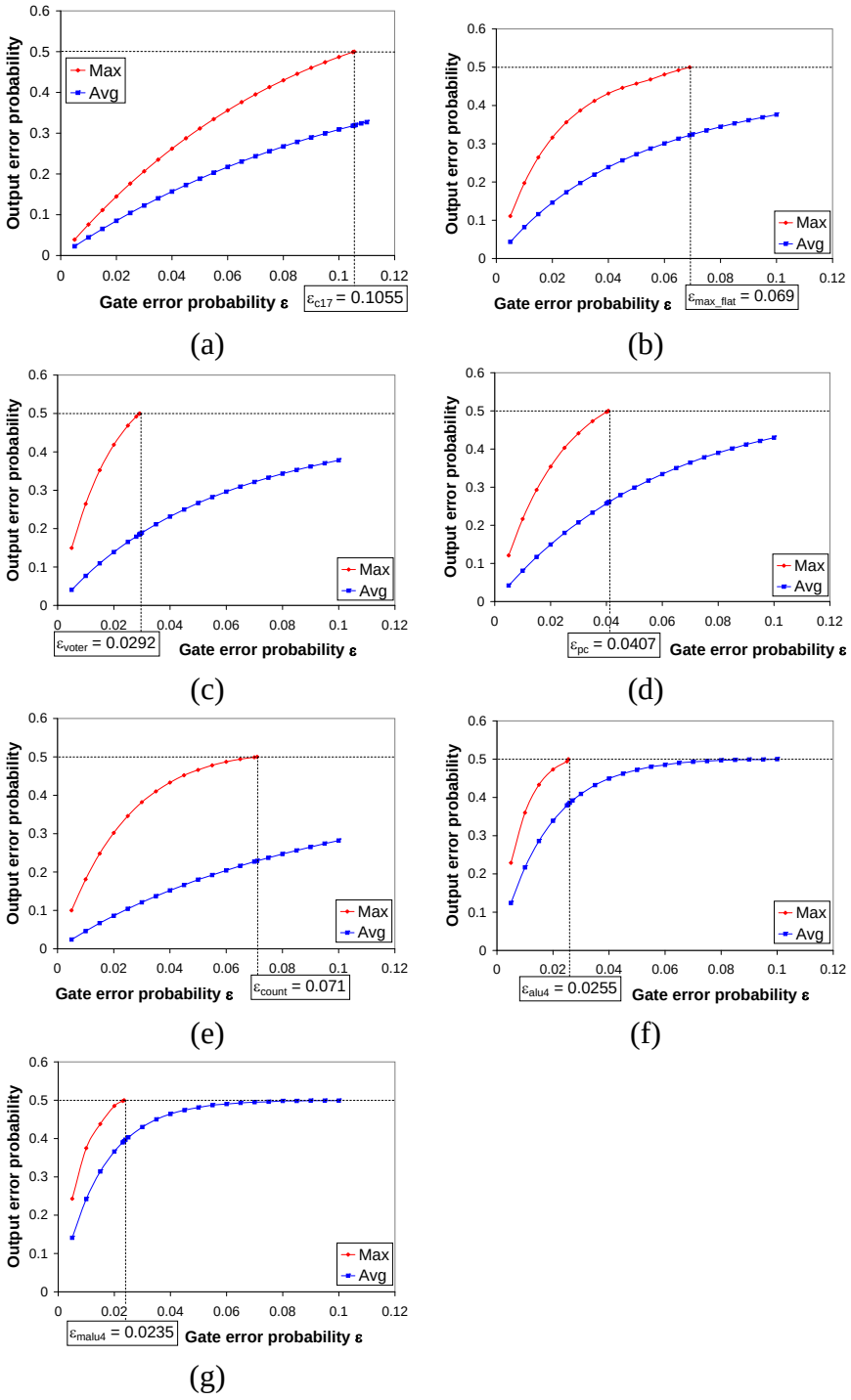


Figure 4.10. Circuit-specific error bound along with comparison between maximum and average output error probabilities for (a) $c17$, (b) max_flat, (c) $voter$, (d) pc, (e) $count$, (f) $alu4$, (g) $malu4$

Table 4.4. Run times for MAP computation

Circuit	No. of Inputs	No. of Gates	Time
c17	5	6	0.047s
max_flat	8	29	0.110s
voter	12	59	0.641s
pc	27	103	225.297s
count	35	144	36.610s
alu4	14	63	58.626s
malu4	14	92	588.702s

abilities for these circuits reach the hard bound for far lesser gate error probabilities ($\varepsilon = 0.0255 for\ alu4, \varepsilon = 0.0235 for\ malu4$).

- While the error bounds for all the circuits, except *c*17, are less than $0.08(8\%)$, the error bounds for circuits like *voter*, *alu*4 and *malu*4 are even less than $0.03(3\%)$ making them highly vulnerable to errors.

Table 4.4. tabulates the run time for MAP computation. The run time does not change significantly for different ε values and so we provide only one run time which corresponds to all ε values. This is expected as MAP complexity (discussed in Sec. 4.1.4) is determined by number of inputs, and number of variables in the largest clique which in turn depends on the circuit complexity. It has to be noted that, even though *pc* has less number of inputs than *count*, it takes much more time to perform MAP estimate due to its complex circuit structure.

4.2.4 Validation Using HSpice Simulator

Using external voltage sources error can be induced in any signal and it can be modeled using HSpice [43]. In our HSpice model we have induced error, using external voltage sources, in every gate's output. Consider signal O_f is the original error free output signal and the signal O_p is the error prone output signal and E is the *piecewise linear* (PWL) voltage source

Table 4.5. Comparison between maximum error probabilities achieved from the proposed model and the HSpice simulator at $\varepsilon = 0.05$

Circuit	Model	HSpice	% diff over HSpice
c17	0.312	0.315	0.95
max_flat	0.457	0.460	0.65
voter	0.573	0.570	0.53
pc	0.533	0.536	0.56
count	0.492	0.486	1.23
alu4	0.517	0.523	1.15
malu4	0.587	0.594	1.18

that induces error. The basic idea is that the signal O_p is dependent on the signal O_f and the voltage E. Any change of voltage in E will be reflected in O_p. If $E = 0v$, then $O_p = O_f$, and if $E = Vdd$ $(supply\ voltage)$, then $O_p \neq O_f$, thereby inducing error. The data points for the PWL voltage source E are provided by computations on a finite automata which models the underlying error prone circuit where individual gates have a gate error probability ε.

Note that, for an input vector of the given circuit, a single simulation run in HSpice is not enough to validate the results from our probabilistic model. Also the circuit has to be simulated for each and every possible input vectors to find out the worst-case one. For a given circuit, *the HSpice simulations are conducted for all possible input vectors, where for each vector the circuit is simulated for* $1\ million$ *runs and the comparator nodes are sampled.* From this data the maximum output error probability and the corresponding worst-case input vector are obtained.

Table 4.5. gives the comparison between maximum error probabilities achieved from the proposed model and the HSpice simulator at $\varepsilon = 0.05$. The notable results are as follows,

- The simulation results from HSpice almost exactly coincides with those of our error model for all circuits.
- The highest % difference of our error model over HSpice is just 1.23%.

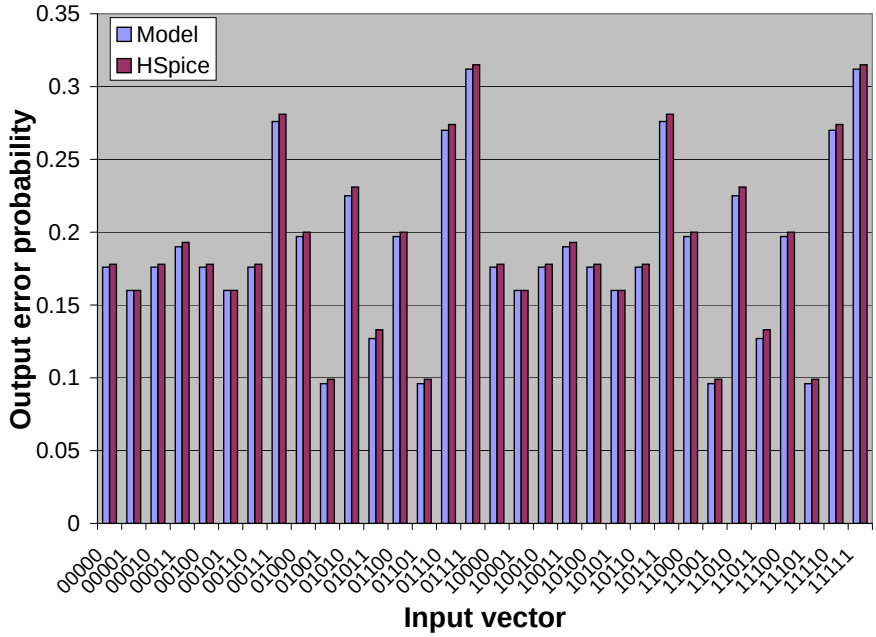


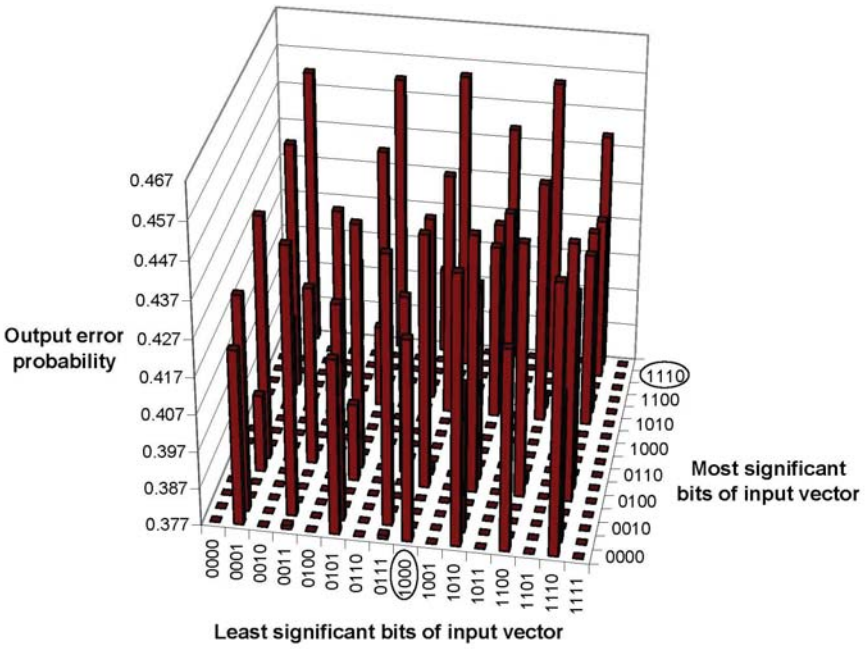
Figure 4.11. Output error probabilities for the entire input vector space with gate error probability $\varepsilon = 0.05$ for $c17$

Fig. 4.11. gives the output error probabilities for the entire input vector space of $c17$ with gate error probability $\varepsilon = 0.05$. The notable results are as follows,

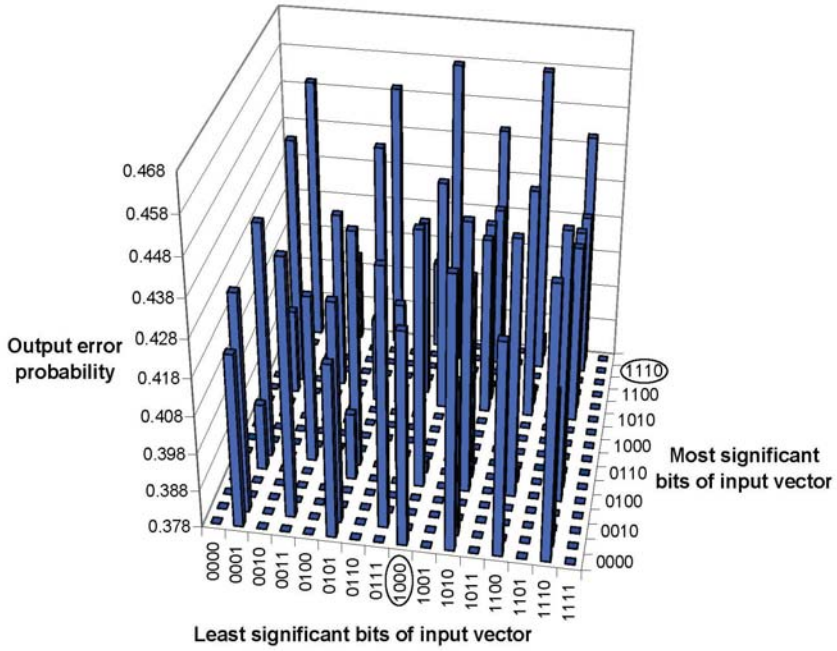
- It can be clearly seen that the results from *both* the probabilistic error model and HSpice simulations show that 01111 gives the maximum output error probability.

Fig. 4.12.(a) and (b) give the output error probabilities, obtained from the probabilistic error model and HSpice respectively, for max_flat with gate error probability $\varepsilon = 0.05$. In order to show that max_flat has large number of input vectors capable of generating maximum output error, we plot output error probabilities $\geq ((\mu) + (\sigma))$, where μ is the mean of output error probabilities and σ is the standard deviation. The notable results are as follows,

- It is clearly evident from Fig. 4.12.(a) that max_flat has a considerably large amount of input vectors capable of generating output error thereby making it error sensitive. Equivalent HSpice results from Fig. 4.12.(b) confirms this aspect.



(a)



(b)

Figure 4.12. (a) Output error probabilities $\geq (\mu+\sigma)$, calculated from probabilistic error model, with gate error probability $\varepsilon = 0.05$ for *max_flat* (b) Corresponding HSpice calculations

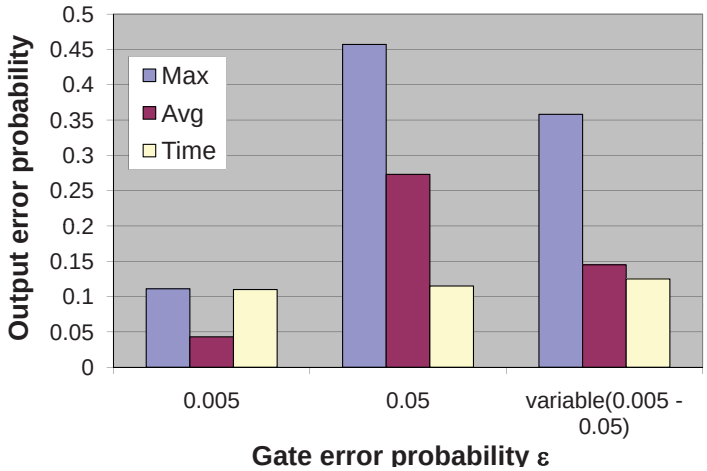


Figure 4.13. Comparison between the average and maximum output error probability and run time for ε=0.005, ε=0.05 and variable ε ranging from 0.005 - 0.05 for max_flat

- It is clearly evident that the results from probabilistic error model and HSpice show the same worst-case input vector, 11101000, that is obtained through MAP hypothesis.

4.2.5 Results with Multiple ε

Apart from incorporating a single gate error probability ε in all gates of the given circuit, our model also supports to incorporate different ε values for different gates in the given circuit. Ideally these ε values has to come from the device variabilities and manufacturing defects. Each gate in a circuit will have an ε value selected in random from a fixed range, say 0.005 - 0.05.

We have presented the result in Fig. 4.13. for max_flat. Here we compare the average and maximum output error probability and run time with ε=0.005, ε=0.05 and variable ε ranging from 0.005 - 0.05. The notable results are as follows,

- It can be seen that the output error probabilities for variable ε are closer to those for ε=0.05 than for ε=0.005 implicating that the outputs are affected more by the erroneous gates with ε=0.05.

- The run time for all the three cases are almost equal, thereby indicating the efficiency of our model.

4.3 Discussion

We have proposed a probabilistic model that computes the exact maximum output error probabilities for a logic circuit and mapped this problem as maximum *a posteriori* hypothesis of the underlying joint probability distribution function of the network. We have demonstrated our model with standard ISCAS and MCNC benchmarks and provided the maximum output error probability and the corresponding worst-case input vector. We have also studied the circuit-specific error bounds for fault-tolerant computing. The results clearly show that the error bounds are highly dependent on circuit structure and computation of maximum output error is essential to attain a tighter bound.

CHAPTER 5

MODELING ERROR IN SEQUENTIAL CIRCUITS

Sequential circuits consist of a combinational logic block, set of inputs, set of state bits where the values of the next state bit is fed back to the present state in the next clock cycle through latches. At a given time instance t_i, the state signals s_{t_i} are uniquely identified as a function of primary input signals i_{t_i} and state signals $s_{t_{i-1}}$ of the previous time instance giving rise to temporal correlations. Due to this, error occurring at one time instance might propagate towards several consecutive time instances making it more vulnerable.

In this chapter, we present a time evolving probabilistic model (Temporal Dependency Model TDM) that can handle the temporal effects of random variables. We form the TDM model (Fig. 5.1.(d)) by unrolling the basic probabilistic model into sufficiently large number of time slices and connecting the present state node of each time slice PS_{t_i} to the next state node of the previous time slice $NS_{t_{i-1}}$ thereby maintaining the temporal correlations.

To form the error model we have used the concept of miter circuits where two copies of the same circuit, one representing the ideal circuit and the other representing the erroneous circuit, are compared. For a given circuit, an ideal TDM model and an erroneous TDM model, where each gate is error-prone by a factor ε, are created. The ideal and erroneous primary output nodes, O_{t_i} and $O^e_{t_i}$ respectively, at each time slice t_i are connected to an XOR logic based comparator node C_{t_i} thereby forming a time evolving miter model. The output error probability is calculated by inferencing the error model and obtaining the probability of state "1" at the comparator nodes, $P(C_{t_i} = 1)$, at each time slice t_i iteratively by adding time slices until the results converge. The number of time slices needed for a given sequential circuit is

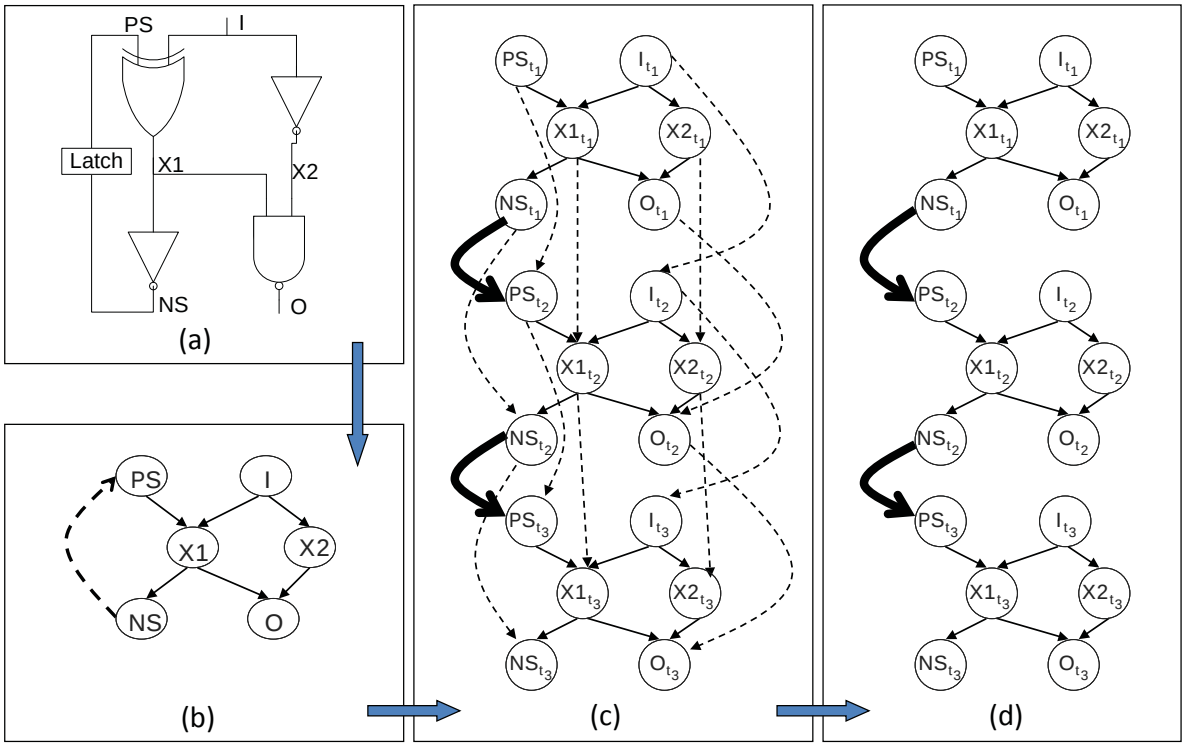


Figure 5.1. (a) Digital logic circuit (b) Corresponding probabilistic model (c) DAG representation which is not minimal (d) TDM model

related to the temporal dependence of output error which in turn is governed by the temporal correlations in the circuit. Our results show that different sequential circuits exhibit different degree of temporal dependence and the required amount of time slices is less than 10 for all the circuits, which is similar to the observations presented in [49].

5.1 Sequential Logic Model

We model the sequential circuits into a time evolved probabilistic network, named as temporal dependency model (TDM), which handles temporal dependencies. In this section we provide the details on the modeling of a sequential logic into a TDM model.

5.1.1 TDM Model

Let us consider the sequential circuit shown in Fig. 5.1.(a) where the present state node is represented as *PS*, the next state node is represented as *NS*, the primary input is represented as *I*, the primary output is represented as *O* and the internal nodes are represented as $X1$ and $X2$.

The equivalent probabilistic model shown in Fig. 5.1.(c) can be represented by $G_{t_i} = (V_{t_i}, E_{t_i})$. The nodes of the probabilistic model, V, are the union of all the nodes for each time slice.

$$V = \bigcup_{i=1}^{n} V_{t_i} \tag{5.1}$$

where n is the number of time slices. In our example $V_{t_i} = \{PS_{t_i}, NS_{t_i}, I_{t_i}, O_{t_i}, X1_{t_i}, X2_{t_i}\}$. The edges, E, of the probabilistic model are not just the union of the edges in a single time slice, E_{t_i}, but also includes the edges between time slices, that is, temporal edges, $E_{t_i,t_{i+1}}$. It has to be noted that the copies of the same variable X_i in all time slices follow a markov property such that the following two sets $\{X_{i,t_1}, \cdots, X_{i,t_{i-1}}\}$ and $\{X_{i,t_{i+1}}, \cdots, X_{i,t_{i+k}}\}$ are independent given X_{i,t_i}. For example, in Fig. 5.1.(c), $X1_{t_1}$ and $X1_{t_3}$ are independent of each other given $X1_{t_2}$. So the temporal edges can be defined as

$$E_{t_i,t_{i+1}} = \{(X_{i,t_i}, X_{i,t_{i+1}}) | X_{i,t_i} \in V_{t_i}, X_{i,t_{i+1}} \in V_{t_{i+1}}\} \tag{5.2}$$

where X_{i,t_i} is any node in time slice t_i and $X_{i,t_{i+1}}$ is the replica of the same node in the adjacent time slice t_{i+1} as shown in Fig. 5.1.(c). Thus, the complete set of edges E is

$$E = E_{t_1} \cup \bigcup_{i=2}^{n} (E_{t_i} + E_{t_{i-1},t_i}) \tag{5.3}$$

In the probabilistic model (Fig. 5.1.(c)), apart from the dependencies from one time slice, we also have the dependencies over two copies of the same variable X_j across adjacent time

slices. But it is evident that X_{j,t_i} and $X_{j,t_{i-1}}$ are independent of each other given the present state node PS_{j,t_i}. For example the nodes $X1_{t_1}$ and $X1_{t_2}$, from Fig. 5.1.(c), are independent of each other given the present state node PS_{j,t_2}; so even if we remove the temporal edges connecting these nodes at consecutive time slices the underlying structure will still be intact. The same can be told for $X1_{t_2}$ and $X1_{t_3}$.

So in the probabilistic model all the temporal edges except those connecting the present state and next state nodes of adjacent slices (bold lines in Fig. 5.1.(c)) can be removed to achieve a *minimal* representation as shown in Fig. 5.1.(d), which is termed as the *TDM model*. In our example, the necessary temporal edges can be given as,

$$E_{t_i,t_{i+1}} = \{(NS_{t_i}, PS_{t_{i+1}}) | NS_{t_i} \in V_{t_i}, PS_{t_{i+1}} \in V_{t_{i+1}}\} \tag{5.4}$$

5.2 Error Model

From the TDM model of a given sequential circuit, an error model is designed where the erroneous behavior of the circuit is compared with the ideal error-free behavior of the circuit.

5.2.1 Structure

The error model contains three sections, (i) *error-free logic* where the gates are ideal, (ii) *error-prone logic* where each gate goes wrong independently by an error probability ε and (iii) XOR based *comparator logic* that compare between the error-free and error-prone primary outputs. At first two copies of the TDM model, of the given sequential circuit, are created where one copy represents the error-free behavior of the circuit while the other represents erroneous behavior of the circuit. Fig. 5.2. illustrates the error model for the sequential circuit given in Fig. 5.1.(a). The *Error-free block* includes nodes representing the ideal combinational part of all the time slices. The *Error-prone block* includes nodes representing the erroneous combinational part of all the time slices. At each time slice t_k an XOR logic based node

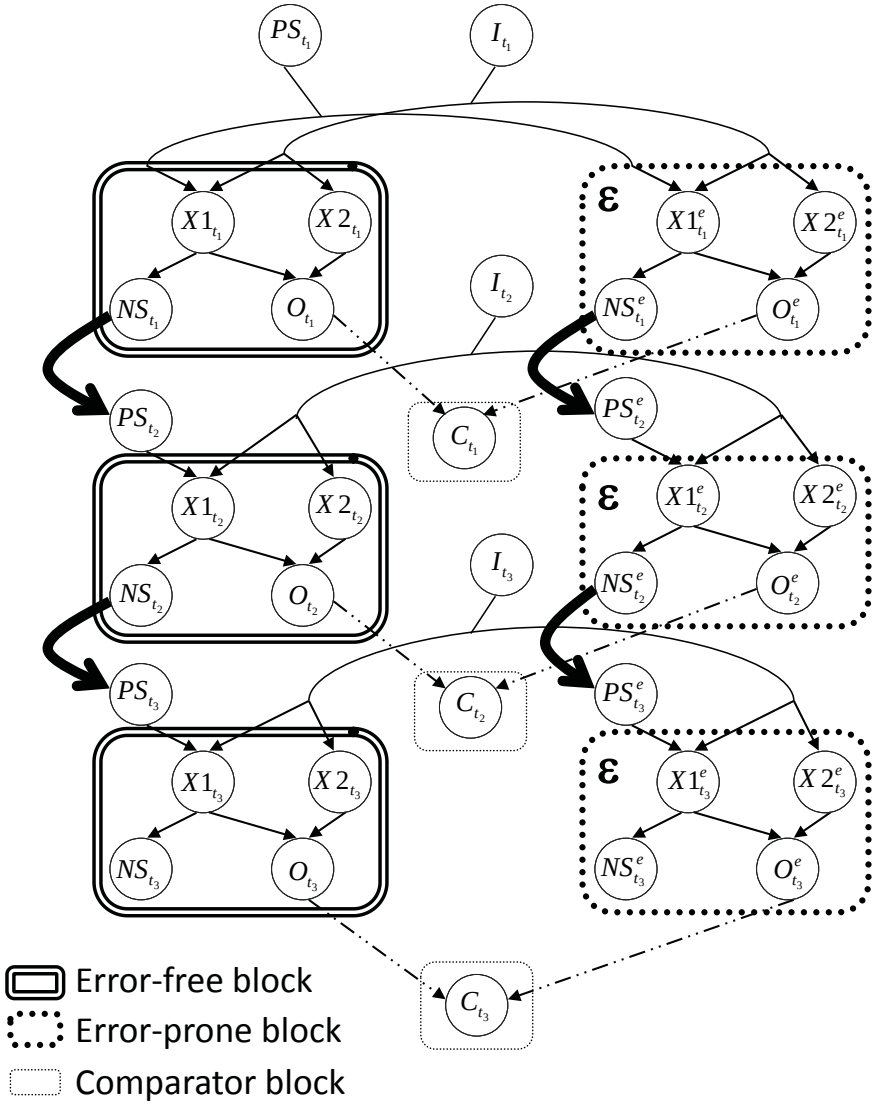


Figure 5.2. Error model obtained from TDM model with 3rd order temporal dependence

C_{t_k} is added to compare between the error-free and error-prone primary outputs O_{t_k} and $O^e_{t_k}$ respectively. These additional nodes are included in the *Comparator block*. Note that at every time slice t_k both error-free and error-prone logic has to be fed from the same primary input node I_{t_k} and at the first time slice t_1 both error-free and error-prone logic has to connect to the same present state (PS) node PS_{t_1}. Also the present state nodes, PS_{t_k} and $PS^e_{t_k}$, for all time slices t_k are error-free, since we assume ideal latches. The comparator nodes C_{t_k} and the primary input nodes I_{t_k} for all time slices t_k are also assumed to be error-free.

Any given probability function $P(x_1, x_2, \cdots, x_N)$ can be written as [1]

$$P(x_1, \cdots, x_N) = \prod_v P(x_v | Pa(X_v)) \tag{5.5}$$

where $Pa(X_v)$ are the parents of the variable X_v, representing its direct causes. This factoring of the joint probability function can be denoted as a graph with links directed from the random variable representing the inputs of a gate to the random variable representing the output. Our error model is one such graph structure where the probabilities $P(x_v|Pa(X_v)$ are provided by *Conditional Probability Tables* (CPTs) as shown in Table 5.1. It gives the CPTs for the nodes O_{t_k} whose parents are $X1_{t_k}$ and $X2_{t_k}$, and $O^e_{t_k}$ whose parents are $X1^e_{t_k}$ and $X2^e_{t_k}$ from Fig. 5.2. The nodes are governed by NAND logic.

The CPTs represent the underlying logic function of each gate. In this setup it is easier to incorporate the individual gate error probability ε by just changing the probabilities in the CPT. For example Table. 5.1. gives the CPTs for error-free O_{t_k} and error-prone $O^e_{t_k}$. In error-prone CPT we just have to replace the probability values 0 by ε and 1 by $1-\varepsilon$. This indicates that there is $(\varepsilon \times 100)$% chance for the signal to go to state "1" when it has to go to state "0" and $(\varepsilon \times 100)$% chance for the signal to go to state "0" when it has to go to state "1".

[1] Probability of the event $X_i = x_i$ will be denoted simply by $P(x_i)$ or by $P(X_i = x_i)$.

Table 5.1. Conditional probabilistic tables for error-free and error-prone NAND logic

Error-free NAND			
$P(X1_{t_k})$	$P(X2_{t_k})$	$P(O_{t_k}=0)$	$P(O_{t_k}=1)$
0	0	0	1
0	1	0	1
1	0	0	1
1	1	1	0
Error-prone NAND			
$P(X1^e_{t_k})$	$P(X2^e_{t_k})$	$P(O^e_{t_k}=0)$	$P(O^e_{t_k}=1)$
0	0	ε	1-ε
0	1	ε	1-ε
1	0	ε	1-ε
1	1	1-ε	ε

Table 5.2. Conditional probabilistic table for error-prone NAND logic having variable gate error probabilities, ε_0 and ε_1

Error-prone NAND			
$P(X1^e_{t_k})$	$P(X2^e_{t_k})$	$P(O^e_{t_k}=0)$	$P(O^e_{t_k}=1)$
0	0	ε_1	1-ε_1
0	1	ε_1	1-ε_1
1	0	ε_1	1-ε_1
1	1	1-ε_0	ε_0

Also, in our model we can provide unequal gate error probabilities for any variable $X^e_{t_k}$ at any time slice t_k, such that if $P(X_{t_k}=0)=1$, then $P(X^e_{t_k}=0)=1-\varepsilon_0$ and $P(X^e_{t_k}=1)=\varepsilon_0$; if $P(Xt_k=1)=1$, then $P(X^e_{t_k}=0)=\varepsilon_1$ and $P(X^e_{t_k}=1)=1-\varepsilon_1$. The corresponding CPT of this implementation for an error-prone NAND logic is given in Table. 5.2. ε_0 is basically the error probability of logic "0" and ε_1 is the error probability of logic "1" at the output of a gate. Increasing ε_0 indicates that the circuit has more $0 \rightarrow 1$ errors, whereas increasing ε_1 indicates that the circuit has more $1 \rightarrow 0$ errors. With this implementation, we can use our error model to study the effect of these errors in the output of the circuit.

5.2.2 Inference Scheme

The inference scheme basically calculates the joint probability distribution $P(x_1, \cdots, x_N)$ efficiently by propagating the probability distributions $P(x_v|Pa(X_v)$ of locally connected variables and thereby calculates the updated individual probability distributions of all random variables. The inference or propagation of belief on the probabilistic error model is done using the Hugin architecture [26, 27] which is an *exact* method. The inference on our model can be performed by forming clusters of nodes (*cliques*) which are directly dependent on each other and performing computations on those clusters, thereby enabling local computing. The network that is formed using these cliques is called *join tree*, where information can be propagated between cliques using message passing mechanism. Since extensive literature is already available, we will not be explaining the inference scheme in detail. Interested readers please refer to [26, 27].

In order to obtain a join tree, a *moral graph* is created from the error model, by adding undirected links between the parents of each common child node, and it is triangulated, to ensure that there are no cycles with more than three nodes, to obtain a *chordal graph*. Then the cliques are formed from the chordal graph and they are linked accordingly to form the join tree. Each adjacent cliques will have one or more common variables which are termed as *separators*. The following steps will explain the formation of join tree using an example circuit given in Fig. 5.3.(a) and its equivalent probabilistic model given in Fig. 5.3.(b).

- A *moral graph*, as shown in Fig. 5.3.(c), is formed from the original probabilistic network by adding undirected links between the parents of each common child node.
  - Additional links (Fig. 5.3.(c)): G1 - G2, G3 - G4
  - These additional links helps to form complete subgraphs of each parent-child set. The nodes in each subgraph can form a clique and thereby enable local computation. But this graphical form does not produce the minimal join tree because

some of the independencies represented by the probabilistic network are lost due to its undirected nature. The dependency structure is however preserved. This non-minimal representation will eventually lead to high computational needs, even when it does not sacrifice accuracy.

- To get a more minimal representation of the join tree which can capture the conditional independencies, a *chordal graph* is formed. It is obtained by triangulating the moral graph. Triangulation is the process of breaking all cycles in the graph to make a composition of cycles over just three nodes by adding additional links. To control the computational demands, the goal is to form a chordal graph with the minimum number of additional links.

  – Additional links (Fig. 5.3.(c)): No additional links since there are no cycles with more than three nodes.

- The cliques are formed from the chordal graph and they are linked accordingly to form the join tree (Fig. 5.3.(d)). Each adjacent cliques will have one or more common variables which are termed as *separators*. In Fig. 5.3.(d), between cliques $C1$ and $C2$, the variables $\{G3, G4\}$ form the separator set $S1$. Also, any two cliques sharing a set of common variables will have these common variables present in all the cliques that lie in the connecting path between these two cliques. In Fig. 5.3.(d), the cliques $C1$ and $C4$ share the common variable $\{G3\}$ and the only clique, $C2$, in their path also contains $\{G3\}$.

To perform local computation, each clique C_i is associated with probability potentials ϕ_{C_i} and each separator S_j is associated with probability potentials ϕ_{S_j}. Also from here on, the set of variables of any clique C_i or separator S_j will be represented in bold letters as $\mathbf{C_i}$ or $\mathbf{S_j}$. The inference is performed as follows,

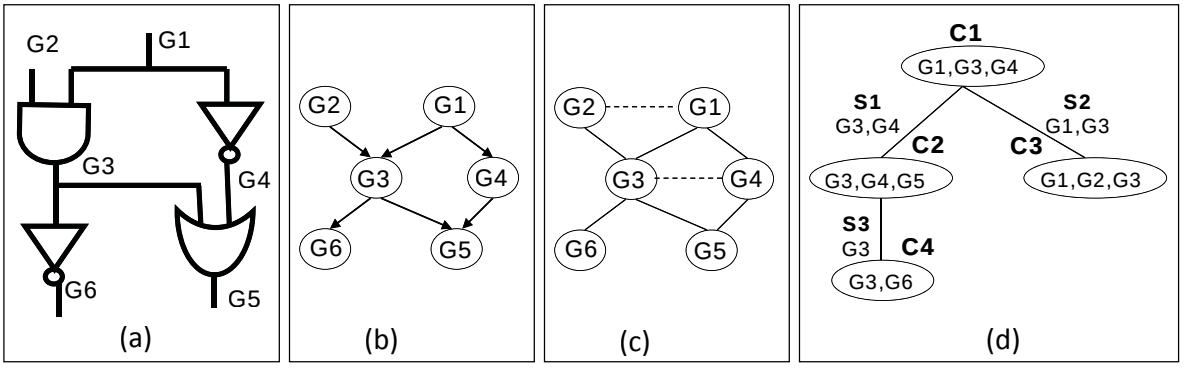


Figure 5.3. (a) Digital logic circuit (b) Corresponding probabilistic model (c) Moral graph obtained by adding undirected links between parents of common child nodes (d) Corresponding join tree obtained

- *Initialization*: Initially all the entries in the clique potentials and separator set potentials are assigned the value 1. In the join tree the variables of the given probabilistic network are divided in separate cliques. Each clique will have its own joint probability governed by its variables. But eventually we need to realize the joint probability of the entire network as given in Eqn. 3.2. To achieve this, for each variable Y_v, a particular clique C_i which contains Y_v along with its parents $Pa(Y_v)$ is selected and the conditional probability potential of Y_v from its CPT is multiplied to the clique potential ϕ_{C_i}

$$\phi_{C_i} = \phi_{C_i} P(y_v | Pa(Y_v)) \tag{5.6}$$

  - *Example*: (Fig. 5.3.(d))

$$\phi_{C3} = \phi_{C3} P(G3 | G1, G2) \tag{5.7}$$

- *Message passing*: After initialization the clique potentials are not consistent with their separator potentials. So the joint probability given in Eqn. 3.2 is not perfectly realized. To achieve this consistency *message passing* is performed. At first the marginal prob-

ability of the separator variables has to be computed from the probability potential of clique C_p and then it is used to scale the probability potential of clique C_q.

- *Marginalization*:

$$\phi_{S_r}^{updated} = \sum_{\mathbf{C_p} \setminus \mathbf{S_r}} \phi_{C_p} \tag{5.8}$$

- *Scaling*:

$$\phi_{C_q} = \phi_{C_q} \frac{\phi_{S_r}^{updated}}{\phi_{S_r}} \tag{5.9}$$

- *Example*: (Fig. 5.3.(d)) Message passing from $C2$ to $C1$

$$\phi_{S1}^{updated} = \sum_{G5} \phi_{C2} \tag{5.10}$$

$$\phi_{C1} = \phi_{C1} \frac{\phi_{S1}^{updated}}{\phi_{S1}} \tag{5.11}$$

- The transmission of this scaling factor is the primary necessity for updating and message passing. Eventually the joint probability of the entire network can be represented as,

$$P(y_1, \cdots, y_N) = \frac{\prod_i \phi_{C_i}}{\prod_j \phi S_j} \tag{5.12}$$

- Message passing in a join tree has to be done in both directions, from root to leaf termed as *outward pass* and from leaf to root termed as *inward pass*. An inward pass followed by an outward pass will completely update all the cliques in the join tree.

- *Individual probability distribution calculation*: Then the individual probability distribution for each variable can be calculated by choosing a clique C_i containing the variable Y_v and marginalizing its potential ϕ_{C_i} over all the other variables $\mathbf{C_i} \setminus Y_v$. This probability

distribution $P(y_v)$ is given as,

$$P(y_v) = \sum_{\mathbf{C_i} \setminus Y_v} \phi_{C_i} \tag{5.13}$$

– *Example*: (Fig. 5.3.(d))

$$P(G6) = \sum_{G3} \phi_{C4} \tag{5.14}$$

5.2.3 Output Error Probability

The output error probability of a given sequential circuit can be obtained by calculating the probability, $P(C_{t_n} = 1)$ of the comparator node C_{t_n} at the final time slice t_n by inferencing the corresponding error model. Each sequential circuit based on its underlying structure will need different amount of time slices. During inference if at any time instance a random variable representing a signal in the error-prone logic picks up a wrong value, this value will propagate for a considerable amount of time before the signal gets back to its original value. This pattern will keep on repeating through several samples. Due to this phenomenon the random variable takes some time to converge at one particular probability distribution. So for each sequential circuit we have to iteratively calculate the output error probability by increasing the time slices and stop when the output error probabilities of consecutive time slices converge. This is the reason for having comparator nodes at every time slice. The number of time slices needed by a sequential circuit is purely dependent on the underlying functionality and circuit structure.

5.3 Experimental Results

The output error probabilities for various sequential circuits are calculated using our experimental setup. We have performed our experiments on standard MCNC and ISCAS benchmark circuits. We have used HUGIN tool [50] to perform inference on the error model and we validate these results with equivalent HSpice simulation.

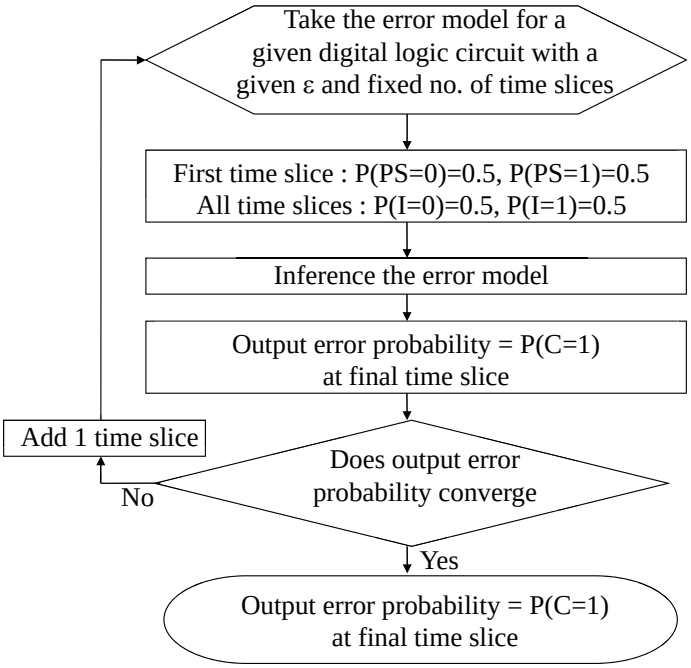


Figure 5.4. Flowchart for experimental procedure

5.3.1 Experimental Procedure

Fig. 5.4. gives the experimental procedure undertaken to obtain the output error probabilities. At first for a given ε value the probabilistic error model is obtained. The primary input nodes I_{t_k} for all the time slices t_k and the present state nodes PS_{t_1} for the first time slice t_1 are set to be equally probable to have state ”0” or state ”1”. The model is then inferenced and the output error probability is obtained by noting the probability of state ”1” at the comparator node, $P(C_{t_k} = 1)$ at every time slice t_k. This inference is an *exact* one and it also handles reconvergence and spatio-temporal dependencies. $P(C_{t_n} = 1)$ of the final time slice t_n and $P(C_{t_{n-1}} = 1)$ of the previous time slice t_{n-1} are checked for convergence. If they do not converge the time slices at both error-free and error-prone blocks are increased by 1 and the procedure is repeated. Thus the circuits are inferenced with different number of time slices

Table 5.3. Output error probabilities at $\varepsilon = 0.001, 0.003, 0.005, 0.01$

Circuits	$\varepsilon = 0.001$	$\varepsilon = 0.003$	$\varepsilon = 0.005$	$\varepsilon = 0.01$
train11	0.0055	0.0161	0.0265	0.0511
lion	0.0060	0.0177	0.0288	0.0545
lion9	0.0069	0.0200	0.0326	0.0614
bbara	0.0074	0.0213	0.0341	0.0621
bbtas	0.0072	0.0211	0.0344	0.0653
s27	0.0075	0.0220	0.0357	0.0676
mc	0.0084	0.0246	0.0399	0.0747

iteratively and stopped when the output error probability values converge at consecutive time slices.

5.3.2 Output Error Probabilities

Table 5.3. gives the output error probabilities for gate error probabilities, ε = 0.001, 0.003, 0.005, 0.01. For a slight increase in ε value from 0.001 to 0.003, there is at least 2.87 fold increase in the corresponding output error probabilities. Also, for a considerably low influx of error at the gates for $\varepsilon = 0.005(0.5\%)$, the output error probability of most of the circuits exceed 3% with *mc* producing the highest output error probability of 3.99% which is almost 8 fold higher than the individual gate error probability. The same can be seen for $\varepsilon = 0.01(1\%)$, where the output error probability of most of the circuits exceed 6%.

5.3.3 Number of Time Slices

Fig. 5.5. shows the number of time slices needed by *bbara* and *bbtas* for $\varepsilon = 0 - 0.006$. It can be seen that the circuits needed less number of time slices for small ε values and then the needed number of time slices gradually increases along with ε value. For *bbtas*, the needed number of time slices gets set to 5 at $\varepsilon = 0.0013$ while *bbtas* takes up more time slices and

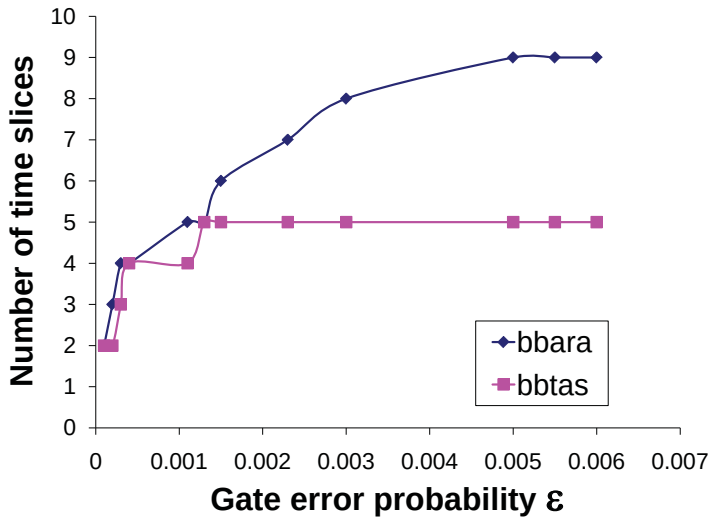
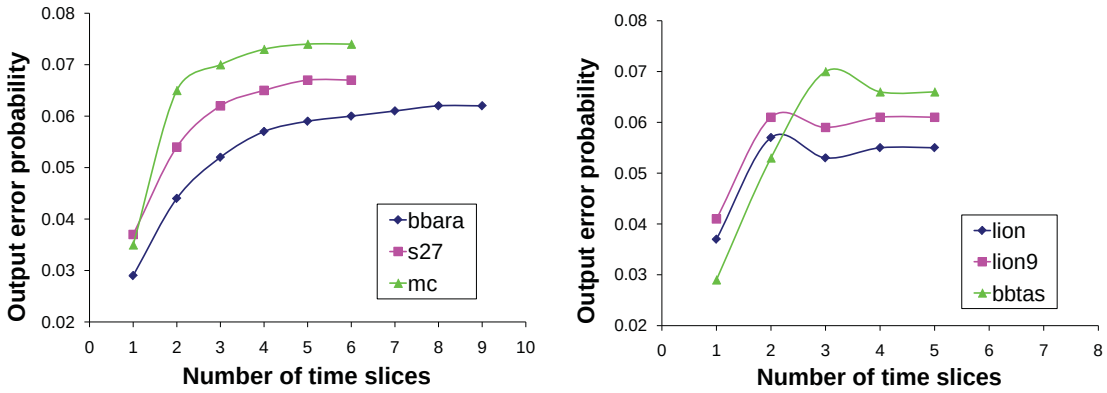


Figure 5.5. Number of time slices needed by *bbara* and *bbtas* for $\varepsilon = 0 - 0.006$

finally gets set at 9 for $\varepsilon = 0.005$. The number of time slices needed is completely dependent on the circuit structure. The following studies will shed more light on this aspect.

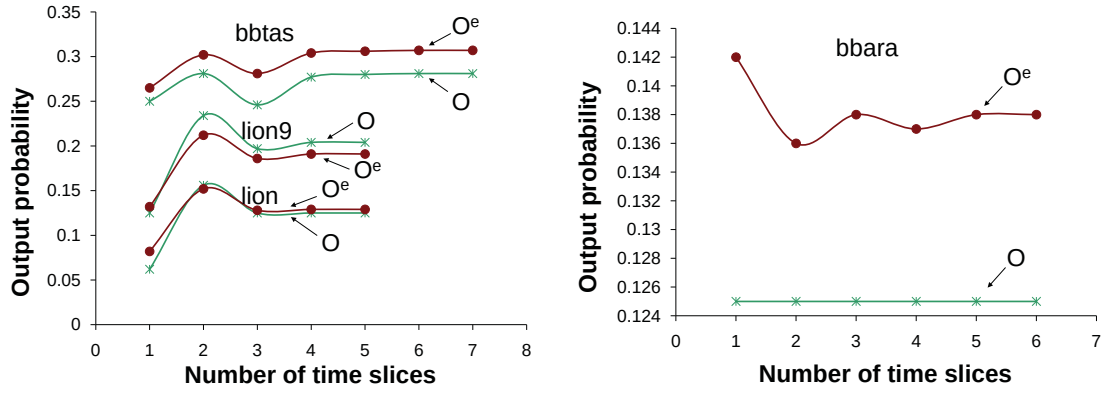
5.3.4 Output Error Propagation Across Time Slices

Fig. 5.6.(a) & (b) gives the transition of output error probability across time slices for $\varepsilon = 0.01$. Here we show two sets of results that shows the difference in the transition of output error across time slices. Fig. 5.6.(a) shows the output error transition for *bbara*, *s*27 and *mc*, where the output error increases gradually across time slices and finally gets converged. Whereas in Fig. 5.6.(b), which shows the output error transition for *lion*, *lion*9 and *bbtas*, the output error reaches a maximum value and then gradually gets back to a steady value. This behavior can be attributed to the relation between the present state nodes and the primary input nodes which have random unbiased state distribution. If the present state nodes are closely connected to the input nodes resulting in having an unbiased state distribution, the output error will not be significant. With a biased state distribution in the present state nodes, the output



(a) (b)

Figure 5.6. (a) Transition of output error probability across time slices for *bbara*, *s*27 and *mc* with $\varepsilon = 0.01$ (b) Transition of output error probability across time slices for *lion*, *lion*9 and *bbtas* with $\varepsilon = 0.01$



(a) (b)

Figure 5.7. (a) Transition of error-free and error-prone output probabilities across time slices for *bbtas*, *lion*9 and *lion* with $\varepsilon = 0.01$ (b) Transition of error-free and error-prone output probabilities across time slices for *bbara* with $\varepsilon = 0.01$

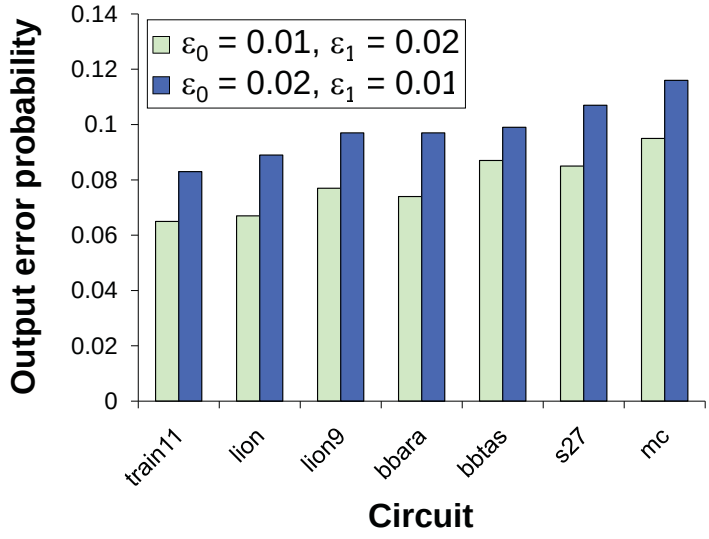


Figure 5.8. Output error probabilities for ($\varepsilon_0 = 0.01$, $\varepsilon_1 = 0.02$) and ($\varepsilon_0 = 0.02$, $\varepsilon_1 = 0.01$)

error becomes more significant and reaches a maximum value in the early time slices as shown in Fig. 5.6.(b).

Fig. 5.7.(a) & (b) gives the transition of error-free (O) and error-prone (O^e) output probabilities across time slices for $\varepsilon = 0.01$. The results in Fig. 5.7.(a) show that, for some circuits, the temporal dependence of the erroneous output conforms with that of the ideal error-free output. Whereas in circuits like *bbara* this is not the case as shown in Fig. 5.7.(b). Due to this, the output error probability in *bbara* takes more time to converge.

5.3.5 Output Error Probabilities for $\varepsilon_0 \neq \varepsilon_1$

In our model we can provide unequal gate error probability values, ε_0 and ε_1, to study the effect of $0 \rightarrow 1$ and $1 \rightarrow 0$ errors on the output of a circuit. Fig. 5.8. gives the output error probabilities for ($\varepsilon_0 = 0.01$, $\varepsilon_1 = 0.02$) and ($\varepsilon_0 = 0.02$, $\varepsilon_1 = 0.01$). It can be clearly seen that when $\varepsilon_0 > \varepsilon_1$ the output error probabilities are higher for all circuits. This indicates that a $0 \rightarrow 1$ error can make the outputs more erroneous and signals that stay at logic ”0” more often can be vulnerable to this effect. This might be favorable in CMOS technology, since the $0 \rightarrow 1$

Table 5.4. Output error probabilities at $\varepsilon = 0.001, 0.003, 0.005, 0.01$ compared with HSpice simulation results

	$\varepsilon = 0.001$			$\varepsilon = 0.003$		
Circuits	Error model	HSpice	% diff	Error model	HSpice	% diff
train11	0.0055	0.0057	3.51	0.0161	0.0159	1.26
lion	0.0060	0.0063	4.76	0.0177	0.0171	3.51
lion9	0.0069	0.0066	4.55	0.0200	0.0208	3.85
bbara	0.0074	0.0070	5.71	0.0213	0.0208	2.40
bbtas	0.0072	0.0069	4.35	0.0211	0.0203	3.94
s27	0.0075	0.0080	6.25	0.0220	0.0217	1.38
mc	0.0084	0.0088	4.55	0.0246	0.0250	1.60
	$\varepsilon = 0.005$			$\varepsilon = 0.01$		
Circuits	Error model	HSpice	% diff	Error model	HSpice	% diff
train11	0.0265	0.0263	0.76	0.0511	0.0497	2.82
lion	0.0288	0.0277	3.97	0.0545	0.0522	4.41
lion9	0.0326	0.0339	3.83	0.0614	0.0607	1.15
bbara	0.0341	0.0345	1.16	0.0621	0.0595	4.37
bbtas	0.0344	0.0354	2.82	0.0653	0.0671	2.68
s27	0.0357	0.0345	3.48	0.0676	0.0638	5.96
mc	0.0399	0.0391	2.05	0.0747	0.0733	1.91

bit flip is much harder than $1 \rightarrow 0$ bit flip due to the less error proneness of pMOS as compared to nMOS since holes are tougher to be dislodged by external particle bombardments. It can also signify that circuits with series pMOS connections can be less error prone as compared to circuits with parallel pMOS connections.

5.3.6 Validation Using HSpice Simulation

We validate our results by comparing them with HSpice simulation results. Even though, our error model can be used for any technology, the lack of benchmark circuits in any of the other emerging technologies has forced us to compare our model with simulations using 45nm CMOS technology. Using external voltage sources error can be induced in any signal

and it can be modeled using HSpice [43]. In our HSpice model we have induced error, using external voltage sources, in every gate's output. Consider signal O_f is the original error free output signal and the signal O_p is the error prone output signal and E is the *piecewise linear* (PWL) voltage source that induces error. The basic idea is that the signal O_p is dependent on the signal O_f and the voltage E. Any change of voltage in E will be reflected in O_p. If $E = 0v$, then $O_p = O_f$, and if $E = Vdd(supply\ voltage)$, then $O_p \neq O_f$, thereby inducing error. The data points for the PWL voltage source E are provided by computations on a finite automata which incorporates the individual gate error probability ε. The width of every error pulse is fixed to $1ns$. The results are obtained by running the circuits for 5 *million* random input vectors and sampling the comparator outputs. Table 5.4. gives the comparison between the output error probabilities obtained from inference in the error model and Hspice simulation for different circuits with gate error probability $\varepsilon = 0.001, 0.003, 0.005, 0.01$. The % difference is calculated as, ((Error model - Hspice) / Hspice) x 100. The highest relative difference between the inference results and HSpice results is just 6.25% and on an average the relative difference is only 4.43%.

5.4 Discussion

We have proposed a compact probabilistic model that can handle error in sequential logic and we have presented experimental results on ISCAS and MCNC benchmark circuits. We have observed that for low gate error probabilities like $\varepsilon = 0.005(0.5\%)$, the output error probabilities are at least 5 fold higher and at most 8 fold higher. Also, our observations showed that the degree of temporal dependence differs for various sequential circuits. Another interesting observation indicated that $0 \rightarrow 1$ errors affects the circuit output more than $1 \rightarrow 0$ errors. We have also validated our model using HSpice simulation results and the average % difference is only 4.43%.

CHAPTER 6

REDUNDANCY SCHEMES FOR ERROR MITIGATION

Reliable computing using unreliable circuit elements can be accomplished using the concept of redundancy. As the name suggests, the basic idea of 'redundancy' is based on analyzing any given erroneous circuit through multiple redundant components or processes. The outputs from these redundant components or processes are subjected to a voting scheme where the majority value of the signal under consideration is chosen to be its ultimate error-free value. In this chapter, the following unique redundancy schemes are discussed.

- Temporal redundancy scheme - the redundancy is applied in the input space by providing multiple instances of the same input combinations, which are highly probable to create an error in the output.
- Spatial redundancy scheme - the redundancy is applied in the intermediate signal space by providing multiple copies of the same gates, whose output signals are erroneous.
- Hybrid redundancy scheme - the redundancy is applied in both input space and intermediate signal space, in order to achieve comprehensive error reduction in the given erroneous circuit.

6.1 Temporal Redundancy Scheme Using Triple Temporal Redundancy (TTR) Technique

Triple Temporal Redundancy (TTR) is an error reduction technique, where specific input combinations are applied *three* times and from the resulting simulation outputs the majority value is accepted as the correct one. Performing this technique on the entire input space will result a large amount of unwanted calculations leading to high simulation time. In order to make it more efficient, the technique has to be applied only on a subset of the input space, where each input combination has a high chance of giving a wrong output compared to those in the rest of the input space. So as a preliminary step for this redundancy technique, the above mentioned subset on the input space should be determined. This can be achieved by using the model that calculates the maximum output error and the corresponding worst-case input combination, explained in Chapter 4.

6.1.1 Determination of the Set of Worst-Case Input Combinations for Selective Redundancy in TTR

As discussed in Chapter 4, the model that calculates maximum output error probability, determines a single worst-case input combination which has the highest probability to provide an error in the output. In order to get a set of worst-case input combinations, the search process explained in Section 4.1.2 can be extended as follows (Fig. 6.1.),

- After obtaining the most probable worst-case input combination, $\mathbf{i}_{MAP}$, the corresponding MAP probability, $MAP(\mathbf{i}_{MAP}, \mathbf{o})$, is noted.

- The number of input combinations needed for TTR is decided and a lower bound of $MAP(\mathbf{i}, \mathbf{o})$, called LB_MAP, with reference to $MAP(\mathbf{i}_{MAP}, \mathbf{o})$, is chosen in order to collect the set of worst-case input combinations. This lower bound can be adjusted based on the amount of input combinations needed for TTR.

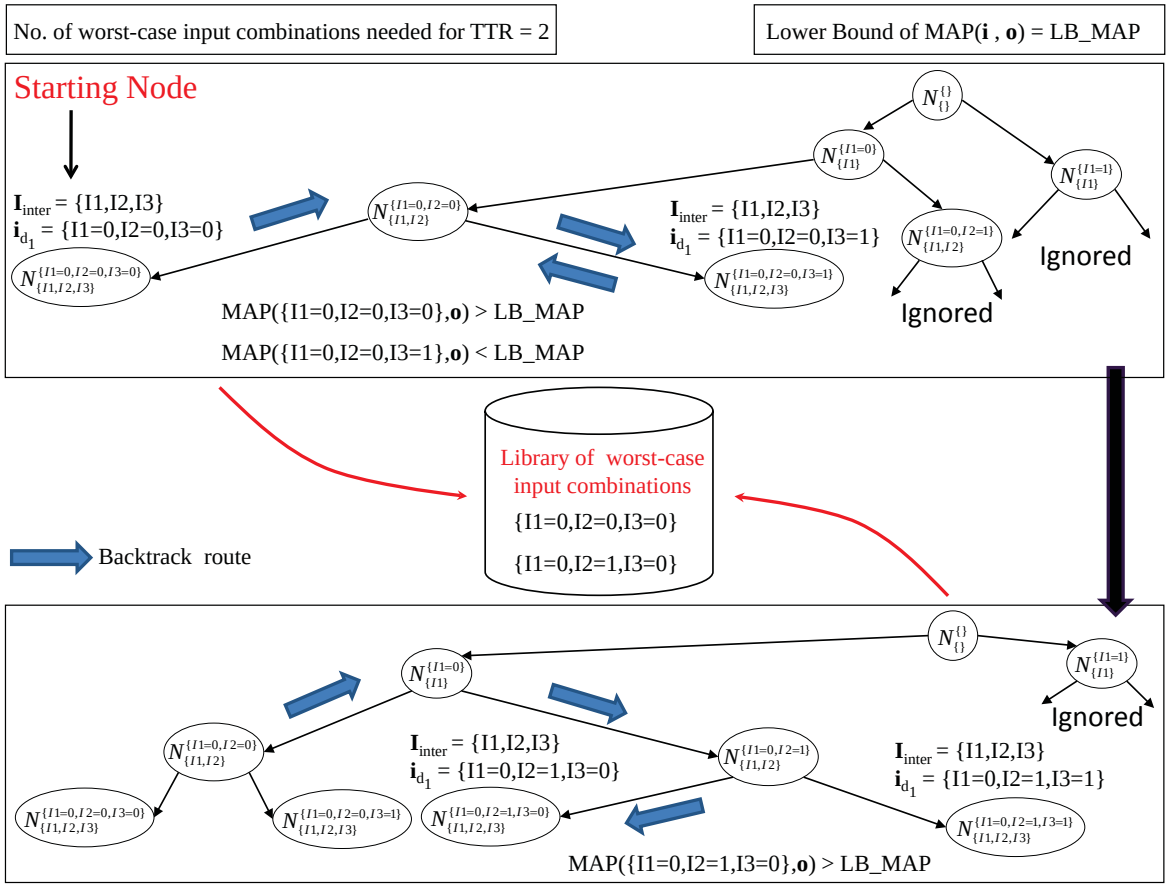


Figure 6.1. Determination of the set of worst-case input combinations by backtracking through the search tree used for MAP computation given in Fig. 4.8.

- The search process for the rest of the worst-case input combinations apart from $\mathbf{i}_{MAP}$, starts from the node $N_{\mathbf{I}}^{\mathbf{i}_{MAP}}$ and backtracks through the depth-first branch and bound search tree, collecting all the input combinations which have $MAP(\mathbf{i}, \mathbf{o})$ above the chosen lower bound. This search process is stopped once the target number of worst-case input combinations for TTR is reached.

- Finally, the resulting set of worst-case input combinations are placed in a library which can be used as a reference while performing TTR.

Note that the search process for MAP computation is conducted in a binary tree, where the root node N and the intermediate nodes $N^{\mathbf{i}_{inter}}_{\mathbf{I}_{inter}}$ are connected to only two child nodes. In the process for determining the set of worst-case input combinations, the basic idea is that every input combination $\mathbf{i}$ in the vicinity of $\mathbf{i}_{MAP}$ should be checked for the possibility of being a worst-case one by comparing the corresponding joint probability $MAP(\mathbf{i},\mathbf{o})$ with the lower bound *LB_MAP*. Also, note that whenever an intermediate node, with two unvisited child nodes, is encountered, the search path goes through the '0' edge first and then to the '1' edge.

In the example given in Fig. 6.1., the backtracking starts from node $N^{\{I1=0,I2=0,I3=0\}}_{\{I1,I2,I3\}}$ and goes to the intermediate node $N^{\{I1=0,I2=0\}}_{\{I1,I2\}}$ and evaluates the node $N^{\{I1=0,I2=0,I3=1\}}_{\{I1,I2,I3\}}$ to check whether the corresponding condition $MAP(\{I1=0,I2=0,I3=1\},\mathbf{o}) > LB\_MAP$ is true. Since it is not true, the input combination $\{I1=0,I2=0,I3=1\}$ is not categorized as a worst-case input combination. Then the backtracking search goes one level above to node $N^{\{I1=0\}}_{\{I1\}}$ and gets to its unvisited child node $N^{\{I1=0,I2=1\}}_{\{I1,I2\}}$, which has two unvisited child nodes. The child node along the '0' edge, $N^{\{I1=0,I2=1,I3=0\}}_{\{I1,I2,I3\}}$, is visited first and the corresponding joint probability, $MAP(\{I1=0,I2=1,I3=0\},\mathbf{o})$, is checked for the condition $MAP(\{I1=0,I2=1,I3=0\},\mathbf{o}) > LB\_MAP$. Since it is true, the input combination $\{I1=0,I2=0,I3=1\}$ is considered as a worst-one and added to the library. Since the target of 2 worst-case input combinations for TTR is reached, the search is stopped.

6.1.2 Experimental Setup for TTR

In order to achieve efficient computation, the unnecessary simulation runs are avoided by incorporating *selective redundancy* in TTR. This is performed through the following steps,

- Performing TTR only on the worst-case input combinations instead of running it on the entire input space.
- Deciding on the third run of TTR based on the first two runs.

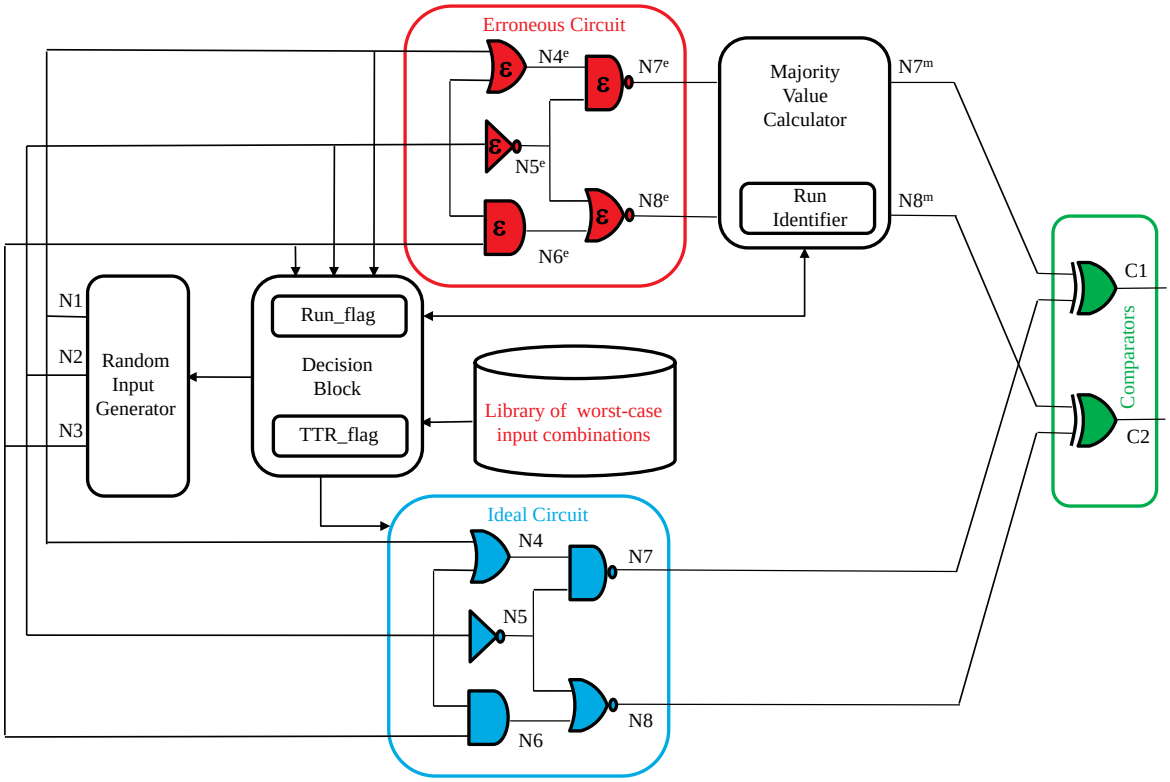


Figure 6.2. Experimental setup for TTR incorporating selective redundancy

Fig. 6.2. illustrates the experimental setup for TTR. The description of the various segments are as follows,

- Erroneous Circuit:- This is the circuit under consideration, where each gate has a gate error probability ε.
- Ideal Circuit:- The fictitious ideal counterpart of the erroneous circuit, under consideration, used to study the erroneous primary output signals.
- Comparators:- XOR gates used to compare between the erroneous primary output signals and their ideal counterparts, in order to detect the occurrence of output error.

- Library of worst-case input combinations:- Collection of input combinations, which have high probability of inducing error in the circuit output as compared to the rest of the input space.

- Random Input Generator:- This segment produces random digital input signals which are applied to the primary inputs of both the erroneous circuit and its ideal counterpart.

- Decision Block:- This segment decides on the necessity of performing TTR and the necessity of the third run. It gets the needed information from the library of worst-case input combinations, the random input generator and the majority value calculator. It also sends the decision information to the random input generator and the majority value calculator.

- TTR_flag:- A boolean flag that triggers the necessity of performing TTR.
  - $TTR\_flag = 1$, implies TTR needed.
  - $TTR\_flag = 0$, implies TTR not needed.

- Run_flag:- A boolean flag that triggers the necessity of performing the third run.
  - $Run\_flag = 1$, implies third run is needed.
  - $Run\_flag = 0$, implies third run is not needed.

- Majority Value Calculator:- This segment compares the output values from the TTR runs and determines the majority value of the primary output signals.

- Run Identifier:- Compares the output values from the first two TTR runs and sends the information to the decision block.

The procedure to perform TTR is as follows,

- An input combination generated from the random input generator is provided to both the erroneous circuit and the ideal circuit. The same input combination is also sent to the decision block.
- The generated input combination is compared with the set of worst-case input combinations. If it conforms with any of the worst-case input combination, then $TTR\_flag = 1$, else $TTR\_flag = 0$.
- TTR_flag is checked for its status.
  - If $TTR\_flag = 1$, then the same input combination is applied again and the value is compared with that of the previous run. Then *Run_flag* is triggered.
  - If $TTR\_flag = 0$, then the next input combination is applied. *Run_flag* is not triggered.
- Run_flag is checked for its status.
  - If $Run\_flag = 1$, then the input combination is applied for the third time. The majority value from the three runs is determined.
  - If $Run\_flag = 0$, then the input combination is not applied for the third time. The value from the second run is decided as the majority value.
- Then the majority value of the erroneous output signal and the ideal output signal are fed to an XOR comparator.
- The output error probabilities are calculated from the comparator outputs.

Note that the decision block can run in parallel with the circuit and so when the inputs are not from the library, there is no penalty for the library search. Also note that if it is decided

that TTR is needed, then the XOR comparators wait for the majority values to be determined before comparing the erroneous and ideal signals, thereby avoiding the occurrence of unnecessary samples in their output signals. Also in the example given in Fig. 6.2., if TTR is not performed, then $N7^m = N7^e$, $N8^m = N8^e$. If TTR is performed, then $N7^m$ = majority of $N7^e$ from the TTR runs, $N8^m$ = majority of $N8^e$ from the TTR runs.

6.2 Spatial Redundancy Scheme Using Cascaded Triple Modular Redundancy (CTMR) Technique

In triple modular redundancy, three copies of the any erroneous gate are created and from their outputs the majority value is accepted as the correct one. Lets say that the three copies of the erroneous digital signal are represented as A, B and C. Then the majority value out of A, B, C can be determined by implementing the boolean function $AB + BC + AC$. The most important aspect to note here is that the error probability of any erroneous gate will reduce when subjected to triple modular redundancy through the majority gate. An even better error probability can be achieved using CTMR, where two cascading levels of triple modular redundancy is applied by replicating the erroneous gate nine times and producing three majority outputs which in turn are supplied to another majority gate to get a final value. Fig. 6.3. illustrates the CTMR technique used to perform spatial redundancy. The signal $N8^e$ whose initial error probability is ε, when subjected under CTMR, attains a better error probability $\varepsilon_s < \varepsilon$. To achieve efficient spatial redundancy, instead of applying CTMR to all gates in the circuit, only some selective gates can be chosen. To determine these gates, a sensitivity analysis can be performed on the corresponding probabilistic error model of the circuit.

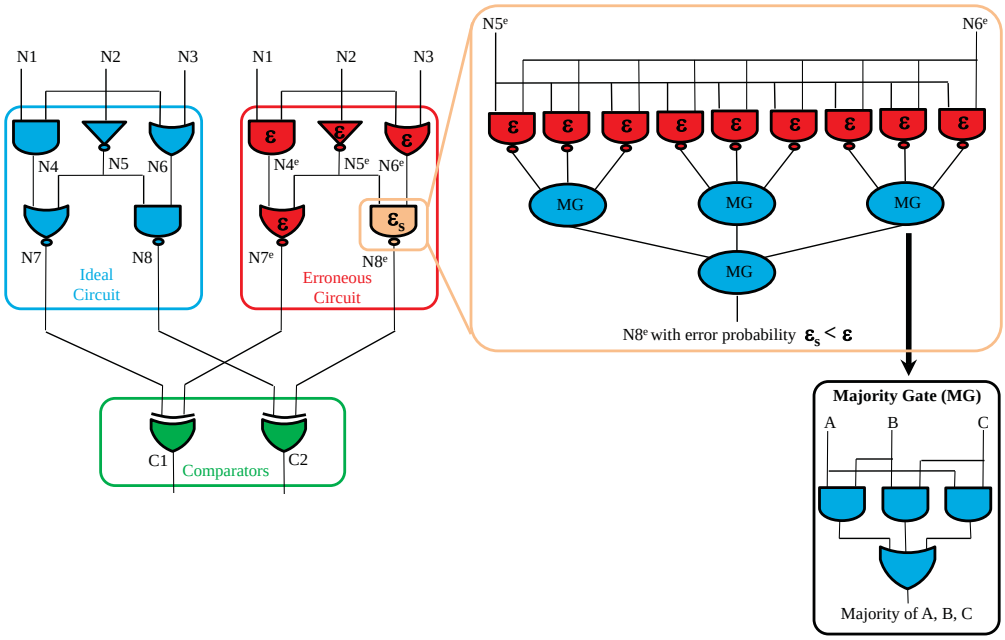


Figure 6.3. Spatial redundancy scheme using CTMR technique incorporating majority logic

6.2.1 Sensitivity Analysis for Selective Redundancy in CTMR

To determine the sensitivity of an erroneous node, that node is perturbed so that it will have a different gate error probability ε_s, which can be similar to the one obtained by performing CTMR on that particular node, while providing gate error probability ε for all other nodes. The output error probabilities for this setup is calculated as, $P_{\varepsilon_s}(O_i)$. Then the output error probabilities, $P_{\varepsilon}(O_i)$, are obtained with gate error probability ε fixed in all the erroneous nodes. The difference between the output error probabilities, $P_{\varepsilon_s}(O_i)$ and $P_{\varepsilon}(O_i)$, determines the node's degree of influence or sensitivity. Nodes are ranked on the basis of the decreasing order of the degree of influence and the top ranked nodes are selected as the *sensitive nodes*.

6.3 Hybrid Redundancy

Hybrid redundancy scheme is the blend of temporal redundancy and spatial redundancy. As shown in Fig. 6.4., hybrid redundancy can be visualized as performing temporal redun-

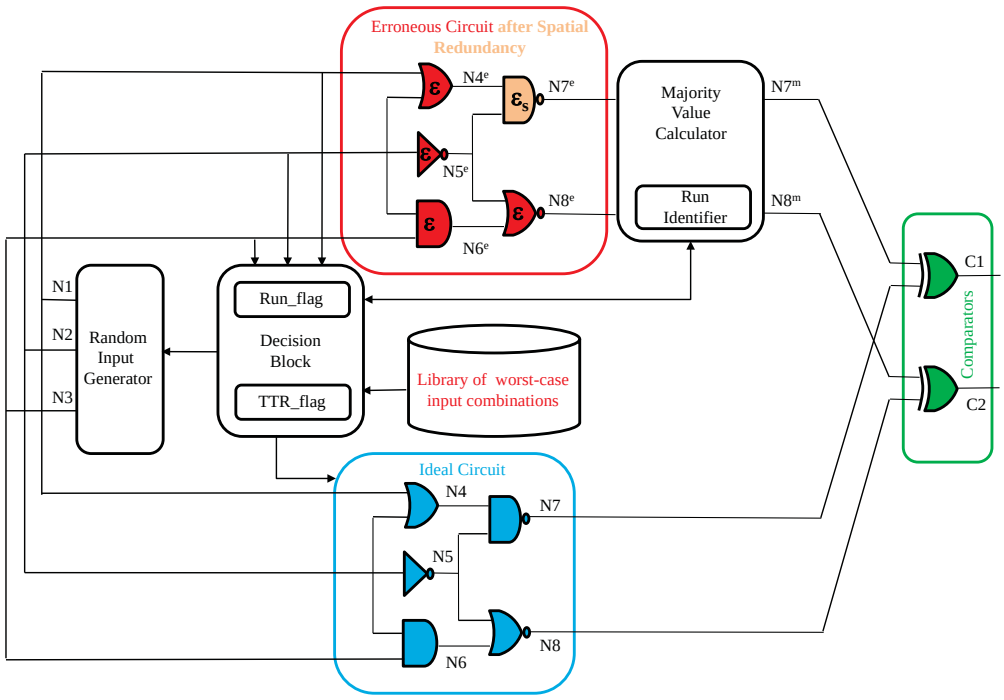


Figure 6.4. Hybrid redundancy scheme using CTMR and TTR techniques

dancy on an erroneous circuit whose error behavior is optimized by spatial redundancy. The spatial redundancy on the error model is first performed and then the modified structure is used to perform temporal redundancy. This procedure can be interpreted as performing temporal redundancy on a circuit whose sensitive gates will have a gate error probability ε_s compared to the less-sensitive nodes with gate error probability ε, where $\varepsilon_s < \varepsilon$. This redundancy scheme will have the relative merits of both temporal and spatial redundancy schemes.

6.4 Experimental Results

The experiments are performed using 8 million random input vectors and the probability of state "1" at the comparator outputs, $P(C_i = 1)$, are observed. For circuits with more than one primary output, the output error is observed as $\max_i P(C_i = 1)$. The results are presented as percentage improvements in mitigation of output error with redundancy over the output

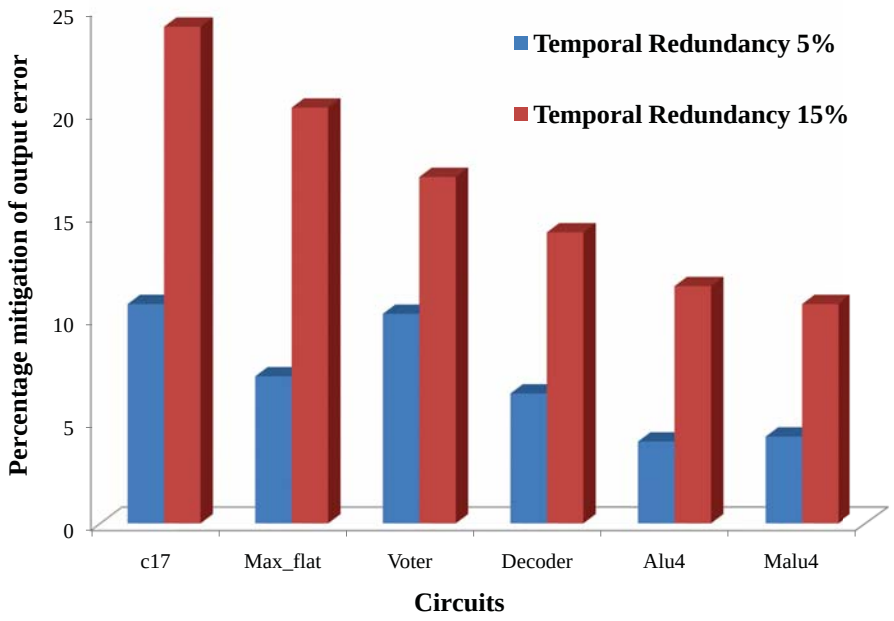


Figure 6.5. Percentage mitigation of output error achieved through 5% and 15% temporal redundancy with ε=0.001

error values without redundancy. Also results for two varying amounts of redundancy (5% and 15%) are presented, where the variation in temporal redundancy is achieved by varying the number of worst-case input combinations while performing TTR and varying the number of perturbed nodes while performing CTMR. All the results presented in this section are for gate error probability ε=0.001. Circuits from the ISCAS85 benchmark suite are used as test benches and the experiments are performed in a Pentium IV, 2.00 GHz, Windows XP computer.

6.4.1 Error Mitigation Through Temporal Redundancy

Fig. 6.5. gives the percentage mitigation of output error achieved through 5% and 15% temporal redundancy with ε=0.001. The important observations are listed as follows,

- For all the circuits, the error mitigation percentage for 15% temporal redundancy, is more than 10%. For 5% temporal redundancy, *c*17 and *voter* show significant error mitigation as compared to other circuits.

- For some circuits like *c*17 and max_flat, the error mitigation percentage is even beyond 20% when 15% of temporal redundancy is applied.

- For all circuits, the results clearly show the improvement in error mitigation when the amount of redundancy is increased. The improvement is more than 13% in circuits like *c*17 and max_flat, while for other circuits it is more than 6%.

6.4.2 Error Mitigation Through Spatial Redundancy

Fig. 6.6. gives the percentage mitigation of output error achieved through 5% and 15% spatial redundancy with ε=0.001. The important observations are listed as follows,

- Significant error mitigation is achieved for all circuits with 15% spatial redundancy. The error mitigation percentage is above 20% for all the circuits for 15% spatial redundancy, with *voter* achieving almost 50% error mitigation, *c*17 achieving 36% error mitigation and max_flat, *malu*4 achieving around 30% error mitigation.

- For 5% spatial redundancy, the error mitigation percentage is more than 10% for circuits like *c*17, max_flat, *voter* and *alu*4, while it is closer to 10% for the other circuits.

- The results clearly show significant improvement in percentage of error mitigation, when spatial redundancy is increased from 5% to 15%. While the improvement is as high as 33% for *voter*, for all the circuits except $decoder$, the improvement is more than 15%.

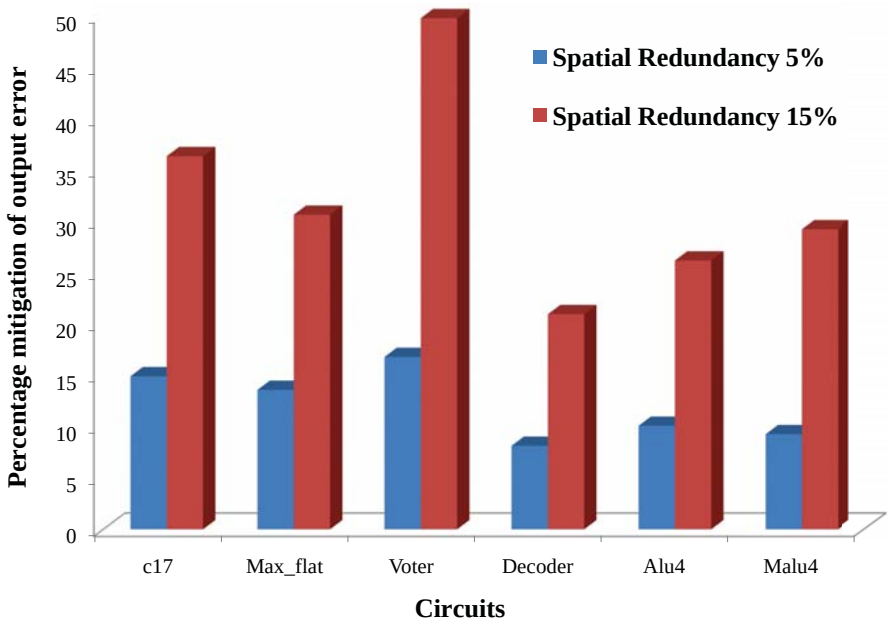


Figure 6.6. Percentage mitigation of output error achieved through 5% and 15% spatial redundancy with ε=0.001

6.4.3 Error Mitigation Through Hybrid Redundancy

Fig. 6.7. gives the percentage mitigation of output error achieved through 5% and 15% hybrid redundancy with ε=0.001. 5% hybrid redundancy is achieved using the combination of 5% temporal and 5% spatial redundancies, while 15% hybrid redundancy is achieved using the combination of 15% temporal and 15% spatial redundancies. The important observations are listed as follows,

- Significant error mitigation is achieved for all circuits with 15% hybrid redundancy. The error mitigation percentage is above 30% for all the circuits, with *voter* achieving as high as 60% error mitigation, $c17$ achieving 51% error mitigation and max_flat achieving around 47% error mitigation.

- Even for 5% hybrid redundancy, the error mitigation percentage is as high as 35% for *voter*, while it is more than 20% for the circuits like $c17$ and max_flat.

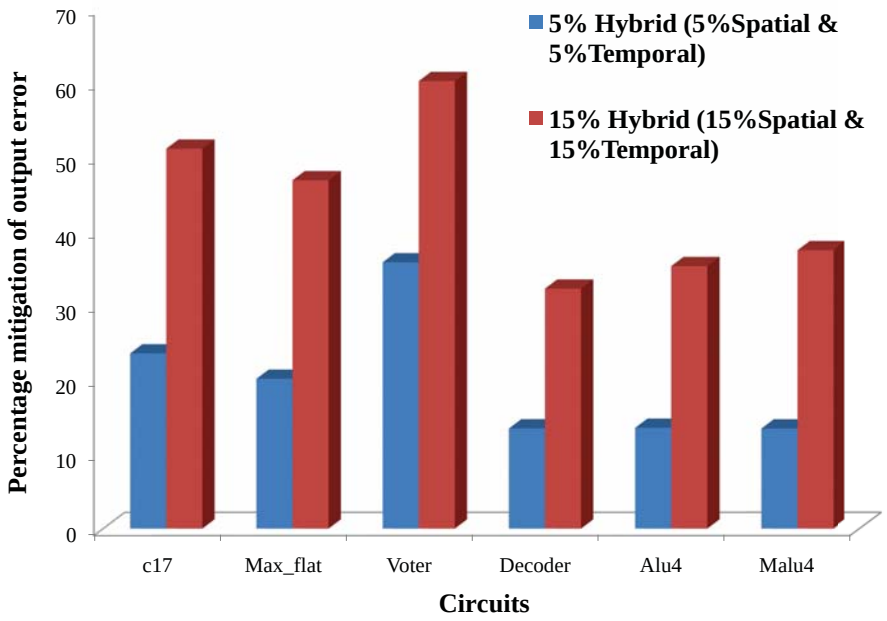


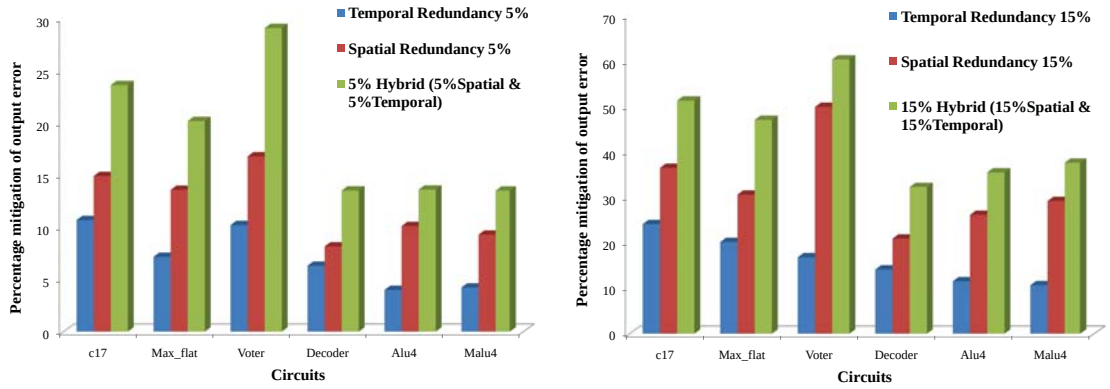
Figure 6.7. Percentage mitigation of output error achieved through 5% and 15% hybrid redundancy with ε=0.001

- The improvement in error mitigation, by increasing the amount of hybrid redundancy from 5% to 15%, is highly significant in all the circuit. While the improvement is above 24% in circuits like *c*17, *max_flat*, *voter* and *malu*4, it is around 20% for the circuits *decoder* and *alu*4.

6.4.4 Comparison Between the Redundancy Schemes

Fig. 6.8. gives the comparison between the redundancy schemes for 5% and 15% redundancy with ε=0.001. The important observations are listed as follows,

- As expected, hybrid redundancy provides better error mitigation as compared to temporal and spatial redundancies. The other notable result is that spatial redundancy provides better error mitigation as compared to temporal redundancy.



(a) (b)

Figure 6.8. Comparison between the redundancy schemes for (a) 5% and (b) 15% redundancy with ε=0.001

- For 5% redundancy case, the improvement in error mitigation using hybrid scheme as compared to temporal scheme is above 10% for circuits *c*17, *max_flat*, and above 18% for *voter*. Even the least improvement is around 7% for *decoder*, while the improvement in other circuits is around 9%.

- For 5% redundancy case, the improvement in error mitigation using hybrid scheme as compared to spatial scheme is above 10% for only *voter*, while the improvement in *c*17 is around 8%. The improvement in rest of the circuits is less than 7%.

- For 15% redundancy case, the improvement in error mitigation using hybrid scheme as compared to temporal scheme is above 20% for all circuits except *decoder*, with *voter* showing the highest improvement of about 43%. Even the least improvement is around 18% for *decoder*.

- For 15% redundancy case, the improvement in error mitigation using hybrid scheme as compared to spatial scheme is above 10% for all the circuits except *alu*4 and *malu*4. The highest improvement, shown by *max_flat*, is around 16%, while the improvement shown by *alu*4 and *malu*4 is above 8%.

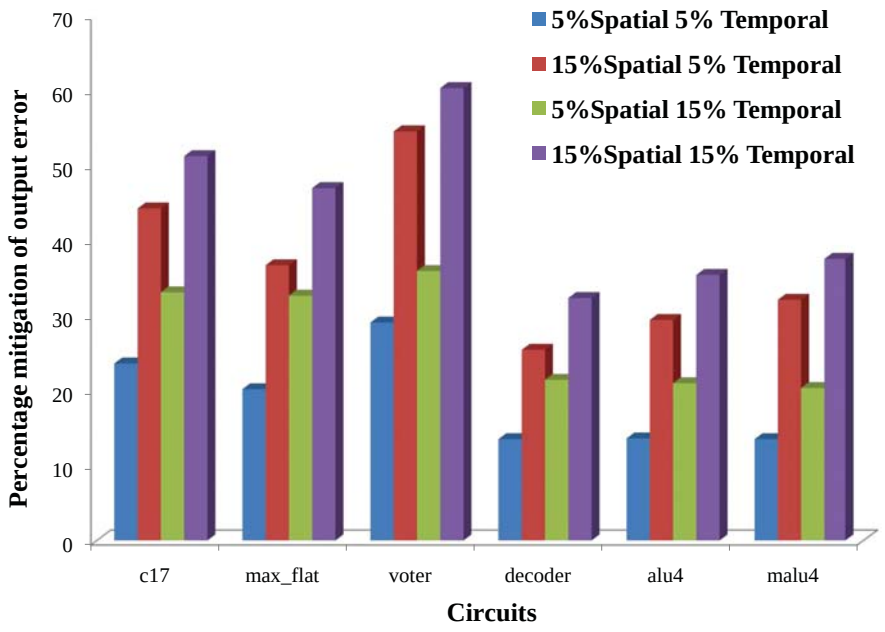


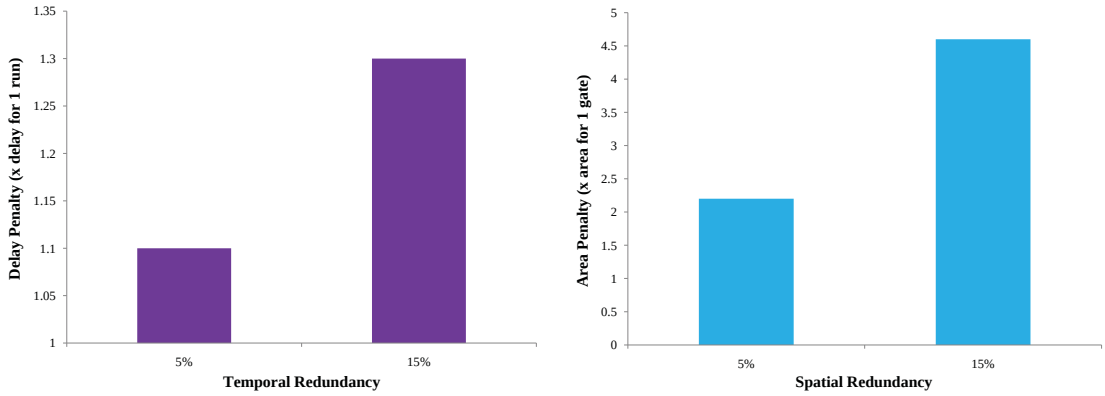
Figure 6.9. Percentage mitigation of output error achieved through hybrid redundancy with different combinations of spatial and temporal redundancies while ε=0.001

6.4.5 Error Mitigation Through Hybrid Redundancy with Different Combinations of Spatial and Temporal Redundancies

Fig. 6.9. gives the percentage mitigation of output error achieved through hybrid redundancy with different combinations of spatial and temporal redundancies while ε=0.001. The combinations include, 5% spatial and 5% temporal, 15% spatial and 5% temporal, 5% spatial and 15% temporal, 15% spatial and 15% temporal. The important observations are listed as follows,

- As expected, the combination of 15% spatial and 15% temporal redundancies yield the best error mitigation, while the combination of 5% spatial and 5% temporal redundancies yield the worst error mitigation.

- Comparing the combination 15% spatial and 5% temporal with the combination 5% spatial and 15% temporal, which is exactly the opposite, it is evident that providing more spatial redundancy is beneficial for error mitigation. The difference between the



(a) (b)

Figure 6.10. (a) Delay penalty in temporal redundancy (b) Area penalty in spatial redundancy

percentage mitigation of output error between these combinations is as high as 18% for *voter*, and more than 10% for *c*17 and *malu*4. While the difference is around 8% for *alu*4, it is as low as 4% for *max_flat* and *decoder*.

6.4.6 Delay and Area Penalties

Fig. 6.10. gives the delay penalty due to multiple runs with the same input vector in temporal redundancy, and the area penalty due to multiple copies of the same gate in spatial redundancy. The important observations are as follows,

- Area penalty is much larger than delay penalty. While the delay penalty is just 1.1 times the delay without redundancy for 5% temporal redundancy and 1.3 times the delay without redundancy for 15% temporal redundancy, the area penalty is 2.2 times the area without redundancy for 5% spatial redundancy and 4.6 times the area without redundancy for 15% spatial redundancy.

- This is obvious since the delay penalty is due to just 2 additional runs, while area penalty is due to 24 additional gates.

6.5 Discussion

We have performed temporal, spatial and hybrid redundancy, using our probabilistic error model, to achieve error mitigation in digital logic circuits. The percentage of error mitigation achieved using all the three types of redundancies are shown through experimental results. On an average, for 15% redundancy, 16% error mitigation was achieved with temporal redundancy, 32% error mitigation was achieved with spatial redundancy and 44% error mitigation was achieved with temporal redundancy. We have also provided a comprehensive study of the relative merits of these redundancy schemes, indicating the effectiveness of the hybrid redundancy, that encapsulates both temporal and spatial redundancy techniques.

CHAPTER 7

CONCLUSION AND FUTURE DIRECTIONS

In this dissertation, we have presented reliability models for nano VLSI circuits using probabilistic graphs and have accomplished the following,

- We have calculated the *maximum* error occurring in digital logic circuits and the corresponding worst-case input combination, through maximum *a posteriori* hypothesis, using an efficient Shenoy-Shafer algorithm. Through the results we have shown the importance of handling maximum error behavior for achieving fault tolerant computing machines. We have also studied the circuit-specific error bounds for fault-tolerant computing and the results clearly show that the error bounds are highly dependent on circuit structure and computation of maximum output error is essential to attain a tighter bound.

- We have calculated the average output error in *sequential* digital logic circuits and studied the transient error behavior across different time instances, using a dynamic time-evolving probabilistic error model. Through the results, we have shown the vulnerability of sequential circuits to transient errors and the dependence of error behavior to the circuit structure.

- We have performed temporal, spatial and hybrid redundancy, using our probabilistic error model, to achieve error mitigation in digital logic circuits. We have shown significant error reduction using all the three techniques and we also have provided a comprehensive study of the relative merits of these redundancy schemes, indicating the effective-

ness of the hybrid redundancy, that encapsulates both temporal and spatial redundancy techniques.

Some possible future directions of this work are as follows,

- This work can be further enhanced by obtaining real time gate error probability, ε, values from device physics and fabrication processes. Also using this model to solve reliability issues in real-time test benches like circuits used in automobiles and biomedical chips can further enlarge the scope and effectiveness of the model.

- To handle large circuits, stochastic heuristic algorithms to detect both average and maximum error can be proposed. This work can serve as a baseline exact estimate to judge the efficacy of the various stochastic heuristic algorithms that will be essential for circuits of higher dimensions.

- The error model to detect error in sequential circuits can be further enhanced by exploring error masking effects, like the latching window masking effect, that can arise in an erroneous latch connected in the feedback path.

- The error models can be enhanced by addressing design aspects like timing violations leading to delay faults.

- Since our model can be more versatile, apart from addressing global erroneous behavior, we should also address specific reliability issues like signal integrity, by modeling the gate error probability values for each gate based on this specific reliability issue.

REFERENCES

[1] J. Von Neumann, "Probabilistic Logics and the Synthesis of Reliable Organisms from Unreliable Components", in *Automata Studies* (C. E. Shannon and J. McCarthy, eds.), pages 43–98, Princeton Univ. Press, Princeton, N.J., 1954.

[2] F. P. Mathur and A. Avižienis, "Reliability Analysis and Architecture of a Hybrid-Redundant Digital System: Generalized Triple Modular Redundancy with Self-Repair", *AFIPS Joint Computer Conferences*, pages 375–383, 1970.

[3] N. Pippenger, "Reliable Computation by Formulas in the Presence of Noise", *IEEE Trans on Information Theory*, vol. 34(2), pages 194–197, 1988.

[4] T. Feder, "Reliable Computation by Networks in the Presence of Noise", *IEEE Trans on Information Theory*, vol. 35(3), pages 569–571, 1989.

[5] B. Hajek and T. Weller, "On the Maximum Tolerable Noise for Reliable Computation by Formulas", *IEEE Trans on Information Theory*, vol. 37(2), pages 388–391, 1991.

[6] W. Evans and L. J. Schulman, "On the Maximum Tolerable Noise of k-input Gates for Reliable Computation by Formulas", *IEEE Trans on Information Theory*, vol. 49(11), pages 3094–3098, 2003.

[7] W. Evans and N. Pippenger, "On the Maximum Tolerable Noise for Reliable Computation by Formulas", *IEEE Transactions on Information Theory*, vol. 44(3), pages 1299–1305, 1998.

[8] J. B. Gao, Y. Qi and J. A. B. Fortes, "Bifurcations and Fundamental Error Bounds for Fault-Tolerant Computations", *IEEE Transactions on Nanotechnology*, vol. 4(4), pages 395–402, 2005.

[9] D. Marculescu, R. Marculescu and M. Pedram, "Theoretical Bounds for Switching Activity Analysis in Finite-State Machines", *IEEE Transactions on VLSI Systems*, vol. 8(3), pages 335–339, 2000.

[10] P. G. Depledge, "Fault-Tolerant Computer Systems", *IEE Proc. A*, vol. 128(4), pages 257–272, 1981.

[11] S. Spagocci and T. Fountain, “Fault Rates in Nanochip Devices”, in *Electrochemical Society*, pages 354–368, 1999.

[12] J. Han and P. Jonker, “A Defect- and Fault-Tolerant Architecture for Nanocomputers”, *Nanotechnology*, vol. 14, pages 224–230, 2003.

[13] S. Roy and V. Beiu, “Majority Multiplexing-Economical Redundant Fault-tolerant Designs for Nano Architectures”, *IEEE Transactions on Nanotechnology*, vol. 4(4), pages 441–451, 2005.

[14] K. Nikolic, A. Sadek, and M. Forshaw, “Fault-Tolerant Techniques for Nanocomputers,” *Nanotechnology*, vol. 13, pages 357–362, 2002.

[15] J. Han, E. Taylor, J. Gao and J. A. B. Fortes, ‘”Reliability Modeling of Nanoelectronic Circuits”, *IEEE Conference on Nanotechnology*, 2005.

[16] J. B. Gao, Yan Qi and J.A.B. Fortes, “Markov Chains and Probabilistic Computation - A General Framework for Multiplexed Nanoelectronic Systems”, *IEEE Transactions on Nanotechnology*, vol. 4(2), pages 395–402, 2005.

[17] E. Taylor, J. Han and J. A. B. Fortes, “Towards Accurate and Efficient Reliability Modeling of Nanoelectronic Circuits”, *IEEE Conference on Nanotechnology*, pages 395–398, 2006.

[18] M. O. Simsir, S. Cadambi, F. Ivancic, M. Roetteler and N. K. Jha, “Fault-Tolerant Computing Using a Hybrid Nano-CMOS Architecture”, *International Conference on VLSI Design*, pages 435–440, 2008.

[19] C. Chen and Y. Mao, “A Statistical Reliability Model for Single-Electron Threshold Logic”, *IEEE Transactions on Electron Devices*, vol. 55, pages 1547–1553, 2008.

[20] A. Abdollahi, “Probabilistic Decision Diagrams for Exact Probabilistic Analysis”, *IEEE/ACM International Conference on Computer-Aided Design*, pages 266–272, 2007.

[21] M. R. Choudhury and K. Mohanram, “Accurate and Scalable Reliability Analysis of Logic Circuits”, *Design, Automation, and Test in Europe (DATE) conference*, pages 1454–1459, 2007.

[22] S. Lazarova-Molnar, V. Beiu and W. Ibrahim, “A Strategy for Reliability Assessment of Future Nano-Circuits”, *WSEAS International Conference on Circuits*, pages 60–65, 2007.

[23] P. P. Shenoy and G. Shafer, “Propagating Belief Functions with Local Computations”, *IEEE Expert*, vol. 1(3), pages 43–52, 1986.

[24] P. P. Shenoy, “Binary Join Trees for Computing Marginals in the Shenoy-Shafer Architecture”, *International Journal of Approximate Reasoning*, pages 239–263, 1997.

[25] P. P. Shenoy, "Valuation-Based Systems: A Framework for Managing Uncertainty in Expert Systems", *Fuzzy Logic for the Management of Uncertainty*, pages 83–104, 1992.

[26] J. Pearl, "Probabilistic Reasoning in Intelligent Systems: Network of Plausible Inference", Morgan Kaufmann Publishers, Inc., 1988.

[27] F. V. Jensen, S. Lauritzen and K. Olesen, "Bayesian Updating in Recursive Graphical Models by Local Computation", *Computational Statistics Quarterly*, pages 269-282, 1990.

[28] R. G. Cowell, A. P. David, S. L. Lauritzen and D. J. Spiegelhalter, "Probabilistic Networks and Expert Systems," Springer-Verlag New York, Inc., 1999.

[29] J. D. Park and A. Darwiche, "Solving MAP Exactly using Systematic Search", *Conference on Uncertainty in Artificial Intelligence*, 2003.

[30] J. D. Park and A. Darwiche, "Approximating MAP using Local Search", *Conference on Uncertainty in Artificial Intelligence*, pages 403–410, 2001.

[31] Sensitivity Analysis, Modeling, Inference and More (SAMIAM), *http://reasoning.cs.ucla.edu/samiam/*, Automated Reasoning Group, University of California, Los Angeles.

[32] J. P. Roth, "Diagnosis of Automata Failures: A Calculus and a Method", *IBM Journal of Research and Development*, vol. 10(4), pages 278–291, 1966.

[33] P. Goel, "An Implicit Enumeration Algorithm to Generate Tests for Combinational Logic Circuits", *IEEE Transactions on Computers*, vol. C-30(3), pages 215–222, 1981.

[34] H. Fujiwara and T. Shimono, "On The Acceleration of Test Generation Algorithms", *IEEE Transactions on Computers*, vol. C-32(12), pages 1137–1144, 1983.

[35] V. D. Agrawal, S. C. Seth and C. C. Chuang, "Probabilistically Guided Test Generation", *IEEE International Symposium on Circuits and Systems*, pages 687–690, 1985.

[36] J. Savir, G. S. Ditlow and P. H. Bardell, "Random Pattern Testability", *IEEE Transactions on Computers*, vol. C-33(1), pages 79–90, 1984.

[37] C. Seth, L. Pan and V. D. Agrawal, "PREDICT - Probabilistic Estimation of Digital Circuit Testability", *IEEE International Symposium on Fault-Tolerant Computing*, pages 220–225, 1985.

[38] S. T. Chakradhar, M. L. Bushnell and V. D. Agrawal, "Automatic Test Generation using Neural Networks", *IEEE International Conference on Computer-Aided Design*, vol. 7(10), pages 416–419, 1988.

[39] M. Mason, “FPGA Reliability in Space-Flight and Automotive Applications”, *FPGA and Programmable Logic Journal*, 2005.

[40] E. Zanoni and P. Pavan, “Improving the Reliability and Safety of Automotive Electronics”, *IEEE Micro*, vol. 13(1), pages 30–48, 1993.

[41] P. Gerrish, E. Herrmann, L. Tyler and K. Walsh, “Challenges and Constraints in Designing Implantable Medical ICs”, *IEEE Transactions on Device and Materials Reliability*, vol. 5(3), pages 435–444, 2005.

[42] L. Stotts, “Introduction to Implantable Biomedical IC Design”, *IEEE Circuits and Devices Magazine*, pages 12–18, 1999.

[43] S. Cheemalavagu, P. Korkmaz, K. V. Palem, B. E. S. Akgul and L. N. Chakrapani, “A Probabilistic CMOS Switch and its Realization by Exploiting Noise”, *IFIP International Conference on Very Large Scale Integration*, 2005.

[44] Military Standard (MIL-STD-883), “Test Methods and Procedures for Microelectronics”, 1996.

[45] S. Krishnaswamy, G. S. Viamontes, I. L. Markov, and J. P. Hayes, “Accurate Reliability Evaluation and Enhancement via Probabilistic Transfer Matrices”, *Design Automation and Test in Europe (DATE) Conference*, pages 282–287, 2005.

[46] J. Han, J. B. Gao, P. Jonker, Y. Qi and J. A. B. Fortes, “Toward Hardware-Redundant Fault-Tolerant Logic for Nanoelectronics”, *IEEE Transactions on Design and Test of Computers*, vol. 22(4), pages 328–339, 2005.

[47] R. I. Bahar, J. Mundy, and J. Chan, “A Probabilistic Based Design Methodology for Nanoscale Computation”, *International Conference on Computer Aided Design (ICCAD)*, pages 480–486, 2003.

[48] D. Bhaduri and S. K. Shukla, “NANOPRISM: A Tool for Evaluating Granularity vs. Reliability Trade-offs in Nano Architectures”, *Great Lakes Symposium on VLSI*, pages 109–112, 2004.

[49] L. P. Yuan, C. C. Teng and S. M. Kang, “Statistical Estimation of Average Power Dissipation in Sequential Circuits,” *Design Automation Conference (DAC)*, pages 377–382, 1997.

[50] HUGIN Inference Tool, *http://www.hugin.com/*, HUGIN EXPERT A/S, Aalborg, Denmark.

[51] N. M. Zivanov and D. Marculescu, “Soft Error Rate Analysis for Sequential Circuits”, *Design Automation and Test in Europe (DATE) Conference*, pages 1–6, 2007.

[52] J. J. Shedletsky and E. J. McCluskey, “The Error Latency of a Fault in a Sequential Digital Circuit”, *IEEE Transactions on Computers*, vol. C-25(6), pages 655–659, 1976.

[53] S. Y. Huang, K. T. Cheng, K. C. Chen and J. Y. Lu, “Fault-Simulation Based Design Error Diagnosis for Sequential Circuits”, *Design Automation Conference (DAC)*, pages 632–637, 1998.

[54] H. Asadi and M. B. Tahoori, “Soft Error Modeling and Protection for Sequential Elements”, *IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems*, pages 463–471, 2005.

[55] R. Baumann, “Soft Errors in Advanced Computer Systems”, *IEEE Design and Test of Computers*, vol. 22(3), pages 258–266, 2005.

[56] M. Zhang and N. R. Shanbag, “A Soft Error Rate Analysis (SERA) Methodology”, *International Conference on Computer Aided Design (ICCAD)*, pages 111–118, 2004.

[57] S. Winograd and J. D. Cowan, “Reliable Computation in the Presence of Noise”, *The MIT Press*, 1963.

[58] G. Norman, D. Parker, M. Kwiatkowska and S. K. Shukla, “Evaluating the Reliability of Defect-Tolerant Architectures for Nanotechnology with Probabilistic Model Checking”, *International Conference on VLSI Design*, pages 907–912, 2004.

[59] International Technology Roadmap for Semiconductors (ITRS), *http://www.itrs.net/Links/2005ITRS/ERD2005.pdf*, 2005.

[60] G. E. Moore, “Cramming More Components onto Integrated Circuits”, *Electronics*, vol. 38(8), 1965.

[61] L. B. Kish, “End of Moore’s Law: Thermal (Noise) Death of Integration in Micro and Nano Electronics”, *Physics Letters A*, vol. 305(3–4), pages 144–149, 2002.

[62] C. W. Gwyn, D. L. Scharfetter and J. L. Wirth, “The Analysis of Radiation Effects in Semiconductor Junction Devices”, *IEEE Transactions on Nuclear Science*, vol. NS-14(6), pages 153–169, 1967.

[63] D. C. DAvanzo, M. Vanzi and R. W. Dutton, “One-Dimensional Semiconductor Device Analysis, *Tech. Rep. no. G-201-5*, Stanford Electronics Laboratories, Stanford University, 1979.

[64] S. Selberherr, W. Fichtner and H. W. Potzl, “MINIMOS - A Program Package to Facilitate MOS Device Design and Analysis”, *NASECODE I*, pages 275–279, 1979.

[65] P. E. Cottrell and E. M. Buturla, “Two-Dimensional Static and Transient Simulation of Mobile Carrier Transport in a Semiconductor, *NASECODE I*, pages 31–64, 1979.

[66] E. M. Buturla, P. E. Cottrell, B. M. Grossman, K. A. Salsburg, M. B. Lawlor and C. T. McMullen, "Three-Dimensional Finite Element Simulation of Semiconductor Devices", *IEEE Int. Solid State Circuits Conf. Dig. Tech. Papers*, pages 76–77, 1980.

[67] M. R. Pinto, C. S. Rafferty and R. W. Dutton, "PISCES-II: Poisson and Continuity Equation Solver", *Stanford Electronics Laboratories*, 1984.

[68] P. E. Dodd, "Device Simulation of Charge Collection and Single-Event Upset", *IEEE Transactions on Nuclear Science*, vol. 43(2), pages 561–575, 1996.

[69] G. R. Srinivasan, H. K. Tang and P. C. Murley, "Parameter-Free, Predictive Modeling of Single Event Upsets due to Protons, Neutrons and Pions in Terrestrial Cosmic Rays", *IEEE Transactions on Nuclear Science*, vol. 41(6), pages 2063–2070, 1994.

[70] M. R. Choudhury, Q. Zhou and K. Mohanram, "Design Optimization for Single-Event Upset Robustness using Simultaneous Dual-VDD and Sizing Techniques", *International Conference on Computer Aided Design (ICCAD)*, pages 204–209, 2006.

[71] K. Bhattacharya and N. Ranganathan, "A New Placement Algorithm for Reduction of Soft Errors in Macro Cell based Design of Nanometer Circuits", *IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, pages 91–96, 2009.

[72] N. Miskov-Zivanov and D. Marculescu, "MARS-C: Modeling and Reduction of Soft Errors in Combinational Circuits", *Design Automation Conference (DAC)*, pages 767–772, 2006.

[73] P. K. Samudrala, J. Ramos and S. Katkoori, "Selective Triple Modular Redundancy (STMR) Based Single-Event-Upset (SEU) Tolerant Synthesis for FPGAs", *IEEE Transactions on Nuclear Science*, vol. 51(5), pages 2957–2969, 2004.

[74] W. A. Moreno, J. R. Samson Jr. and F. J. Falquez, "Laser Injection of Soft Faults for the Validation of Dependability Design", *Journal of Universal Computer Science*, vol. 5(10), pages 712–729, 1999.

[75] Paris D. Wiley, "Fault Tolerant Design Verification Through The Use of Laser Fault Injection", *Masters Thesis, Department of Electrical Engineering, University of South Florida*, 2004.

[76] A. Sanyal, S. M. Alam and S. Kundu, "A Built-In Self-Test Scheme for Soft Error Rate Characterization", *IEEE International On-Line Testing Symposium*, pages 65–70, 2008.

[77] S. Bhanja and N. Ranganathan, "Switching Activity Estimation of VLSI Circuits using Bayesian Networks", *IEEE Transactions on VLSI Systems*, pages 558–567, 2003.

[78] S. Bhanja and N. Ranganathan, "Cascaded Bayesian Inferencing for Switching Activity Estimation with Correlated Inputs", *IEEE Transaction on VLSI Systems*, vol. 12(12), pages 1360–1370, 2004.

[79] S. Bhanja and N. Ranganathan, "Modeling Switching Activity Using Cascaded Bayesian Networks for Correlated Input Streams ", *International Conference on Computer Design (ICCD)*, pages 388–390, 2002.

[80] S. Bhanja and N. Ranganathan, "Accurate Switching Activity Estimation of Large Circuits using Multiple Bayesian Networks", *15th Intl. Conference of VLSI Design and 7th ASP-Design and Automation Conference*, pages 187–192, 2002.

[81] S. Bhanja and N. Ranganathan, "Dependency Preserving Probabilistic Modeling of Switching Activity using Bayesian Networks", *IEEE/ACM Design Automation Conference (DAC)*, pages 209–214, 2001.

[82] S. Bhanja and S. Sarkar, "Probabilistic Modeling of QCA Circuits using Bayesian Networks", *IEEE Transactions on Nanotechnology*, vol. 5(6), pages 657–670, 2006.

[83] S. Bhanja and S. Sarkar, "Switching Error Modes of QCA Circuits", *IEEE Conference on Nanotechnology*, vol. 1, pages 383–386, 2006.

[84] S. Bhanja and S. Sarkar, "Graphical Probabilistic Inference for Ground State and Near-Ground State Computing in QCA Circuits", *IEEE Conference on Nanotechnology*, pages 290–293, 2005.

[85] T. Rejimon and S. Bhanja, "A Timing-Aware Probabilistic Model for Single-Event-Upset Analysis", *IEEE Transactions on VLSI Systems*, vol. 14(10), pages 1130–1139, 2006.

[86] T. Rejimon and S. Bhanja, "Probabilistic Error Model for Unreliable Nano-logic Gates", *IEEE Conference on Nanotechnology*, pages 717–722, 2006.

[87] T. Rejimon and S. Bhanja, "Time and Space Efficient Method for Accurate Computation of Error Detection Probabilities", *IEE Computers and Digital Techniques*, vol. 152(5), pages 679–685, 2005.

[88] T. Rejimon and S. Bhanja, "A Stimulus-Free Probabilistic Model for Single-Event-Upset Sensitivity", *IEEE Intl. Conference on VLSI Design*, 2006.

[89] T. Rejimon, L. Hoffmann and S. Bhanja, "A Probabilistic Model for Single-Event-Upset", *12th NASA Symposium on VLSI*, 2005.

[90] T. Rejimon and S. Bhanja, "An Accurate Probabilistic Model for Error Detection", *18th International Conference in VLSI Design*, pages 717–722, 2005.

[91] T. Rejimon and S. Bhanja, "Scalable Probabilistic Computing Models using Bayesian Networks", *IEEE Midwest Symposium on Circuits and Systems*, pages 712–715, 2005.

[92] S. Ramani and S. Bhanja, "Anytime Probabilistic Switching Model using Bayesian Networks," *International Symposium on Low Power Electronic Design*, pages 86–89, 2004.

[93] N. Ramalingam and S. Bhanja, "Causal Probabilistic Input Dependency Learning for Switching Model in VLSI Circuits", *ACM Great Lake Symposium on VLSI*, pages 112–115, 2005.

[94] S. Srivastava and S. Bhanja, "Hierarchical Probabilistic Macromodeling for QCA Circuits", *IEEE Transactions on Computers*, vol. 56(2), pages 174–190, 2007.

[95] S. Srivastava and S. Bhanja, "Bayesian Macromodeling for Circuit Level QCA Design", *IEEE Conference on Nanotechnology*, pages 31–34, 2006.

[96] S. Srivastava and S. Bhanja, "Hierarchical Bayesian Macromodeling for QCA Circuits", *12th NASA Symposium on VLSI*, 2005.

[97] S. Bhanja and S. Srivastava, "Bayesian Modeling of Quantum-dot Cellular Automata Circuits", *NSTI, Nanotechnology Conference*, 2005.

[98] S. Bhanja, K. Lingasubramanian and N. Ranganathan, "A Stimulus-Free Graphical Probabilistic Switching Model for Sequential Circuits using Dynamic Bayesian Networks", *ACM Transactions on Design Automation of Electronic Systems*, vol. 11(3), pages 773–796, 2006.

[99] T. Rejimon, K. Lingasubramanian and S. Bhanja, "Probabilistic Error Model for Nano-Domain Logic Circuits", *IEEE Transactions on VLSI*, vol. 17(1), pages 55–65, 2008.

[100] K. Lingasubramanian and S.Bhanja, "Probabilistic Maximum Error Modeling for Unreliable Logic Circuits", *ACM Great Lake Symposium on VLSI*, pages 223–226, 2007.

[101] K. Lingasubramanian and S. Bhanja, "Probabilistic Error Modeling for Sequential Logic", *IEEE International Conference on Nanotechnology*, pages 616–620, 2007.

[102] K. Lingasubramanian and S. Bhanja, "An Error Model to Study the Behavior of Transient Errors in Sequential Circuits", *IEEE International Conference on VLSI Design*, pages 485–490, 2009.

[103] A. Shareef, K. Lingasubramanian and S. Bhanja, "Selective Redundancy: Evaluation of Temporal Reliability Enhancement Scheme for Nanoelectronic Circuits", *IEEE Conference on Nanotechnology*, pages 895–898, 2008.

[104] S. Bhanja, K. Lingasubramanian and N. Ranganathan, "Estimation of Switching Activity in Sequential Circuits Using Dynamic Bayesian Networks", *18th International Conference in VLSI Design*, pages 586–591, 2005.

ABOUT THE AUTHOR

Karthikeyan Lingasubramanian received the B.E. degree in electronics and communication engineering from Kumaraguru College of Technology, India, in 2001. He received his M.S. degree in electrical engineering from University of South Florida, Tampa, USA, in 2004, where he is currently pursuing the Ph.D. degree in electrical engineering. His research interests include design automation and testing, comprehensive nano-domain probabilistic and statistical models for estimation and optimization of error and power.