University of Alberta

Probabilistic and Stochastic Computational Models: from Nanoelectronic to Biological Applications

by

Jinghang Liang

A thesis submitted to the Faculty of Graduate Studies and Research
in partial fulfillment of the requirements for the degree of

Master of Science
in
Computer Engineering

Department of Electrical and Computer Engineering

To my family and friends

ABSTRACT

A finite state machine (FSM) is a classical abstract model for sequential circuits that are at the core of any digital system. Due to fabrication defects and transient faults, the reliable operation of sequential circuits is greatly desired. In this thesis, computational models are initially constructed using state transition matrices (STMs) and binary decision diagrams (BDDs) of an FSM; a phenomenon called error masking and the restoring properties of sequential circuits are then analyzed in detail. This analysis provides a basis for further devising efficient and robust implementation when designing FSMs.

Arithmetic circuits play an important role in many digital systems and have fundamentally critical applications in signal processing. Addition is perhaps the most important and basic arithmetic operation for many applications. Recent research has focused on probabilistic and approximate adders that trade off accuracy for energy saving. Since there was a lack of appropriate metrics to evaluate the efficacy of these inexact designs, several new metrics are proposed in this work for evaluating the reliability as well as the power efficiency of an adder. These new metrics can be used in future designs for a better assessment of the power and precision tradeoff.

Although current digital systems are based on complementary metal–oxide–semiconductor (CMOS) technology and employ binary values in the representation of signals, multiple valued logic (MVL) circuits using novel nanodevices have been investigated due to their advantages in information density and operating speed. In this thesis, pseudo-complementary MVL circuits are further proposed for implementations using carbon nanotube field effect transistors (CNTFETs). Because of the fabrication non-idealities, reliability evaluation of these MVL circuits becomes important. Subsequently, stochastic computational models (SCMs) are developed to analyze the reliability of CNT MVL circuits.

Finally, the stochastic computational model is applied in the modeling of biological networks. Specifically, stochastic Boolean networks (SBNs) are proposed for an efficient modeling of genetic regulatory networks (GRNs). The proposed SBN can accurately and efficiently simulate a GRN without and with random gene perturbation, which will help to reveal biologically meaningful insights for a better understanding of the dynamics of GRNs.

PREFACE

As technology scales down in the nanometer regime, either complementary metal–oxide–semiconductor (CMOS) or novel technology-based circuits suffer from the problems of fabrication defects and transient faults. Noise and process variations are unavoidable in an implementation [1]. As a result, the behavior of future circuits is predicted to be probabilistic instead to be deterministic and therefore reliability evaluation techniques will become more important for future circuit design. Several computational methodologies have been developed for evaluating the reliability of combinational logic circuits. The analysis of the operation of sequential circuits is mostly considered as a direct extension of combinational techniques. However, sequential circuits have some unique features. As the feedback signals in a sequential circuit can be logically masked by specific combinations of primary inputs, the cumulative effects of soft errors can be logically masked. This phenomenon, referred to as error masking, is related to the presence of so-called restoring inputs and/or the consecutive presence of specific inputs in multiple clock cycles (equivalent to a synchronizing sequence in switching theory [96, 97]). State transition matrices (STMs) and binary decision diagrams (BDDs) are used to analyze the reliable function of a finite state machine (FSM). Extensive simulations of benchmark circuits are performed to provide a basis for further devising efficient and robust implementations of FSMs.

As a fundamental arithmetic operation in many applications of inexact computing, soft addition has attracted a lot of research attention [37, 38, 40]. Generally, a soft adder is based on the operation of deterministic approximate logic or probabilistic imprecise arithmetic. Since the traditional metric of reliability (defined as the probability of system survival) is not appropriate for use in evaluating approximate designs, new design metrics are urgently needed to evaluate the effectiveness of various designs. This thesis has proposed new metrics for evaluating adder designs with respect to reliability and power efficiency for inexact computing. A detailed analysis and simulation results are presented to assess the reliable performance of these adders using the proposed new metrics. It is shown that these metrics can be useful in assessing future inexact adder designs.

Recently, carbon nanotube field effect transistors (CNTFETs) have been extensively studied as a possible successor to silicon metal–oxide–semiconductor field effect transistors (MOSFETs) and as a possible implementation of multiple-valued logic (MVL) [48, 49]. In this thesis, a pseudo-complementary CNTFET-based MVL design is proposed. In particular, ternary and quaternary gates are illustrated in detail as examples. Due to the presence of manufacturing defects, these proposed gates operate in a probabilistic way. Therefore the reliability analysis of the proposed MVL gates and circuits is considered next. Since previous methods of reliability evaluation have a large complexity, an approach using stochastic computational models (SCMs) is developed for the proposed MVL gates and circuits as a scalable technique for reliability evaluation in this thesis.

Similar to nanoscale circuits, biological networks also present probabilistic behaviors. Probabilistic Boolean networks (PBNs) have been proposed to address these probabilistic behaviors of gene regulatory networks (GRNs) [66, 67]. However, their computational complexity makes them difficult to use. To address this, stochastic Boolean networks (SBNs) are proposed as an efficient approach to modeling GRNs. The SBN approach is able to reproduce biologically-proven regulatory behaviors, such as the oscillatory dynamics of the p53-Mdm2 network and the dynamic attractors in a T cell immune response network. The proposed approach can further predict the network dynamics when the genes are under perturbation, thus providing biologically-meaningful insights for a better understanding of the dynamics of GRNs. Our proposed algorithms and methods have been implemented in Matlab packages and can be used for future applications of GRN analysis.

ACKNOWLEDGEMENTS

The work presented in this thesis could not have happened without the help of many people. My supervisor, Dr. Jie Han, has always been providing me with not only support and guidance in my research work, but also a constant source of encouragement throughout the duration of this thesis. I would like to thank Dr. Fabrizio Lombardi for his insightful advices into this research. I would also like to thank Dr. Bruce Cockburn and Dr. Guohui Lin for serving on my dissertation committee.

I would like to thank Hao Chen, who has always provided wonderful feedback, ideas, and friendship. I would also like to thank many other people for the discussions: Zhiyin Zhou, Russel Dodd, Joyce Li, Zhixi Yang, Peican Zhu and Linbin Chen.

I wish to thank my entire family, for their love, support, and encouragement.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ACRONYMS

ACCNT	Asymmetrically-Correlated Carbon Nanotube
AMA	Approximate Mirror Adder
BDD	Binary Decision Diagram
BN	Boolean Network
CMOS	Complementary Metal Oxide Semiconductor
CNTFET	Carbon Nanotube Field Effect Transistor
CFA	Conventional Full Adder
CME	Chemical Master Equation
CSTM	Cumulative State Transition Matrix
DA-BN	Deterministic-Asynchronous Boolean Network
DA-PBN	Deterministic-Asynchronous Probabilistic Boolean Network
DRAM	Dynamic Random Access Memory
ED	Error Distance
FET	Field Effect Transistor
FSM	Finite State Machine
GRN	Gene Regulatory Network
IDCT	Inverse Discrete Cosine Transform
ITRS	International Technology Roadmap for Semiconductors

LIA	Lower-bit Ignored Adder
LOA	Lower-part OR Adder
MA	Mirror Adder
MCMC	Markov Chain Monte Carlo
MC	Monte Carlo
MED	Mean Error Distance
MPU	Micro Processor Unit
MVL	Multiple Valued Logic
NED	Normalized Error Distance
NS	Next States
NTI	Negative Ternary Inverter
PBN	Probabilistic Boolean Network
PCMOS	Probabilistic Complementary Metal Oxide Semiconductors
PDD	Probabilistic Decision Diagram
PFA	Probabilistic Full Adder
PGM	Probabilistic Gate Model
PS	Present States
PTI	Positive Ternary Inverter
PTM	Probabilistic Transfer Matrix
SBN	Stochastic Boolean Network
SCM	Stochastic Computational Model

SER	Soft Error Rate
SEU	Single Event Upset
SPICE	Simulation Program with Integrated Circuit Emphasis
SPTM	Sequential Probabilistic Transfer Matrix
SSA	Stochastic Simulation Algorithm
STI	Standard Ternary Inverter
STM	State Transition Matrix
VLSI	Very-Large-Scale Integration

CHAPTER 1

Introduction

Finite state machines (FSMs) play significant roles in modern circuits and systems. Reliability evaluation of FSMs is urgently needed due to the problems of fabrication defects and transient faults. As reliability is becoming a major design metric [1], various reliability evaluation techniques have been proposed for use in combinational logic circuits. These include accurate but computationally expensive techniques such as those using probabilistic transfer matrices (PTMs) [2], probabilistic gate models (PGMs) [3] and probabilistic decision diagrams (PDDs) [4], as well as those more efficient but approximate approaches using Bayesian networks [5], Boolean difference calculus [6], circuit transformations [7] and several other scalable techniques [8]. A recent approach using stochastic computational models (SCMs) has been presented for a highly accurate analysis of logic circuits with moderate computational complexity [9].

This chapter includes an introduction to several approaches for evaluating the reliability of combinational circuits. It is organized as follows. Section 1.1 describes the background and motivation of reliability evaluation. Section 1.2 reviews several existing approaches and related work. Section 1.3 provides a description of the contribution of this work and Section 1.4 gives the outline of this thesis.

1.1. Background and Motivation

Following Moore's Law, the number of transistors in a single chip is doubled every 18 months. Device scaling refers to the reductions in feature size and voltage levels of transistor. This improves performance because smaller devices require lower voltages to turn on or off, and thus can be operated at higher frequencies. As transistors become smaller, they switch faster, dissipate less power, and are cheaper to manufacture. However, this downscaling also brings many challenges for future technologies.

1.1.1. Technology Scaling

According to the International Technology Roadmap for Semiconductors (ITRS), which predicts complementary metal–oxide–semiconductor (CMOS) feature size scaling and directs research effort to address future challenges, the feature size of CMOS transistors has been becoming smaller and the density of transistors per unit area is doubled every generation [10].

There is no doubt that downscaling has a positive impact on performance and cost; however, the power density has increased with higher clock frequency and higher V_{DD} than predicted [11]. Therefore, temperature-related noise and interference become more significant. In order to reduce static power consumption, lower V_{DD} as well as higher V_{th} are adopted. This increases supply voltage noise [12] and threshold variations [13] that lead to lower circuit reliability. Besides, smaller transistor sizes make devices more susceptible to manufacturing defects [14]. Consequently, future design will have to place more emphasis on coping with those unpredictable circuit behaviors.

1.1.2 Permanent Defects and Soft Errors

In practical implementations, defects are almost unavoidable. Random defects in flash, dynamic random access memory (DRAM) and micro-processor unit (MPU) remain the same generation by generation [10]. Typical defects in very-large-scale integration (VLSI) chips include process defects, material defects, age defects as well as package defects. Permanent defects that occur either during manufacturing or during the use of devices and could cause the system to fail permanently.

Different from permanent defects, soft errors, also called transient faults or single-event upsets (SEUs), are due to electrical noises or external radiations [15]. Specifically, many soft errors are caused by high energy neutrons resulting from cosmic rays colliding with particles in the atmosphere. The existence of cosmic ray radiation has been known for over 50 years, and the capacity for this radiation to create transient faults in semiconductor circuits has been studied since the early 1980s [15].

1.2. Related Work

1.2.1. Probabilistic Transfer Matrices (PTMs)

Most faults in nanometric logic circuits are either inherently probabilistic, or can be modeled probabilistically [2]. The signal probability of an input or output of a logic gate is usually defined as the probability that the signal is logical “1”. A logic function transforms its inputs to its output probability. The reliability of an output is defined as the probability of the output with an expected logic value of “1,” or its complement otherwise.

In PTMs, signal probabilities are represented in matrices and signal propagations are formulated as matrix operations. Accurate modeling of reliability makes PTMs an elegant approach for combinational logic circuits [2]. Circuit PTMs that map the probabilistic distribution of primary inputs into that of the primary outputs can be obtained by combining gate PTMs using simple matrix operation rules. Each entry in a circuit PTM represents a joint transition probability between an input and an output combination, and as a result, the circuit PTM scales exponentially with the number of inputs and outputs of a circuit, shown in Figure 1.1. PTMs therefore become impractical when dealing with large scale circuits.

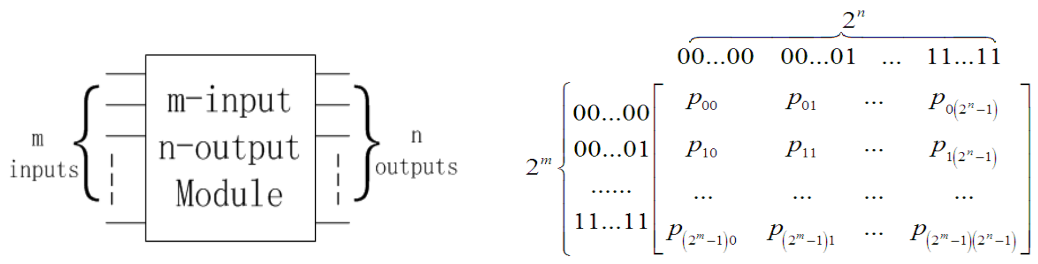

m inputs
m-input n-output Module
n outputs
2^n
00...00 00...01 ... 11...11
2^m
00...00
00...01
......
11...11
p_{00} p_{01} ⋯ $p_{0(2^n-1)}$
p_{10} p_{11} ⋯ $p_{1(2^n-1)}$
$p_{(2^m-1)0}$ $p_{(2^m-1)1}$ ⋯ $p_{(2^m-1)(2^n-1)}$

Figure 1.1 (a) A general combinational circuit with *m* inputs and *n* outputs; (b) PTM of the circuit.

1.2.2. Probabilistic Gate Models (PGMs)

Soft error, or SEU, can be modeled by a von Neumann fault that flips a gate’s correct output with an error rate ε. Therefore, the output of a gate can be described as follows:

$$Z_v = (1-p)\cdot\varepsilon + p\cdot(1-\varepsilon), \quad (1.1)$$

where ε is the gate error rate and p is the probability that a fault-free gate outputs a “1”. A PGM is developed based on (1.1) and describes the relationship between a gate’s output probability, its input probabilities and the gate error rate [3]. Circuit reliability can then be calculated by the executions of PGMs guided by the connectivity of gates. In particular, by identifying and decomposing the reconvergent fanouts following the circuit topology, problems such as signal correlations due to reconvergent fanouts and correlated inputs can be effectively handled. However, with the number of dependent reconvergent fanouts increasing, the PGM algorithm has a computational complexity that increases exponentially [3]. Due to the very large computational overhead, the accurate analysis of large circuits is likely to be impractical and the use of PGM is therefore only limited to small circuits.

1.2.3. Stochastic Computational Models (SCMs)

In stochastic computation, real numbers or probabilities are represented by random binary bit sequences. Typically, signal probabilities are encoded as the proportion of the mean number of 1's in a bit stream. Figure 1.2 illustrates a stochastic encoding [16].

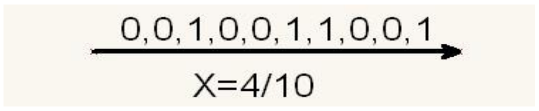


Figure 1.2 A stochastic encoding [16]

In [9], Chen and Han used stochastic computational models (SCMs) to compute the signal probabilities as specified by the PGM equation (1.1). A PGM equation for an arbitrary logic function can be implemented using an SCM with a stochastic logic version of XOR as follows:

$$\mathrm{XOR}_{\mathrm{sto}}(\mathrm{p}, \varepsilon) = \mathrm{p} \cdot (1 - \varepsilon) + (1 - \mathrm{p}) \cdot \varepsilon. \qquad (1.2)$$

Therefore, an SCM can be obtained by adding an XOR gate to an unreliable gate where the gate error is processed by the XOR.

The implementation of the SCM approach is straightforward in that a stochastic computational network can be constructed by adding an XOR gate to each logic gate in a combinational circuit, feeding random input sequences into the network and propagating them from the primary inputs to outputs. Since signal dependencies are encoded in the random distribution of the binary streams, the computational complexity of the SCM approach is significantly reduced. Simulation results in [9] indicate an average error rate of less than 0.1% for the

LGSynth91 benchmarks, compared to the results obtained by the accurate PGM algorithm. It is potentially useful for various VLSI applications.

1.3. Contribution of this Work

With the scaling of CMOS technology into nanometric feature sizes, the reliable operation of FSMs has attracted much concern. In this thesis, we provide the analysis of the operation of sequential circuits, especially the case where the feedback signals in a sequential circuit can be logically masked by specific combinations of primary inputs. With state transition matrices (STMs) and binary decision diagrams (BDDs) of a finite state machine (FSM) model, the so-called error masking phenomenon is extensively analyzed. The analysis of error masking is potentially useful in implementing efficient and robust sequential circuits. For instance, the use of restoring inputs can eliminate the need for an external reset signal and therefore reduce the required numbers of pins and pads in chip packaging.

Arithmetic circuits are critical components for signal processing. As one of the most fundamental functions in arithmetic operations, addition has attracted a lot of research interests. Different adders, such as those based on probabilistic logic and approximate logic, have been proposed for inexact computing to trade off accuracy for energy in recent years. Appropriate metrics to evaluate the efficacy of different adder implementations are thus urgently desired. Several new metrics are proposed in this thesis for evaluating the reliability of adders [18, 19]. These metrics can effectively describe the trade-off between power consumption and precision and can be used to point out a direction for adder design in the future.

Most modern circuits and systems are binary and based on CMOS field effect transistors (FETs). However, multiple valued logic (MVL) circuits have potential advantages in information density and operating speed compared to their binary counterparts. Previous designs of MVL using carbon nanotube field effect transistors (CNTFETs) are either resistor-loaded that require off-chip resistors or complementary designs that require more CNTFETs. Here in my thesis a pseudo-complementary MVL design is proposed for implementations in CNTFETs. Since there are fabrication non-idealities of CNTFETs, a transistor-level reliability analysis method is proposed to accurately estimate the error rates of MVL gates [21]. Further, the stochastic approach is developed to achieve scalability of reliability evaluation for general MVL circuits [20, 21].

Stochastic computation can not only be applied in reliability evaluation of circuits, but also in the modeling of biological systems. A detailed study is applied in modeling genetic regulatory networks (GRNs). Various computational models have been proposed for GRNs previously. In this thesis, a novel implementation of GRNs based on the notions of stochastic logic and stochastic computation is originally proposed. The new structures are referred to as stochastic Boolean networks (SBNs) [22]. The computational complexity of an SBN is effectively reduced, and it is shown that an SBN is able to reproduce biologically-proven regulatory behaviors, and predict the network dynamics when the genes are under perturbation. The algorithms and methods of SBNs have been implemented in Matlab packages and can be applied in general GRN modeling.

1.4. Thesis Outline

In this dissertation, we focus on probabilistic and stochastic models of nanoscale circuits and biological networks. The thesis proceeds as follows. Chapter 2 presents the analysis of error masking/restoring properties of sequential circuits. Chapter 3 discusses the new metrics to evaluate the performance of different inexact adders. Chapter 4 provides designs of multiple-valued logic gates with CNTFETs as well as approaches to evaluating the reliability of these circuits. Chapter 5 demonstrates an application of stochastic computation techniques in modeling biological networks. Chapter 6 concludes the thesis.

CHAPTER 2

Analysis of Error Masking and Restoring Properties of Sequential Circuits*

The scaling of complementary metal–oxide–semiconductor (CMOS) technology down to nanometric feature sizes has raised concerns for the reliable operation of logic circuits. For example, such circuits are expected to be vulnerable to the presence of soft errors. This chapter deals with the analysis of the operation of sequential circuits. As the feedback signals in a sequential circuit can be logically masked by specific combinations of primary inputs, the cumulative effects of soft errors can be eliminated. This phenomenon, referred to as error masking, is related to the presence of so-called restoring inputs and/or the consecutive presence of specific inputs in multiple clock cycles (equivalent to a synchronizing sequence in switching theory). In this chapter, error masking is extensively analyzed using the operations of the state transition matrices (STMs) and binary decision diagrams (BDDs) of a finite state machine (FSM) model. The characteristics of the state transitions with respect to the correlation between the restoring inputs and the time sequence are mathematically established using STMs; although the applicability of the STM analysis is restricted due to its complexity, the BDD approach is more efficient and scalable for use in the analysis of large circuits. These results are supported by simulations of

**A version of this chapter has been accepted for publication [17].*

benchmark circuits and may provide a basis for further devising efficient and robust implementation when designing FSMs.

This chapter is organized as follows. Section 2.1 defines the terminologies used in this chapter. Section 2.2 presents one of the major contributions of this chapter, namely the STM framework for characterizing error masking. Section 2.3 presents an efficient evaluation using BDDs; simulation results are also provided. Section 2.4 concludes the chapter.

2.1. Defintions

Consider a Mealy model of a sequential circuit, as shown in Figure 2.1. In this circuit, there are *m*+*n* inputs: *m* of them are Primary Inputs while the remaining *n* inputs are Present States (i.e., the feedback signals from the flip-flops). There are also *l*+*n* outputs: *l* of them are Primary Outputs, while the remaining *n* outputs are Next States (they will be stored in the flip-flops and then fed back into the combinational logic as Present States during the next clock period).

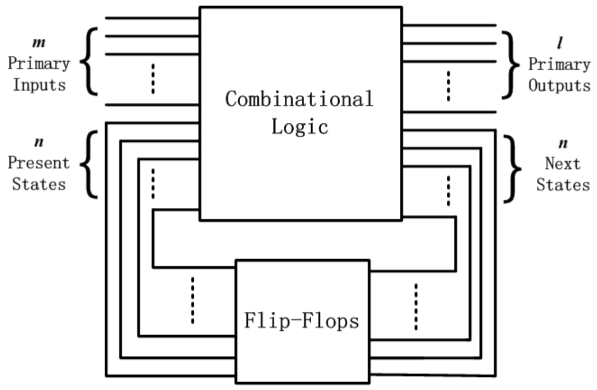


Figure 2.1. Mealy model of a sequential circuit.

A *finite state machine* (FSM) is a classical abstract model for the functions of a sequential circuit. An FSM of the general Mealy model is defined as a six tuple $<I, S, \delta, S_0, O, \lambda>$, where I is the set of inputs, S is the set of states, $\delta: S \times I \to S$ is the next-state function of an input and the present state, $S_0 \subseteq S$ is the set of initial states, O is the set of outputs, and $\lambda: S \times I \to O$ is the output function of an input and the present state [23]. For the sequential circuit of Figure 2.1, *I* is a set of vectors of *m* bits, *O* is a set of vectors of *l* bits, and *S* can be represented by a set of vectors of *n* bits. An FSM can efficiently be described by a state transition graph, in which every node (or vertex) represents a state of the machine and every arc (or directed edge) indicates a state transition and an output.

The state transitions in an FSM can be described by a *state transition matrix* (STM); in the traditional representation, the STM has Boolean entries (0 or 1) to denote the deterministic functions of a sequential circuit. For a probabilistic operation (due to the occurrence of soft errors for instance), the state transitions are described by a *transition probability matrix* [24, 25] due to the underlying Markov nature of the FSM. For the sequential circuit of Figure 2.1, let $I = \{x_0, x_1, \dots, x_{2^m-1}\}$, $S = \{s_0, s_1, \dots, s_{2^n-1}\}$ and $O = \{y_0, y_1, \dots, y_{2^l-1}\}$; the transition probability matrix $\mathbf{\Phi_i}$ is a $2^n \times 2^n$ matrix for a given input vector x_i:

$$\mathbf{\Phi_i} = \begin{bmatrix} p(s_0|s_0) & p(s_1|s_0) & \dots\ p(s_{2^n-1}|s_0) \\ p(s_0|s_1) & p(s_1|s_1) & \dots\ p(s_{2^n-1}|s_1) \\ \dots & \dots & \dots \\ \dots & \dots & \dots \\ p(s_0|\ s_{2^n-1}) & p(s_1|s_{2^n-1}) & \dots\ p(s_{2^n-1}|s_{2^n-1}) \end{bmatrix}, \qquad (2.1)$$

where the *(k, j)* entry $p(s_k|s_j)$ denotes the transition probability from the present state s_j to the next state s_k for the input x_i . For deterministic operations, $p(s_k|s_j) = 0$ or 1 for any *k* and *j*, thus yielding an ideal STM $\mathbf{T_i}$ for the input x_i. Since an STM is unique for every input vector x_i, a total of 2^m STMs are required to describe the operations of the sequential circuit of Figure 2.1.

Next, a cumulative STM (CSTM) is defined; a CSTM, $\mathbf{T_{t_1,t_2}}$, describes the state transitions from time t_1 to t_2 for a sequence of inputs (applied between t_1 and t_2). It is given by:

$$\mathbf{T_{t_1,t_2}} = \prod_{r=t_1}^{t_2} \mathbf{T}(r), \qquad (2.2)$$

where $\mathbf{T}(\mathrm{r})$ (with $t_1 \leq r \leq t_2$) is an STM at time *r* (for a corresponding input). For a sequence of inputs between time *0* and *t-1*, for example, the corresponding CSTM is $\mathbf{T_{0,t-1}}$. Given an initial state, $\boldsymbol{s}(\mathbf{0})$, the state at a subsequent time t can be computed as:

$$\boldsymbol{s}(\boldsymbol{t}) = \boldsymbol{s}(\mathbf{0}) * \mathbf{T_{0,t-1}}. \tag{2.3}$$

Similar matrices can be defined for the transition probabilities between the present states and the outputs. These matrices are referred to as *output transition matrices*. The STMs and the transition probability matrices are essentially equivalents of the ideal transfer matrices (ITMs) and probabilistic transfer matrices (PTMs) [26, 27], so they can be constructed by extending and combining the gate ITMs and PTMs (as applicable to combinational circuits) to the topology and operation of a sequential circuit. Transition probability matrices have also been used for the Markovian analysis of FSMs [28] and fault-tolerant systems [29].

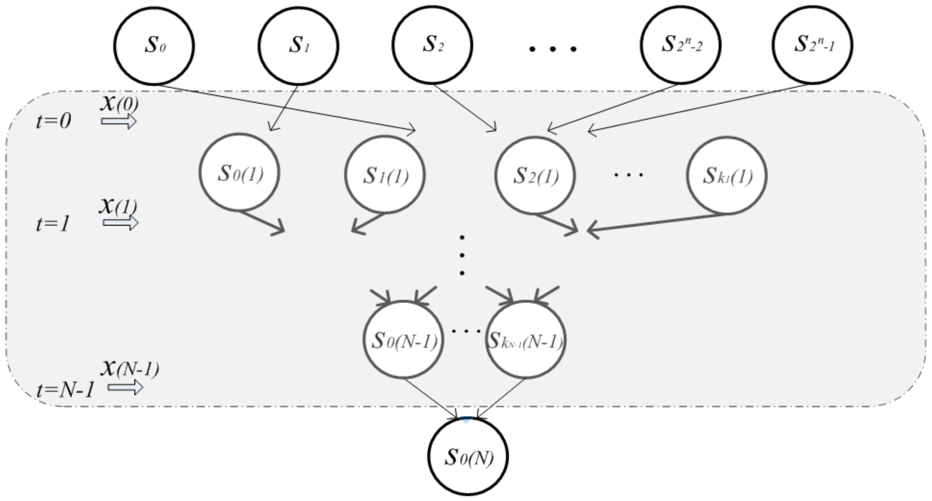


Figure 2.2. State transition diagram for an N-step error masking scenario. An arrow indicates a state transition for a given input (shown on the left).

In a sequential circuit, the *restoring inputs* are the primary inputs or a sequence of primary inputs that logically mask the feedback signals. *Error masking* in a sequential circuit refers to the phenomenon that the feedback signals are logically masked by specific sequences of primary inputs (i.e., the restoring inputs). This error masking can occur in one or multiple steps. An N-step error masking scenario is illustrated in the state transition diagram of Figure 2.2. Assume that the state of an FSM at *t=0* is not deterministic, but probabilistic (possibly due to the effects of soft errors in a sequential circuit); so in principle, it can take any of the 2^n states, as shown in the first row in Figure 2.2. However, this state space can be reduced at later steps as result of the state transition properties of the FSM. This is determined by the STMs and thus the primary input at each step. If this state space is reduced to one that has only one single state after N steps, then the initial state at time t_0 becomes *irrelevant* for determining the final state; hence, any initial error would be masked by this N-step transition process. The inputs that result in the occurrence of this error masking scenario are a sequence of restoring inputs. In a state transition graph, this is indicated by various state transition paths (represented by directed edges) that eventually lead to the same destination state (represented by a vertex). Therefore, a sequential circuit is said to be *reliable* if error masking frequently occurs; it is *unreliable* otherwise.

2.2. Error Masking

2.2.1. Restoring inputs

In switching theory it is well known that among all output combinations of a sequential circuit, some are determined only by the primary inputs, and not by the feedback signals (or present states). This property is very useful as these inputs can be utilized for determining the values of the outputs and therefore voids the cumulative effects of errors. As defined previously, these inputs are called *restoring inputs*. A circuit is considered *reliable* if these restoring inputs frequently appear. In a reliable circuit, when the feedback signals are logically masked (i.e. not relevant for determining the next state or output values), the cumulative effects of errors are effectively mitigated and/or possibly eliminated.

The next-state function in an FSM is particularly important as it determines whether an accumulation of errors could occur. If the next state is always fully or partially determined by the present state, then the design is considered as unreliable, i.e., errors and their effects through the feedback signals will accumulate. An unreliable design will fail with a high probability after a sufficiently long time. Hence, a Moore machine in which the outputs are only determined by the present states (or feedback signals), tends to be unreliable in the presence of random soft errors. This is consistent with the homogeneity in the Markov characterization of the FSMs.

For nanoscale computing, the rate of an error is projected to be finite but small, so the next state is expected to be ideal with a very high probability. In a sequential circuit, therefore, a transition probability matrix is expected to have entries that are approximately 0's and 1's. This leads to the *convergence* of the transition probability matrix and its ideal STM. In fact, the STMs contain original information on the distinctive features of a circuit, so they are of fundamental importance when determining the restoring inputs.

$$
T_i = \begin{array}{c|cccccccc|}
 & \multicolumn{8}{c}{\overbrace{\hspace{9cm}}^{NS}} \\
\underbrace{PS} & 0\ldots00 & 0\ldots01 & \ldots & \ldots & \ldots & \ldots & \ldots & 1\ldots11 \\
\hline
00\ldots000 & \ldots & \ldots & \ldots & \ldots & \ldots & 1 & \ldots & \ldots \\
00\ldots001 & \ldots & \ldots & \ldots & \ldots & \ldots & 1 & \ldots & \ldots \\
00\ldots010 & \ldots & \ldots & \ldots & \ldots & \ldots & 1 & \ldots & \ldots \\
00\ldots011 & \ldots & \ldots & \ldots & \ldots & \ldots & 1 & \ldots & \ldots \\
00\ldots100 & \ldots & \ldots & \ldots & \ldots & \ldots & 1 & \ldots & \ldots \\
\ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots \\
11\ldots110 & \ldots & \ldots & \ldots & \ldots & \ldots & 1 & \ldots & \ldots \\
11\ldots111 & \ldots & \ldots & \ldots & \ldots & \ldots & 1 & \ldots & \ldots
\end{array}
$$

Figure 2.3. An STM indicating the existence of a restoring input. 'T_i' is the STM for the ith input. 'PS' denotes the present state and 'NS' denotes the next state.

Consider as an example the STM shown in Figure 2.3 for an input in the set I_0. In this STM, if the entries in a column, which indicate the transitions from the present states to a specific next state, are all 1's, then the primary inputs for this STM (i.e., those in the set I_0) are a set of restoring inputs. A similar procedure can be applied to the analysis of output transition matrices, which characterizes whether restoring inputs exist to at least partially eliminate the accumulated effects of errors to the primary outputs.

2.2.2. Error masking in multiple steps

Restoring inputs can appear in single and multiple steps (or clock cycles). For a single step, the STM for a restoring input is expected to have all 1's in a column. For multiple steps, restoring inputs can be found by analyzing the CSTM obtained by (2), i.e., the product of STMs at these steps.

2.2.2.1 Two-step process

As an example, the two-step case will be first presented for establishing the conditions such that the restoring inputs exist. Let $\mathbf{T_u}$ and $\mathbf{T_v}$ be the two $2^n \times 2^n$ STMs involved in a two-step operation (for inputs x_u and x_v respectively). If x_u and x_v are the restoring inputs for the jth next state, the two-step CSTM is given by

$$\mathbf{T_u} * \mathbf{T_v} = \begin{array}{c} \begin{array}{cccccccc} 0 & \ldots & \mathrm{j}-1 & \mathrm{j} & \mathrm{j}+1 & \ldots & 2^{\mathrm{n}}-1 \end{array} \\ \begin{bmatrix} 0 & \vdots & 0 & 1 & 0 & \vdots & 0 \\ 0 & \vdots & 0 & 1 & 0 & \vdots & 0 \\ 0 & \vdots & 0 & 1 & 0 & \vdots & 0 \\ \ldots & \vdots & \ldots & \ldots & \ldots & \vdots & \ldots \\ 0 & \vdots & 0 & 1 & 0 & \vdots & 0 \end{bmatrix} \end{array}. \tag{2.4}$$

Here, "*" means conventional matrix multiplications.

Let the elements in the ith row and the jth column in $\mathbf{T_u}$ and $\mathbf{T_v}$ be given by a_{ij} and b_{ij} respectively ($0 \le i \le 2^n - 1$ and $0 \le j \le 2^n - 1$), then

$$\mathbf{T_u} * \begin{bmatrix} \mathrm{b}_{0\mathrm{j}} \\ \mathrm{b}_{1\mathrm{j}} \\ \mathrm{b}_{2\mathrm{j}} \\ \cdots \\ \mathrm{b}_{(2^{\mathrm{n}}-1)\mathrm{j}} \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ 1 \\ \cdots \\ 1 \end{bmatrix}. \qquad (2.5)$$

Also, the following is always applicable

$$\mathbf{T_u} * \begin{bmatrix} 1 \\ 1 \\ 1 \\ \cdots \\ 1 \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ 1 \\ \cdots \\ 1 \end{bmatrix}. \qquad (2.6)$$

Subtracting (2.5) from (2.6) gives

$$\mathbf{T_u} * \begin{bmatrix} 1-\mathrm{b}_{0\mathrm{j}} \\ 1-\mathrm{b}_{1\mathrm{j}} \\ 1-\mathrm{b}_{2\mathrm{j}} \\ \cdots \\ 1-\mathrm{b}_{(2^{\mathrm{n}}-1)\mathrm{j}} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ \cdots \\ 0 \end{bmatrix}. \qquad (2.7)$$

Since a_{ij} and b_{ij} can be either 0 or 1 for any *i* and *j*, there is only one element in each row of $\mathbf{T_u}$ being 1, due to the deterministic operation in the state transitions.

Assume for any *i*,

$$a_{ic_i} = 1. \qquad (2.8)$$

Then by (2.7),

$$1 - b_{c_i j} = 0 \text{ or } b_{c_i j} = 1. \qquad (2.9)$$

Both (2.8) and (2.9) determine the positions of 1’s in the two STMs that establish the conditions for the restoring inputs in two steps.

Moreover, (2.8) and (2.9) can be used to analyze two extreme conditions.

If $c_i \neq c_j \, (\forall \, i \neq j)$, the following condition must be applicable for a two-step masking:

$$b_{0j} = b_{1j} = b_{2j} = \ldots b_{(2^n-1)j} = 1. \tag{2.10}$$

This implies that if the first step does not contribute to error masking, then the second step must solely contribute to masking. Consider the second extreme condition; if $c_0 = c_1 = c_2 = \cdots = c_{2^n-1} = k$, then for a two-step masking it is only required that,

$$b_{kj} = 1 \, . \tag{2.11}$$

Of course, this corresponds to the opposite case of the first condition.

To better understand the relationship between $\mathbf{T_u}$ and $\mathbf{T_v}$ in a two-step restoring process, an example is presented next.

Example I: Let $\mathbf{T_{u_0}}$ and $\mathbf{T_{v_0}}$ be two 4x4 STMs, as shown in Figure 2.4. The Boolean digit under each column of a matrix indicates whether the next state corresponding to this column is possible (value of 1) or not (value of 0) after each step. For example, a 1 for the first and third columns of $\mathbf{T_{u_0}}$ indicates that the first and third next states (i.e., “00” and “10”) are the two possible next states after the first step of operation. Since the output from the first step serves as the present state for the second step, only the possible next states from the first step are still relevant and should be considered in the second step. In this example, these next states are the first and third, i.e., “00” and “10”. They determine that only the first and third rows of $\mathbf{T_{v_0}}$ are relevant in the second step of operation (all other rows are masked in the first step). This is illustrated in the second row of Figure 2.4, where the relevant columns are marked in red. Subsequently, if the 1’s in the remaining rows of $\mathbf{T_{v_0}}$belong to a single column, this two-step process is then expected to yield a single next state corresponding to this column, regardless of the present states. Therefore, a two-step error masking occurs.

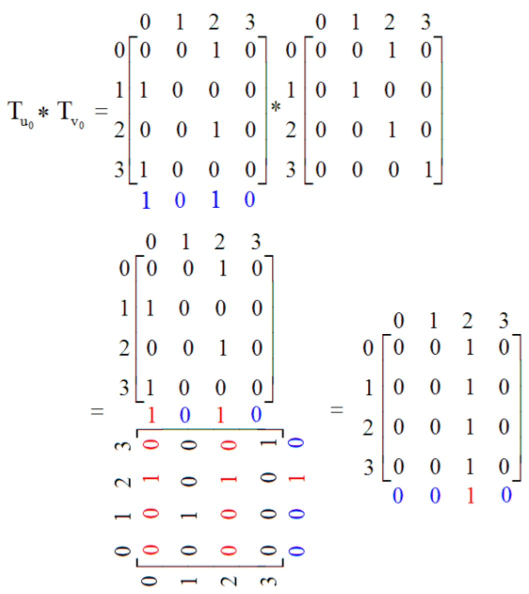


Figure 2.4. Example of a two-step error masking.

2.2.2.2 N-step process

In the general case that restoring occurs in *N* steps, the CSTM is given by

$$\mathbf{T}_{\mathbf{u}_1} * \mathbf{T}_{\mathbf{u}_2} * \ldots * \mathbf{T}_{\mathbf{u}_N} = \begin{array}{c} \begin{array}{cccccccc} 0 & \ldots & j-1 & j & j+1 & \ldots & 2^n-1 \end{array} \\ \begin{bmatrix} 0 & \vdots & 0 & 1 & 0 & \vdots & 0 \\ 0 & \vdots & 0 & 1 & 0 & \vdots & 0 \\ 0 & \vdots & 0 & 1 & 0 & \vdots & 0 \\ \ldots & \vdots & \ldots & \ldots & \ldots & \vdots & \ldots \\ 0 & \vdots & 0 & 1 & 0 & \vdots & 0 \end{bmatrix} \end{array}, \tag{2.12}$$

as applicable to a restoring to the *j*th next state.

Due to the state-transition property, each row in $\mathbf{T}_{\mathbf{u}_i}$ $(1 \le i \le N)$ has only one 1 and all other elements are 0. Assume that $\mathbf{r}_{\mathbf{ij}} =$ （0,0 ... 0,1,0 ... 0,0) is the *j*th row in $\mathbf{T}_{\mathbf{u}_i}$, then

$$\mathbf{T}_{\mathbf{u}_i} = \begin{bmatrix} \mathbf{r}_{\mathbf{i0}} \\ \mathbf{r}_{\mathbf{i1}} \\ \mathbf{r}_{\mathbf{i2}} \\ \ldots \\ \mathbf{r}_{\mathbf{i(2^n-1)}} \end{bmatrix}. \tag{2.13}$$

Further, let c_{ij} be the column index of the 1 in $\mathbf{r_{ij}}$;

Then for the first two steps,

$$\mathbf{T_{u_1}} * \mathbf{T_{u_2}} = \begin{bmatrix} \mathbf{r_{10}} \\ \mathbf{r_{11}} \\ \mathbf{r_{12}} \\ \cdots \\ \mathbf{r_{1(2^n-1)}} \end{bmatrix} * \begin{bmatrix} \mathbf{r_{20}} \\ \mathbf{r_{21}} \\ \mathbf{r_{22}} \\ \cdots \\ \mathbf{r_{2(2^n-1)}} \end{bmatrix} = \begin{bmatrix} \mathbf{r_{2c_{10}}} \\ \mathbf{r_{2c_{11}}} \\ \mathbf{r_{2c_{12}}} \\ \cdots \\ \mathbf{r_{2c_{1(2^n-1)}}} \end{bmatrix}. \qquad (2.14)$$

For the N steps, the CSTM is

$\mathbf{T_{u_1}} * \mathbf{T_{u_2}} * \ldots * \mathbf{T_{u_N}}$

$$= \begin{bmatrix} \mathbf{r_{10}} \\ \mathbf{r_{11}} \\ \mathbf{r_{12}} \\ \cdots \\ \mathbf{r_{1(2^n-1)}} \end{bmatrix} * \begin{bmatrix} \mathbf{r_{20}} \\ \mathbf{r_{21}} \\ \mathbf{r_{22}} \\ \cdots \\ \mathbf{r_{2(2^n-1)}} \end{bmatrix} * \ldots * \begin{bmatrix} \mathbf{r_{N0}} \\ \mathbf{r_{N1}} \\ \mathbf{r_{N2}} \\ \cdots \\ \mathbf{r_{N(2^n-1)}} \end{bmatrix} = \begin{bmatrix} \mathbf{r}_{\mathbf{N}\left(\mathbf{c}_{(\mathbf{N-1})\ldots\ldots(\mathbf{c}_{2(\mathbf{c}_{10})})}\right)} \\ \mathbf{r}_{\mathbf{N}\left(\mathbf{c}_{(\mathbf{N-1})\ldots\ldots(\mathbf{c}_{2(\mathbf{c}_{11})})}\right)} \\ \mathbf{r}_{\mathbf{N}\left(\mathbf{c}_{(\mathbf{N-1})\ldots\ldots(\mathbf{c}_{2(\mathbf{c}_{12})})}\right)} \\ \cdots \\ \mathbf{r}_{\mathbf{N}\left(\mathbf{c}_{(\mathbf{N-1})\ldots\ldots(\mathbf{c}_{2\left(\mathbf{c}_{1(2^n-1)}\right)})}\right)} \end{bmatrix}, \qquad (2.15)$$

where $c_{(N-1)\ldots\ldots(c_{2(c_{1i})})}$ with $0 \leq i \leq 2^n - 1$, determined by the positions of 1's in the STMs of previous steps, gives the index of the row vector in $\mathbf{T_{u_N}}$.

If error masking occurs, then all the 1's are in the same column in the matrix obtained by (2.15), so all the rows of the matrix are the same, i.e.,

$$\mathbf{r}_{\mathbf{N}\left(\mathbf{c}_{(\mathbf{N-1})\ldots\ldots(\mathbf{c}_{2(\mathbf{c}_{10})})}\right)} = \mathbf{r}_{\mathbf{N}\left(\mathbf{c}_{(\mathbf{N-1})\ldots\ldots(\mathbf{c}_{2(\mathbf{c}_{11})})}\right)} = \mathbf{r}_{\mathbf{N}\left(\mathbf{c}_{(\mathbf{N-1})\ldots\ldots(\mathbf{c}_{2(\mathbf{c}_{12})})}\right)} = \cdots = \mathbf{r}_{\mathbf{N}\left(\mathbf{c}_{(\mathbf{N-1})\ldots\ldots(\mathbf{c}_{2\left(\mathbf{c}_{1(2^n-1)}\right)})}\right)}, \qquad (2.16)$$

and equivalently, the column indices of 1's are the same, i.e.,

$$\mathrm{c}_{\mathrm{N}\left(\mathrm{c}_{(\mathrm{N-1})\ldots\ldots(\mathrm{c}_{2(\mathrm{c}_{10})})}\right)} = \mathrm{c}_{\mathrm{N}\left(\mathrm{c}_{(\mathrm{N-1})\ldots\ldots(\mathrm{c}_{2(\mathrm{c}_{11})})}\right)} = \mathrm{c}_{\mathrm{N}\left(\mathrm{c}_{(\mathrm{N-1})\ldots\ldots(\mathrm{c}_{2(\mathrm{c}_{12})})}\right)} = \cdots =$$

$$c_{N}\left(c_{(N-1)\ldots\ldots}\left(c_{2}\left(c_{1(2^{n}-1)}\right)\right)\right) \tag{2.17}$$

For a two-step restoring process (as given by (2.4)), it can be obtained from (2.17) that

$$c_{2c_{10}} = c_{2c_{11}} = c_{2c_{12}} = \cdots = c_{2c_{1(2^n-1)}} = j, \tag{2.18}$$

i.e., as $a_{ic_{1i}} = 1$ and $c_{2c_{1i}} = j$, we have $b_{c_{1i}j} = 1$, for any $0 \leq i \leq 2^n - 1$.

Hence, (2.16) and (2.17) reveal the underlying relationships among multiple STMs, as required for obtaining the restoring inputs. STMs are sparse matrices, so these relationships can be used to efficiently analyze *N*-step error masking.

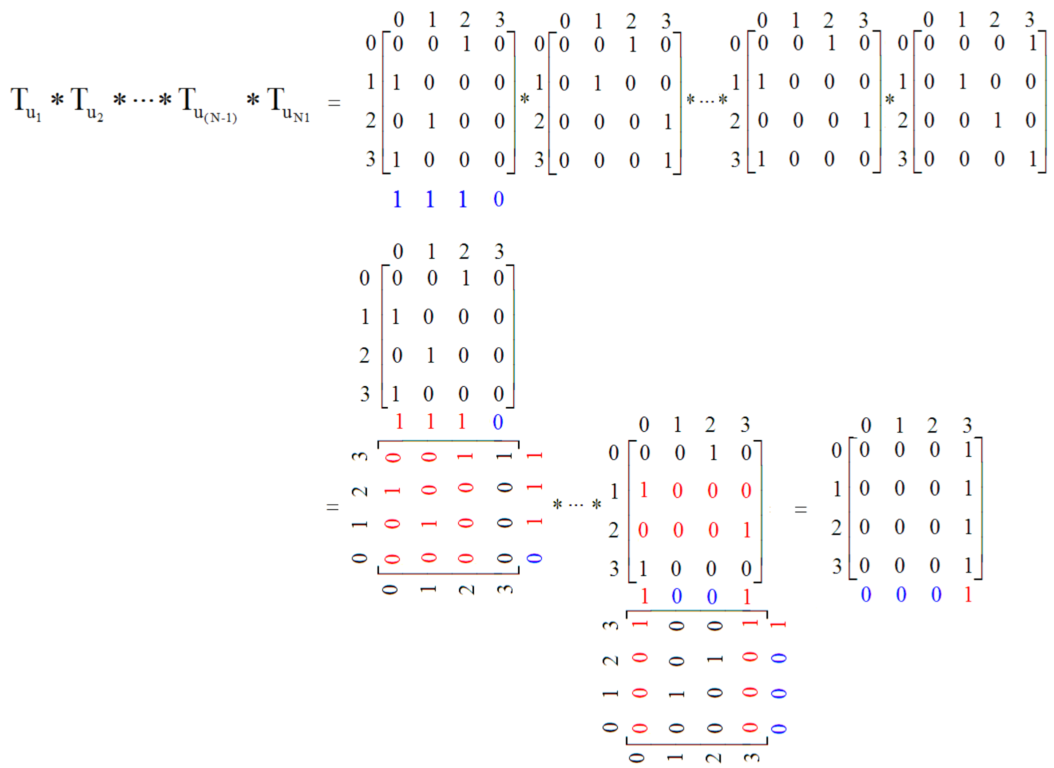


Figure 2.5. Example of N-step error masking.

Similar to the case of two-step masking presented previously, an example is given in Figure 2.5. It reveals the mappings between the STM elements (as characterized by (2.16) and (2.17)) and the accumulating effect of error masking in an *N*-step process.

For an unreliable design, fault-tolerant and error-mitigation techniques can be used to reduce the error effects [30, 31]. A simple method to accomplish this objective is to reset a sequential system periodically as this may help to recover from the accumulated errors. This however may not be possible in all applications due to the disruption of normal circuit operation caused by the reset.

A possible solution to this problem is to use approximate logic to change the truth table, thus introducing error masking into the circuit. This will also introduce a tradeoff in the precision of the computed function and hence is applicable to inexact computing (also often referred to as soft computing), attaining more masking and less error accumulation.

2.2.3. Case study: s27, ISCAS'89 benchmark circuit

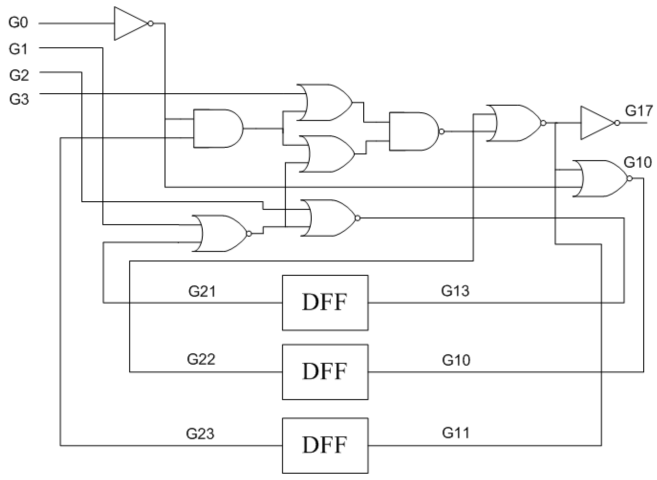


Figure 2.6. Schematic diagram of s27.

The ISCAS'89 benchmark circuit s27 is considered in this section using the STM analysis to substantiate the characterization of error masking. s27 is a circuit with four primary inputs, three D flip-flops and one primary output (Figure 2.6). When error masking occurs, the next state is totally dependent on the primary inputs, but not on the feedback (present state). This is applicable to s27 and shown as follows.

For s27, its STM is an 8x8 matrix. Consider the STM $\mathbf{T}_{14}$ for the input “1110,” as shown in Figure 2.7; the next state (NS) is expected to be “110” regardless of the present state (*PS*). So when “1110” appears at the primary input, errors may have accumulated but are logically masked; therefore the reliability increases. An error masking STM can also be a result of several steps of STM operations. For example, neither the input “1001” nor “0110” is a restoring input; however, the synergetic effect of the consecutive presence of these two inputs leads to a two-step error masking.

$$
T_{14} = \begin{array}{c|cccccccc|}
\overbrace{PS} & 000 & 001 & 010 & 011 & 100 & 101 & 110 & 111 \\
\hline
000 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\
001 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\
010 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\
011 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\
100 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\
101 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\
110 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\
111 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0
\end{array}
\quad \text{(columns: } \overbrace{NS}\text{)}
$$

Figure 2.7. $\mathbf{T}_{14}$ of s27 is the STM for the 14th primary input (i.e., “G0G1G2G3”=”1110”). “PS” denotes the present state (G21, G22 and G23) and “NS” denotes the next state (G13, G10 and G11).

Table 2.1 shows the probability of occurrence of multiple-step error masking in s27; this probability is given by the ratio of the number of input sequences causing error masking over the total number of input sequences. For example, 34 out of the total 256 input sequences result in error masking for the two-step operation. Note that an *N*-step error masking process means that at least *N* steps are required to ensure error masking, i.e. if error masking occurs in *M* steps

(*M*<*N*), then this is classified as an *M*-step masking and is not considered as a part of the *N*-step masking process.

Table 2.1. Simulation results for s27. The probability that multiple-step error masking occurs is given as the ratio between the number of restoring inputs and the total number of inputs.

Number of steps	**1**	**2**	**3**	**4**	**5**
Error masking probability	5/16	34/256	60/4096	92/65536	136/1048576
Run time (s)	0.0055	0.0159	0.2019	3.7354	72.3031
Memory (MByte)	0.1	0.3	1.1	2.1	2.8

The run time and memory usage of the STM analysis are also shown in Table 2.1. Although the memory usage steadily increases with the number of error masking steps, the run time changes more drastically because a significantly increased number of inputs must be considered in a multiple step masking. This makes the analysis of large circuits difficult, if not impossible. The complexity issues are further discussed in Section 2.3.5.

2.2.4. Partial error masking

When errors only affect some of the state bits, their effects can be affected by so-called *partial error masking*. Partial error masking refers to the phenomenon in which some of the feedback signals are logically masked by specific combinations of primary inputs and other feedback signals. Consider the sequential circuit model of Figure 2.1; if some of the present state signals are unreliable, then their error effects can be masked by a combination of the other present state signals and a primary input. In the STM for such a primary input, this means that the rows can be re-ordered such that the unreliable state bits are next to each other and the corresponding adjacent rows lead to the same next state. This process is shown in Figure 2.8.

$$
T_i = \begin{array}{c|cccccccc|}
\underbrace{PS}_{} & \multicolumn{8}{c}{\overbrace{0\ldots00 \quad 0\ldots01 \quad \ldots \quad \ldots \quad \ldots \quad \ldots \quad \ldots \quad 1\ldots11}^{NS}} \\
00\ldots000 & \ldots & \ldots & \ldots & \ldots & \ldots & 1 & \ldots & \ldots \\
\ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots \\
1000\cdots0 & \ldots & \ldots & 1 & \ldots & \ldots & \ldots & \ldots & \ldots \\
1001\cdots0 & \ldots & \ldots & 1 & \ldots & \ldots & \ldots & \ldots & \ldots \\
1010\cdots0 & \ldots & \ldots & 1 & \ldots & \ldots & \ldots & \ldots & \ldots \\
1011\cdots0 & \ldots & \ldots & 1 & \ldots & \ldots & \ldots & \ldots & \ldots \\
\ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots \\
11\ldots111 & \ldots & \ldots & \ldots & 1 & \ldots & \ldots & \ldots & \ldots
\end{array}
$$

Figure 2.8. An STM with the occurrence of partial error masking. In T_i (i.e., the STM for the ith input), the third and fourth bits in the present state (PS) are masked by the combinations of the ith input and the remaining present-state signals, shown in the four rows in the middle of the STM.

The analytical procedure outlined in Section 2.3.2 is applicable also to multiple-step partial error masking; however, since only a subset of the state signals are of interest, a sub-matrix of each STM is needed in the analysis. Similarly, as partial error masking also applies to output signals, therefore it can be analyzed using the output transition matrices of the sequential circuit.

2.2.5. Complexity

The analysis using STMs reveals the fundamental mechanism of error masking. The mapping relationships given in (2.16) and (2.17) can be used for an optimized analysis by leveraging the fact that STMs are sparse matrices. Nevertheless, this analysis incurs a large computational complexity for finding the restoring inputs. Consider the circuit model of Figure 2.1 as an example; there are a total of 2^m STMs. To find the restoring inputs in an N-step process, a total of $(2^m)^N$ CSTMs need to be examined, thus resulting in a computational complexity of at least $O(2^{mN})$. This computation is of course not scalable for analysing large circuits. In the next section, an approach using BDDs is proposed for a more efficient analysis.

2.3. Analysis using BDDs

A binary decision diagram (BDD) is a canonical (or unique) representation of a Boolean function [32]. It is also efficient at representing a large combinatorial set. BDDs have been shown to be effective in many applications involving FSMs [23]. In this section, a computationally-efficient technique employing BDDs is used for analyzing error masking in sequential circuits. The CUDD package has been used throughout this study [33].

2.3.1. Finding the restoring inputs

A BDD is a directed acyclic graph, in which each node represents a variable and each edge is labeled "True" or "False" (or, "1" or "0"). The edges lead to leafs labelled "1" or "0" at the bottom of the graph. For the sequential circuit model of Figure 2.1, a BDD can be generated for every variable (or bit) of the next state (and the primary output) as a function of the primary inputs and the present state. Therefore, a total number of *n* diagrams must be generated for the *n* variables in the next state. The variables in a BDD are usually ordered to find an optimal diagram; this ordering is typically done heuristically by relying on specific features of the system being analyzed [34]. To find the restoring inputs, a special ordering is imposed such that the variables in the primary inputs are first analyzed, followed by those in the present state. The BDDs generated for the next state of the benchmark circuit s27 are shown in Figure 2.9.

In a BDD, if there exists a path that starts from the root (or a primary input) and reaches a leaf "1" or "0", without traversing through any present state variable, then the primary input dictated by this path is a restoring input for this variable in the next state. Given the BDDs for the other variables in the next state, the restoring inputs for those variables can be found in the same way. The restoring inputs for a circuit are then obtained as the intersection of the set of restoring inputs for each variable in the next state. This establishes the conditions for a single-step error masking. For partial error masking, a similar procedure applies when only a subset of the present state variables are considered as required by nature of the partial masking process.

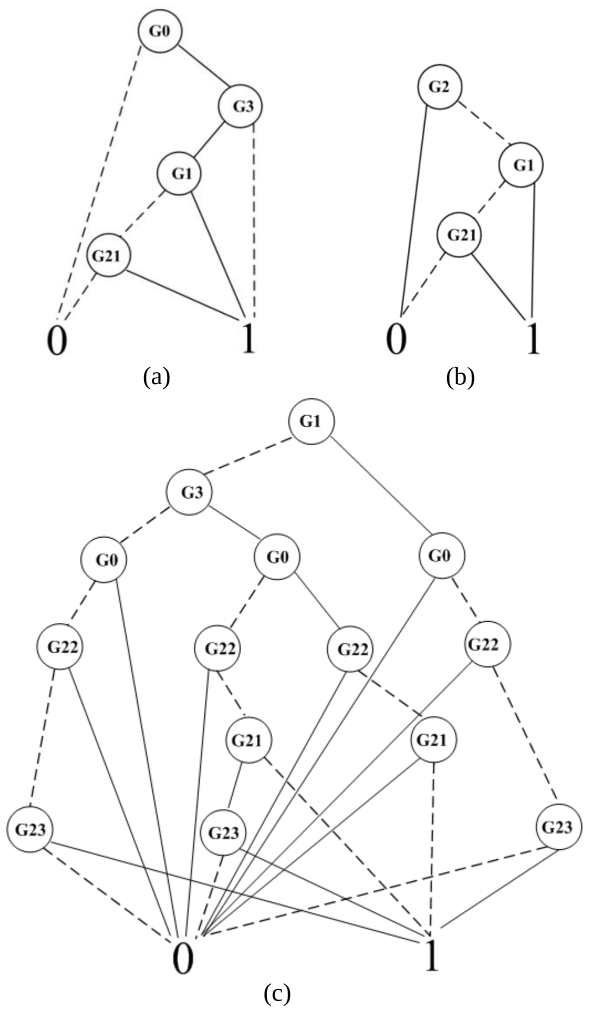


Figure 2.9. BDDs for the next state of s27: (a) G10, (b) G13, (c) G11.

Example II: Consider s27 again. Figure 2.9(b) shows the BDD for the variable G13 in the next state. In this BDD, G21 is the only existing present state variable and two paths starting from the primary input reach the end leafs without traversing G21. These paths correspond to the primary inputs G2=”1” and G2G1=”01,” or equivalently, G0G1G2G3 =”XX1X” and “X10X,” where “X” denotes the don’t-care condition. Similar analysis can be performed for G10 and G11, and the results are shown in the second row of Table 2.2. The restoring inputs for s27 can then be obtained as the intersection of the three sets of inputs as shown in the third row of Table 2.2. This confirms the results obtained by the STM analysis in Table 2.1.

Table 2.2. Restoring inputs obtained by the BDD analysis for s27; "X" denotes a don't-care value.

The next state variable	G10	G11	G13
Restoring inputs for each variable (G0G1G2G3)	0XXX 1XX0 11X 1	11XX 10X0	XX1X X10X
Restoring inputs for s27	1010, 1100, 1101,1110,1111		

2.3.2. Multiple-step error masking

For a multiple-step operation in the temporal domain, the so-called time-frame expansion technique can be used to convert the operation into a single step process, as proposed for the soft error analysis of sequential circuits in [35]. This is illustrated in Figure 2.10. For an *N*-step operation, the present states from the second to the *N*th steps are treated as internal signals; only the initial present state serves as the present state of the expanded iterative circuit, while all the primary inputs in the N steps become the primary inputs of the new circuit. The BDD analysis proposed in the previous subsection can then be used for a multiple-step error masking analysis.

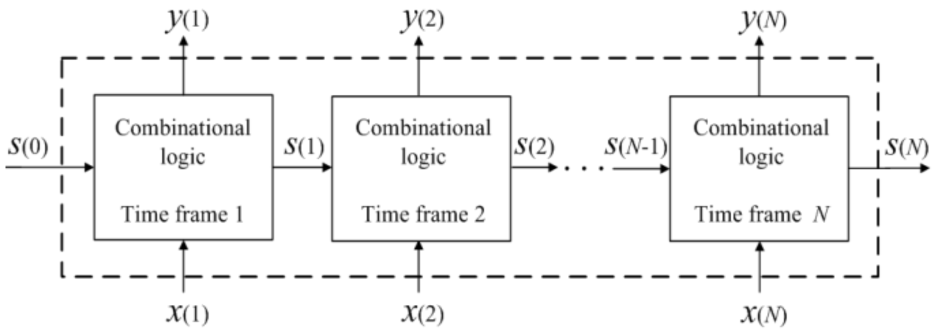


Figure 2.10. Time frame expansion of a sequential circuit. The N frame expansion of a sequential circuit can be treated as a single sequential circuit with x(1), x(2),…,x(N) as inputs, y(1), y(2),…,y(N) as outputs, s(0) as the present state and s(N) as the next state.

2.3.3. Simulation results

The CUDD package [33] was used to generate the BDDs from the netlist of a circuit. A customized program was then written for extracting the restoring inputs from the BDDs. Table 3 shows the simulation results for the ISCAS'89 sequential benchmark circuits for finding (if any) single-step restoring inputs using the proposed BDD method. These benchmarks have also been used in [14] and [24] for SER analysis. A single-step error masking mostly occurs due to the presence of a "reset" signal, as observed for s382, s400, s444, s526, s820, s832 and s1488. Although other single-step restoring inputs are present in some circuits (such as s27, as discussed previously, and s1196), they do not always exist in a sequential circuit; this is generally due to the feature of a sequential circuit by which the next state is determined by both the primary inputs and the present state. Even though beyond the scope of this chapter, don't-care values in the logic definition of a sequential circuit could be assigned through synthesis to implement restoring inputs, thus avoiding the additional implementation of the "reset" signal as well as improving error masking.

Table 2.3. The number of single-step restoring inputs (No.) for ISCAS'89 benchmark circuits found using BDDs with runtime (T) and memory usage (M). The runtime includes the time for generating the BDDs and extracting the restoring inputs; the memory usage is for the use of the CUDD package in producing BDDs.

Circuits	Gates	Inputs	Outputs	FFs	No.	T (s)	M (MByte)
s27	10	4	1	3	5	0.8147	1.068
s382	158	3	6	21	4	1.6324	1.089
s400	164	3	6	21	4	1.9575	1.089
s444	181	3	6	21	4	2.1576	1.089
s526	193	3	6	21	4	1.8003	1.089
s820	289	18	19	3	2^{17}	335.79	1.172
s832	287	18	19	5	2^{17}	389.91	1.172
s953	395	16	23	29	0	23.098	1.160
s1196	529	14	14	18	4890	79.965	1.166
s1488	653	8	19	6	128	15.971	1.145

The runtime and memory usage required by this approach are also reported in Table 2.3. The runtime includes the time for generating BDDs and extracting the restoring inputs, while the memory usage is only for using the CUDD package to produce the BDDs. While the memory usage is relatively stable for different circuits, the runtime is largely affected by the number of restoring inputs that must be extracted from the BDDs. Typically, it takes no more than a few seconds to generate BDDs for circuits of this size.

To find a multiple-step restoring input, the netlist of a time frame extended circuit was first produced. Table 2.4 shows the results of multiple-step error masking; the reported runtime and memory usage further confirm the efficiency of the proposed BDD method. In these cases, the next state of the circuit is determined by a multiple clock-cycle state dependency. This process is more complicated than a single-step masking as it implies that a time domain overhead will be incurred in the masking process due to the inherent latency. Hence, a designer is confronted with a tradeoff assessment of achieving error masking with a smaller number of cycles (or simply in one step) versus additional design complexity in the implementation of a sequential circuit.

The error masking phenomena observed for some circuits, such as the semaphore circuits s382 and s400, result from the “reset” signal. This occurs because the next state of a semaphore circuit is always determined by its present state unless the circuit is reset. Binary counters, whose next state is totally determined by the present state, also exhibit this property. Therefore, these features should be considered by designers when assessing the reliable operation of these types of circuits.

2.4. Summary

This chapter analyzes the reliable operation of sequential circuits in the presence of errors as likely to occur at nanometric feature sizes. The major contribution of this chapter is the analysis of the phenomenon (referred to as “error masking”) that affects the reliability of a sequential circuit, by utilizing the state transition matrices (STMs) and the binary decision diagrams (BDDs) in an FSM model.

Table 2.4. The number of restoring inputs of benchmark circuits found using BDDs (No.) and the required memory usage (M) and runtime (T). Both the numbers of restoring inputs for up to N-step masking and for only N-step masking are reported (separated by '/'). The memory is from the use of the CUDD package in producing the BDDs and the runtime includes the time for generating the BDDs and extracting the restoring inputs.

Circuits	N=1			N=2			N=3			N=4			N=5		
	No.	**M (MByte)**	**T (s)**	**No.**	**M (MByte)**	**T (s)**	**No.**	**M (MByte)**	**T (s)**	**No.**	**M (MByte)**	**T (s)**	**No.**	**M (MByte)**	**T (s)**
lion	0/0	1.068	0.0743	1/1	1.068	0.3922	13/5	1.068	1.1712	88/0	1.079	4.0344	476/0	1.079	8.6948
train4	1/1	1.068	0.0609	7/0	1.068	0.7547	37/0	1.068	1.6463	175/0	1.079	7.4456	781/0	1.079	12.490
train11	0/0	1.068	0.0186	0/0	1.068	0.4387	2/2	1.068	1.3816	21/5	1.079	4.7655	154/20	1.079	7.7952
dk27	0/0	1.068	0.0224	0/0	1.068	0.5497	0/0	1.068	1.7537	2/2	1.068	2.0119	9/1	1.068	3.5688
s8	0/0	1.068	0.0585	9/9	1.079	1.5853	552/282	1.079	8.3371	16978 /1042	1.079	21.531	399238 /4454	1.089	48.779
tav	0/0	1.068	0.0498	0/0	1.068	0.8308	0/0	1.079	1.0540	0/0	1.079	3.1299	0/0	1.089	4.0540
bbtas	0/0	1.068	0.1190	0/0	1.068	0.3517	1/1	1.068	1.9340	13/6	1.079	2.6892	79/0	1.079	5.7482
mc	0/0	1.068	0.0959	0/0	1.068	0.9172	24/24	1.079	5.6541	464/80	1.079	12.263	7584/1152	1.079	32.469
beecount	4/4	1.068	0.1340	52/4	1.068	0.3804	351/0	1.079	2.6020	2464/0	1.079	15.162	24303/0	1.079	26.794
dk17	0/0	1.068	0.0163	0/0	1.068	0.8407	4/4	1.068	1.3112	43/15	1.079	4.5285	228/32	1.079	7.4427
dk512	0/0	1.068	0.0655	0/0	1.068	0.2543	0/0	1.068	1.0782	2/2	1.068	3.1656	8/0	1.068	5.2290
donfile	0/0	1.068	0.0680	0/0	1.084	0.2435	6/6	1.111	2.1524	88/40	1.111	6.9961	609/0	1.128	13.913
ex2	0/0	1.068	0.0276	0/0	1.084	0.2511	0/0	1.111	0.8258	2/2	1.128	2.1067	33/18	1.160	5.0838
ex3	0/0	1.068	0.1057	0/0	1.068	0.8143	0/0	1.068	1.5383	13/13	1.079	3.2599	72/17	1.095	5.8687
ex5	0/0	1.068	0.0849	1/1	1.068	0.1966	19/11	1.068	2.7749	126/12	1.079	3.4505	502/0	1.079	6.0046
ex7	0/0	1.068	0.1386	0/0	1.068	0.3500	1/1	1.068	1.9106	20/12	1.079	3.4314	154/26	1.095	7.9619
s27	5/5	1.068	0.8147	169 /34	1.079	2.6787	3471/60	1.079	7.7577	61173 /92	1.079	18.849	1018443 /136	1.089	38.934
s208.1	0/0	1.095	1.5472	0/0	1.122	1.9293	0/0	1.149	4.1450	0/0	1.163	7.8173	0/0	1.224	21.182
s382	4/4	1.089	1.9593	48/0	1.106	2.6160	448/0	1.155	5.3998	3840/0	1.303	9.2638	31744/0	1.424	20.622
s400	4/4	1.089	1.8909	48/0	1.106	2.4733	448/0	1.155	5.1233	3840/0	1.303	9.5132	31744/0	1.424	18.580
s444	4/4	1.089	1.6991	48/0	1.139	2.7513	448/0	1.204	6.0497	3840/0	1.405	14.945	31744/0	1.608	26.369
s526	4/4	1.089	1.5060	48/0	1.139	3.2551	448/0	1.239	5.1112	3840/0	1.528	11.184	31744/0	1.685	21.344

In a sequential circuit, restoring inputs allow for the masking of feedback signals and thus eliminating the cumulative effect of errors. A partial error masking occurs when part of the feedback signals are logically masked by a specific combination of the primary input and the other feedback signals.

In spite of its large computational complexity and limited applicability, the STM-based analysis reveals the fundamental mechanism of error masking. This framework is enhanced by using BDDs to extend the proposed analysis to large circuits. Computational efficiency can further be improved by using appropriate ordering of variables in the construction of BDDs, as well as an optimized process for extracting the restoring inputs.

Simulation results have shown the effectiveness of the proposed approach. They also point out a few attractive features that albeit beyond the scope of this chapter, can be exploited to improve the reliable operation of sequential circuits. In an implementation of FSMs, for example, the don't-care values at the inputs can be configured into restoring inputs in logic synthesis such that errors in the state variables can be corrected during normal operation. Although an external reset can be utilized to clear the state variables, the use of restoring inputs has the following advantages:

1) Error masking due to restoring inputs occurs as an inherent part of the operation of an FSM without incurring an interruption. Therefore, the restored state is readily available for the next-step operation of the FSM. The time overhead incurred in this process is therefore significantly reduced.

2) The use of restoring inputs eliminates the need for an external reset signal, so it simplifies the related logic design and reduces the required numbers of pins and pads in chip packaging; this subsequently has an impact on the performance, area and cost of a chip [36].

3) Multiple-step restoring and partial error masking allow for more flexibility as well as an extended functionality in the operation of an FSM, compared to the basic function of a reset.

Hence, the proposed error masking is a potentially useful property of FSMs that can be exploited for an efficient and robust implementation of sequential circuits.

CHAPTER 3

New Metrics for the Reliability of Approximate and Probabilistic Adders*

Addition is a fundamental function in arithmetic operation; several adder designs have been proposed for implementations in inexact computing [37, 40]. These adders show different operational profiles; some of them are approximate in nature while others rely on probabilistic features of nanoscale circuits. However, there has been a lack of appropriate metrics to evaluate the efficacy of different designs. In this chapter, new metrics are proposed for evaluating the reliability as well as the power efficiency of approximate and probabilistic adders. Reliability is analyzed using the so-called sequential probability transition matrices (SPTMs). The error distance (ED) is first defined as the arithmetic distance between an erroneous output and the correct output for a given input. The mean error distance (MED) and the normalized error distance (NED) are then proposed as unified figures that consider the averaging effect of multiple inputs and the normalization of multiple-bit adders. It is shown that the NED is a nearly invariant metric that is almost independent of the size of an adder implementation; the NED, is therefore, useful in characterizing the reliability of a specific design. Since inexact adders are often used for saving power, the product of power and NED is further utilized

**A version of this chapter has been accepted for publication in [18] and [19].*

for evaluating the tradeoffs between power consumption and precision. Although illustrated using adders, the proposed metrics are potentially useful in assessing other arithmetic circuit designs for applications of inexact computing.

The rest of the chapter is organized as follows. Section 3.1 contains a review and Section 3.2 presents sequential probability transition matrices (S-PTMs). Section 3.3 presents the notions of error distance (ED) and the evaluations of mean error distance (MED) and normalized error distance (NED). Section 3.4 discusses the power and precision tradeoff. Section 6 concludes the chapter.

3.1. Review

In most digital systems, sequential circuits are utilized, so this chapter primarily deals with sequential adders. A sequential adder, such as the one shown in Figure 3.1, consists of a k-bit full adder concatenated with a k-bit register. In this section, different implementations of a full adder are reviewed. Particular emphasis is devoted to features relevant to soft and inexact computing.

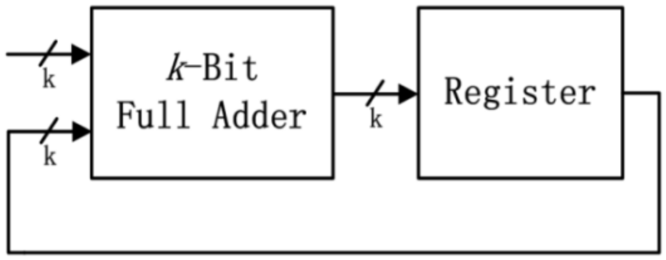


Figure 3.1. A k-bit sequential adder.

3.1.1 Conventional Full Adder (CFA)

Figure 3.2 shows a one-bit (precise) conventional full adder (CFA); the CFA is commonly connected in a ripple-carry implementation, i.e., by cascading instances of the circuit of Figure 3.2 in a linear array.

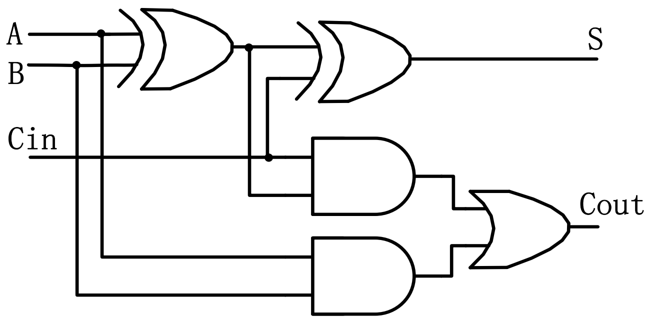


Figure 3.2. A one-bit conventional full adder (CFA).

3.1.2 Lower-Part-OR Adder (LOA)

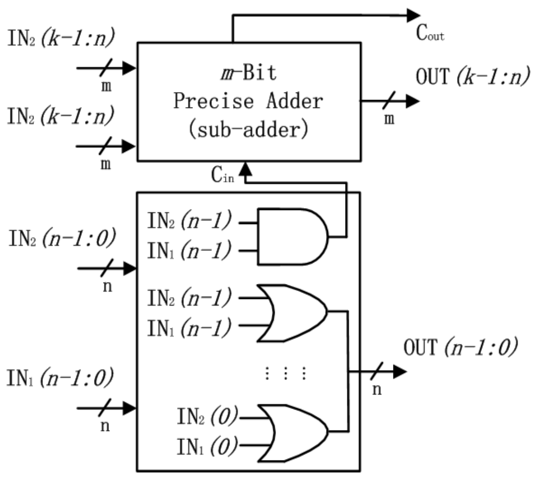


Figure 3.3. Hardware structure of the lower-part-OR adder (LOA) [37].

Differently from conventional designs that strictly operate according to the exact function (as defined by its truth table), an approximate logic implementation alters some entries in the truth table. This feature allows balancing precision with other performance metrics. The recently proposed LOA is based on such a design [37]. A LOA divides a *k*-bit addition into two smaller parts, i.e., two modules of *m*-bits and *n*–bits. As shown in Figure 3.3, the *m*-bit module of a LOA uses a smaller but precise adder (referred to as the *sub-adder*) to compute the exact values of the *m* most significant bits of the result (also referred to as the *upper part*). Additional OR gates are used to approximately compute the *n* least significant bits (also referred to as the *lower part*) of the sum by applying a bitwise OR operation on the respective input bits. An additional AND gate is used to generate the carry-in for the precise sub-adder when the most significant bits of both the lower-part inputs are “1.” As this implementation ignores the “trivial” carries in the lower part of the LOA adder, it may result in a loss of precision. Albeit using different structures, the approximate adder designs in [38] and [39] belong to the same category.

3.1.3 Approximate Mirror Adder (AMA)

A Mirror Adder (MA) is not based on the complementary structure of CMOS logic. It is based on a special arrangement of the transistors and is yet another common design for implementing conventional adders [40]. When approximate logic is applied to the MA cells, approaches such as IMPACT [40] have been reported to tradeoff precision for power and area. Three implementations of an approximate mirror adder (AMA) are proposed in [40] by removing different numbers of transistors. The truth table of these three approximate implementations is shown in Table 3.1. Similarly, AMAs can be used in the least significant *n*-bit (or the lower part) of an approximate sequential adder.

Table 3.1. Truth table of conventional mirror adder and its approximate implementations [40]; enclosed entries indicate incorrect outputs.

Inputs			Accurate		AMA1		AMA2		AMA3	
			Cout	Sum	Cout	Sum	Cout	Sum	Cout	Sum
0	0	0	0	0	0	①	0	0	0	0
0	0	1	0	1	0	1	0	1	0	⓪
0	1	0	0	1	①	⓪	0	⓪	0	1
0	1	1	1	0	1	0	⓪	①	⓪	①
1	0	0	0	1	0	1	①	⓪	0	⓪
1	0	1	1	0	1	0	1	0	1	0
1	1	0	1	0	1	0	1	0	1	①
1	1	1	1	1	1	⓪	1	1	1	1

3.1.4 Probabilistic Full Adder (PFA)

Probabilistic CMOS (PCMOS) is a technique for achieving savings in power consumption, while balancing performance at nanometric ranges. In a *k*-bit PFA, the most significant *m*-bit adders are implemented using perfectly reliable (and thus deterministic) gates, while the least significant *n* bits are implemented in PCMOS. Although new adder architectures have been proposed to optimize the use of PCMOS technology [41], probabilistic implementations of a conventional full adder are considered in this work. The structure of this type of adder is shown in Figure 3.4.

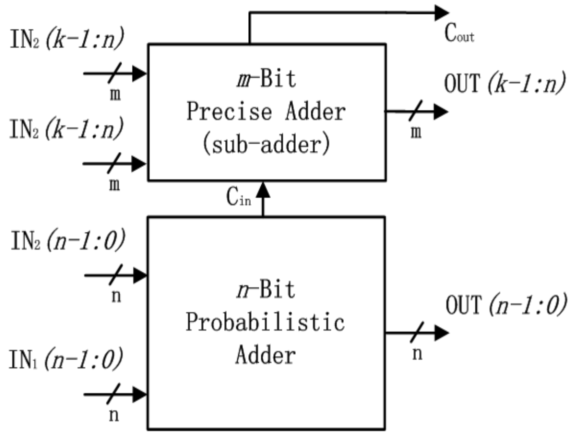


Figure 3.4. Hardware structure of the probabilistic full adder (PFA).

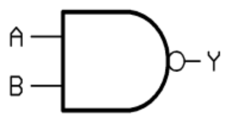
3.2. Sequential Probability Transition Matrices

3.2.1 Definitions

Prior to presenting the analysis, several definitions are first introduced. The signal probability is defined as the probability of a signal being a logical "1." In a combinational circuit, the output probability for a specific input vector *i* is defined as the joint probability that all of the outputs are "1" when the input vector is *i*. The output reliability for input *i* is defined as the joint probability that all outputs are correct for *i*. Any output that deviates from the correct value, is considered a faulty output. This definition of reliability considers the correlation among signals and is therefore used in this chapter unless noted otherwise. Circuit reliability is defined as the average output reliability over all applicable input vectors. (In this chapter, the adders introduced previously are analytically assessed for circuit reliability using the so-called sequential probability transition matrices (S-PTMs), as presented next.)

3.2.2 Probabilistic transfer matrices (PTMs)

The PTM approach represents a computational framework for the evaluation of circuit reliability in the presence of both deterministic and probabilistic errors [2]. A PTM is a matrix, in which the $(i, j)^{\text{th}}$ entry represents the conditional probability of the output vector of value j, determined by the circuit structure for the input vector i. Since a fault-free circuit has correct outputs with probability 1, it has an ideal transfer matrix (ITM), in which an entry is either 0 or 1. A two-input NAND gate's ITM and PTM are shown in Figure 3.5.



(a)

$$AB\left\{\begin{array}{c} \\ 00 \\ 01 \\ 10 \\ 11 \end{array}\right. \begin{array}{c} \overbrace{\begin{array}{cc} 0 & 1 \end{array}}^{Y} \\ \begin{bmatrix} 0 & 1 \\ 0 & 1 \\ 0 & 1 \\ 1 & 0 \end{bmatrix} \end{array}$$

(b)

$$AB\left\{\begin{array}{c} \\ 00 \\ 01 \\ 10 \\ 11 \end{array}\right. \begin{array}{c} \overbrace{\begin{array}{cc} 0 & \quad 1 \end{array}}^{Y} \\ \begin{bmatrix} 1-p & p \\ 1-p & p \\ 1-p & p \\ p & 1-p \end{bmatrix} \end{array}$$

(c)

Figure 3.5. An ITM and PTM for a two-input NAND gate: (a) a two-input NAND gate, (b) ITM of the NAND gate, (c) PTM of the NAND gate (the gate has a probability of 1-p to produce an error).

A circuit PTM can be obtained by combining the gate PTMs using simple rules of matrix operations and the connectivity of the gates. Matrix multiplications are used for gates connected in series, while tensor products are used for gates connected in parallel. Special PTMs and ITMs are constructed for fanouts, interconnects and swapping of wires by taking into account their topologies. Since signal representation and propagation are incorporated into the formulation of PTMs, a circuit PTM contains accurate and comprehensive information about the probabilistic behavior of a circuit. A PTM can also be extended to account for more complex circuit structures. In [27], PTMs are used to estimate the soft error effect in sequential circuits. In this chapter, the notions of sequential PTMs (S-PTMs) are defined and a detailed formulation is presented to consider the topology and structure of sequential circuits for reliability evaluation.

3.2.3 Formulation of sequential PTMs (SPTMs)

Consider a general Mealy model of a sequential circuit (Figure 3.6). In this circuit, there are $M+N$ inputs: M of them are Primary Inputs and the remaining N inputs are Present States (i.e., the feedback signals from the flip-flops). There are also $K+N$ outputs: K of them are Primary Outputs, while the remaining N outputs are Next States, which will be stored in the flip-flops and then fed back into the inputs during the next clock cycle.

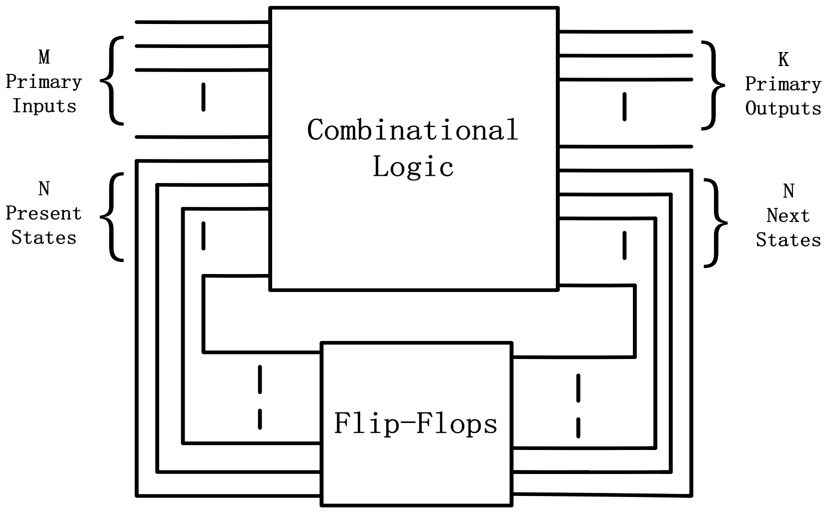


Figure 3.6. Mealy model of a sequential circuit.

For this circuit, S-PTMs define several matrices mapping from the $M+N$ inputs to the N next states (denoted by $\boldsymbol{\Phi_{ns}}$) and to the K primary outputs (denoted by $\boldsymbol{\Phi_{po}}$), also from the N next states to the N present states. In the S-PTM $\boldsymbol{\Phi_{ns}}$, the entries denote the transition probabilities from the $2^{\mathrm{M+N}}$ input combinations to the 2^{N} next states. $\boldsymbol{\Phi_{ns}}$ is therefore a $2^{\mathrm{M+N}} \times 2^{\mathrm{N}}$ matrix, given by

$$\boldsymbol{\Phi_{ns}} = \begin{bmatrix} p(0|0) & p(1|0) & \dots & p(j|0) & \dots & p(2^N-1|0) \\ p(0|1) & p(1|1) & \dots & p(j|1) & \dots & p(2^N-1|1) \\ \dots & \dots & \dots & \dots & \dots & \dots \\ p(0|i) & p(1|i) & \dots & p(j|i) & \dots & p(2^N-1|i) \\ \dots & \dots & \dots & \dots & \dots & \dots \\ p(0|2^{M+N}-1) & p(1|2^{M+N}-1) & \dots & p(j|2^{M+N}-1) & \dots & p(2^N-1|2^{M+N}-1) \end{bmatrix} \quad (3.1)$$

where $p(j|i)$ represents the conditional transition probability that the next state has the j^{th} value, given that the inputs have the i^{th} value. Similarly, the entries in $\boldsymbol{\Phi_{po}}$ are given by the transition probabilities from the 2^{M+N} input combinations to the 2^K primary output combinations.

Furthermore, the circuit S-PTM $\boldsymbol{\Phi_c}$ is defined for the mapping from the *M+N* inputs to the *K+N* outputs. In the likely circuit scenario, the primary outputs and the next states are correlated due to the fanouts of signals to both the primary inputs and the next states (an example is evident in the half adder circuit of Figure 3.7). So, $\boldsymbol{\Phi_c}$, like any other SPTM, can be obtained in a similar way as the combinational PTMs, i.e. by combining the gate PTMs following the matrix operation rules and the connectivity of the gates in a circuit.

If the flip-flops are also subject to errors, the SPTM $\boldsymbol{\Phi_{ff}}$ must be defined. For a single flip-flop with an error rate of $\boldsymbol{\varepsilon}$, its SPTM $\boldsymbol{\Phi_{1-ff}}$ is given by

$$\boldsymbol{\Phi_{1-ff}} = \begin{bmatrix} 1-\varepsilon & \varepsilon \\ \varepsilon & 1-\varepsilon \end{bmatrix}. \quad (3.2)$$

For *N* independent flip-flops, the S-PTM $\boldsymbol{\Phi_{ff}}$ is obtained as

$$\boldsymbol{\Phi_{ff}} = \boldsymbol{\Phi_{1-ff}}^{\otimes N}, \quad (3.3)$$

i.e., $\boldsymbol{\Phi_{ff}}$ is the *n*th tensor product of $\boldsymbol{\Phi_{1-ff}}$. Its entries are the transition probabilities from the 2^N input combinations and the 2^N output combinations of the flip-flops.

At time *t*, the primary-input vector is given by

$$\mathbf{P_{pi}}(\mathrm{t}) = (\mathrm{P_{pi}}(0,\mathrm{t}), \mathrm{P_{pi}}(1,\mathrm{t}), \dots \dots, \mathrm{P_{pi}}\left((2^{\mathrm{M}} - 1), \mathrm{t}\right), \tag{3.4}$$

where $\mathrm{P_{pi}}(\mathrm{i},\mathrm{t})$ is the probability that the primary inputs have the i^{th} value at time *t*. The present-state vector is given by

$$\mathbf{P_{ps}}(\mathrm{t}) = (\mathrm{P_{ps}}(0,\mathrm{t}), \mathrm{P_{ps}}(1,\mathrm{t}), \dots \dots, \mathrm{P_{ps}}\left((2^{\mathrm{M}} - 1), \mathrm{t}\right), \tag{3.5}$$

where $\mathrm{P_{ps}}(\mathrm{i},\mathrm{t})$ is the probability that the present states have the i^{th} value at time *t*.

Therefore inputs can be represented by a vector of length $2^{\mathrm{M+N}}$ given by

$$\mathbf{P_{in}}(\mathrm{t}) = \mathbf{P_{pi}}(\mathrm{t}) \otimes \mathbf{P_{ps}}(\mathrm{t}), \tag{3.6}$$

where $\otimes$ indicates the tensor product.

For the N next states, their joint distribution is described by a vector $\mathbf{P_{ns}}(\mathrm{t})$ of length 2^{N}, and

$$\mathbf{P_{ns}}(\mathrm{t}) = \mathbf{P_{in}}(\mathrm{t}) * \boldsymbol{\Phi_{ns}}. \tag{3.7}$$

The present-state vector at time *t+1*, $\mathbf{P_{ps}}(\mathrm{t}+1)$**,** can be calculated as follows:

$$\mathbf{P_{ps}}(\mathrm{t}+1) = \mathbf{P_{ns}}(\mathrm{t}) * \boldsymbol{\Phi_{ff}}. \tag{3.8}$$

Similarly, the primary-output vector $\mathbf{P_{po}}(\mathrm{t})$ is given by

$$\mathbf{P_{po}}(\mathrm{t}) = \mathbf{P_{in}}(\mathrm{t}) * \boldsymbol{\Phi_{po}}. \tag{3.9}$$

3.2.4 Reliability evaluation using SPTMs

In a sequential circuit, the cumulative effect of errors is modeled by matrix operations (as according to the Markov process); specifically, error propagation is described by matrix multiplications, while the combination of signals is described by tensor products. By considering the S-PTMs defined previously for the sequential circuit of Figure 3.6, the input vector at time *t+1*, $\mathbf{P_{in}}(\mathrm{t}+1)$, can be computed by

$$\mathbf{P_{in}}(\mathrm{t}+1) = \mathbf{P_{pi}}(\mathrm{t}+1) \otimes \mathbf{P_{ps}}(\mathrm{t}+1) = \mathbf{P_{pi}}(\mathrm{t}+1) \otimes \left(\mathbf{P_{in}}(\mathrm{t}) * \boldsymbol{\Phi_{ns}} * \boldsymbol{\Phi_{ff}}\right). \quad (3.10)$$

It can be shown that

$$\mathbf{P_{in}}(\mathrm{t}+1) = \mathbf{P_{in}}(\mathrm{t}) * \left(\mathbf{P_{pi}}(\mathrm{t}+1) \otimes (\boldsymbol{\Phi}_{\mathbf{ns}} * \boldsymbol{\Phi_{ff}})\right), \quad (3.11)$$

as per the Theorem in the Appendix.

Let

$$\boldsymbol{\Phi_{in}}(t) = \mathbf{P_{pi}}(\mathrm{t}+1) \otimes (\boldsymbol{\Phi}_{\mathbf{ns}} * \boldsymbol{\Phi_{ff}}), \quad (3.12)$$

(3.11) can be written as

$$\mathbf{P_{in}}(\mathrm{t}+1) = \mathbf{P_{in}}(\mathrm{t}) * \boldsymbol{\Phi_{in}}(t). \quad (3.13)$$

$\boldsymbol{\Phi_{in}}(t)$ is a time-dependent SPTM, whose entries are the transition probabilities from the $2^{\mathrm{M+N}}$ input combinations at time *t* to the $2^{\mathrm{M+N}}$ input combinations at time *t+1*. Therefore, (3.13) describes the (error) characteristics of the sequential circuit (under its current primary inputs) and captures the temporal correlation between the inputs at different time steps.

At time *t+1*, the primary output vector can now be computed as

$$\mathbf{P_{po}}(\mathrm{t}+1) = \mathbf{P_{in}}(\mathrm{t}+1) * \boldsymbol{\Phi_{po}} = \mathbf{P_{in}}(\mathrm{t}) * \boldsymbol{\Phi}_{\mathbf{in}}(\mathrm{t}) * \boldsymbol{\Phi_{po}}$$

$$= \mathbf{P_{in}}(\mathrm{t}-1) * \boldsymbol{\Phi_{in}}(t-1) * \boldsymbol{\Phi_{in}}(t) * \boldsymbol{\Phi_{po}}$$

$$= \cdots$$

$$= \mathbf{P_{in}}(\mathrm{t-k}) * \boldsymbol{\Phi_{in}}(t-k) * \boldsymbol{\Phi_{in}}(t-k+1) * \ldots * \boldsymbol{\Phi_{in}}(t-1) * \boldsymbol{\Phi_{po}}$$

$$= \cdots$$

$$= \mathbf{P_{in}}(0) * \boldsymbol{\Phi_{in}}(0) * \boldsymbol{\Phi_{in}}(1) * \ldots * \boldsymbol{\Phi_{in}}(t-1) * \boldsymbol{\Phi_{po}}$$

$$= \mathbf{P_{in}}(0) * (\textstyle\prod_{k=0}^{t} \boldsymbol{\Phi_{in}}(k)) * \boldsymbol{\Phi_{po}}.$$

(3.14)

Furthermore, assume that in the error-free case, the ideal output vector is given by $\mathbf{I_{po}}(\mathrm{t}+1)$; so if the primary inputs at every time step and the initial state are known, the output reliability of a sequential circuit at time *t+1* can be computed by:

$$\mathbf{R_{po}}(\mathrm{t}+1) = \mathbf{P_{po}}(\mathrm{t}+1) \cdot \mathbf{I_{po}}(\mathrm{t}+1), \tag{3.15}$$

i.e. the dot product of the two vectors.

Example: Consider the sequential half adder circuit shown in Figure 3.7. Given the gate PTMs, the circuit PTM can be found by first dividing the combinational part of the circuit into several levels and then evaluating the PTMs at each level. Alternatively, $\boldsymbol{\Phi_{ns}}$ and $\boldsymbol{\Phi_{po}}$ can be obtained for the next state and output logic.

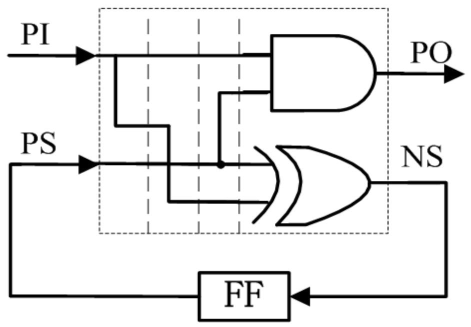


Figure 3.7. S-PTM evaluation of a sequential half adder with one primary input, one flip-flop and one primary output.

Assume a gate error rate of 0.01, we obtain

$$\boldsymbol{\Phi}_{po} = \begin{bmatrix} 0.99 & 0.01 \\ 0.99 & 0.01 \\ 0.99 & 0.01 \\ 0.01 & 0.99 \end{bmatrix}, \tag{3.16}$$

and

$$\boldsymbol{\Phi}_{ns} = \begin{bmatrix} 0.99 & 0.01 \\ 0.01 & 0.99 \\ 0.01 & 0.99 \\ 0.99 & 0.01 \end{bmatrix}. \tag{3.17}$$

Also assume that the flip-flop has an error rate of 0.02, i.e., $\boldsymbol{\varepsilon} = 0.02$, then

$$\boldsymbol{\Phi}_{ff} = \boldsymbol{\Phi}_{1-ff} = \begin{bmatrix} 0.98 & 0.02 \\ 0.02 & 0.98 \end{bmatrix}. \tag{3.18}$$

Given the primary input at each time step, the reliability of the primary output can be calculated as follows. Let the initial state of the flip-flop be $\mathbf{P_{ps}}(0) = (0.99, 0.01)$. If the primary-input vector is given by $\mathbf{P_{pi}}(0) = (0.05, 0.95)$, then at *t=0*, the input vector can be calculated as

$$\mathbf{P_{in}}(0) = \mathbf{P_{pi}}(0) \otimes \mathbf{P_{ps}}(0) = (0.0495, 0.0005, 0.9405, 0.0095). \tag{3.19}$$

At *t=1*, the present-state vector is given by

$$\mathbf{P_{ps}}(1) = \mathbf{P_{ns}}(0) * \boldsymbol{\Phi}_{ff} = \mathbf{P_{in}}(0) * \boldsymbol{\Phi}_{ns} * \boldsymbol{\Phi}_{ff} = (0.0851, 0.9149). \tag{3.20}$$

If the primary input vector at *t=1* is given by $\mathbf{P_{pi}}(1) = (0.98, 0.02)$, then the input vector $\mathbf{P_{in}}(1)$ is given by

$$\mathbf{P_{in}}(1) = \mathbf{P_{pi}}(1) \otimes \mathbf{P_{ps}}(1) = (0.0834, 0.8966, 0.0017, 0.0183). \tag{3.21}$$

For the sequential adder, all outputs (both primary outputs and the next states) must be included for the final reliability evaluation. The output vector is obtained as

$$\mathbf{P_{po}}(1) = \mathbf{P_{in}}(1) * \boldsymbol{\Phi_{po}} = (0.9721, 0.0279). \qquad (3.22)$$

(3.22) gives the output distribution after one clock cycle for the probabilistic sequential adder. The above process can also be computed directly from (3.14) for *t=0*.

Since the ideal error-free output vector is $\mathbf{I_{po}}(1) = (1, 0)$, the reliability can be obtained from (3.12) as:

$$\mathbf{R_{po}}(1) = \mathbf{P_{po}}(1) \cdot \mathbf{I_{po}}(1) = 0.9721, \qquad (3.23)$$

i.e. the output reliability at this time is 0.9721.

Table 3.2. The reliability of some sequential circuits obtained using SPTMs. The input vector is randomly generated at each clock cycle.

Circuits	**Characteristics**				**Reliability (ε=0.005)**				
	Gates	**Inputs**	**Outputs**	**DFFs**	**T=1**	**T=2**	**T=3**	**T=4**	**T=5**
2-bit counter	2	0	2	2	0.990	0.980	0.971	0.961	0.952
Simplified semaphore	6	0	2	2	0.983	0.965	0.948	0.932	0.916
s27	10	4	1	3	0.975	0.990	0.985	0.960	0.948
2-bit probabilistic sequential adder	10	4	1	2	0.976	0.962	0.948	0.935	0.922
2-bit Lower-part OR sequential adder	3	4	1	2	1	0	0	0	0
2-bit AMA1 based sequential adder	22 (transistors)	4	1	2	0	1	1	0	0
2-bit AMA2 based sequential adder	22 (transistors)	4	1	2	1	0	0	0	0
2-bit AMA3 based sequential adder	N/A	4	1	2	0	0	0	0	0

Table 3.2 shows the simulation results of some sequential circuits that were analyzed using the SPTM approach. Albeit with an exponential complexity, the SPTM approach can efficiently be used by decomposing a large circuit into small modules [27]. Alternatively, the reliability can be evaluated using probabilistic gate models [3] or stochastic computational models [9]. As can be seen in Table 3.2, the reliability values are very different at different clock cycles (different 'T') for the probabilistic and approximate designs. For example, for the LOA and AMAs, the reliability is mostly 0, which indicates that these designs are totally unreliable. However, they can be useful in many applications, where a partial loss of precision can be tolerated. Therefore, these new paradigms need to be better assessed as efficient design alternatives. Toward this end, new metrics are proposed subsequently to characterize the extent of loss of precision and its gain in reducing power consumption.

3.3. Mean and Normalized Error Distances

3.3.1 Error Distance

In this section, a *new metric* is proposed for evaluating the reliability of an adder and SPTMs are used in the computation of this metric. Consider as an example the case in which the exact output Sum of an adder is "100101" and other values can result as inexact outputs. For example, both "100100" and "110101" represent inexact values. However, these two output values have different implications when compared to the correct value: "100100" means the output is different by 1 (or at a distance of 1) from the correct value, while "110101" is different by 16 (or at a distance of 16) from the correct value. So, an output can take erroneous values that are substantially different from the correct one. This is determined by the error effects on the addition; for example, a lower bit error has less impact on the output of an adder.

Under these circumstances, the metric of circuit reliability has limited usefulness in assessing an adder because it considers only the presence of an error, but not the error's implication on the performed addition. A new metric referred to as *error distance* (ED) is therefore proposed to better characterize the reliability of an adder.

In general, the ED between two binary numbers, a (erroneous) and b (correct, i.e., golden), is defined as the arithmetic distance between these two numbers, i.e.,

$$ED(\mathrm{a}, \mathrm{b}) = |\mathrm{a} - \mathrm{b}| = \left|\sum_i \mathrm{a}[i] * 2^{\mathrm{i}} - \sum_j \mathrm{b}[\mathrm{j}] * 2^{\mathrm{j}}\right|, \tag{3.24}$$

where i and j are the indices for the bits in a and b, respectively. In the previous example, the two erroneous values "100100" and "110101" have an ED of 1 and 16 to the golden "100101," respectively.

For a non-deterministic implementation, the output is probabilistic and usually follows a distribution for a given input a_i. In this case, the ED of the output (denoted by d_i) is defined as the weighted average of EDs of all possible outputs to the nominal output. Assume that for a given input, the output has a nominal value b, but it can take any value given in a set of vectors b_j ($1 \leq j \leq r$); the ED of the output is then given by:

$$d_i = \sum_j \mathrm{ED}(\mathrm{b_j}, \mathrm{b}) * \mathrm{p_j}, \tag{3.25}$$

where p_j is the output probability of b_j.

3.3.2 Mean Error Distance (MED)

When the primary inputs to a circuit are non-deterministic and thus each input occurs at certain probability, the *mean error distance* (MED) of a circuit (denoted by d_m) is defined as the mean value of the EDs of all possible outputs for each input. Assume that the input is given by a set of vectors a_i ($1 \leq i \leq s$) and that each vector occurs with a probability given by a corresponding value q_i ($1 \leq i \leq s$). Then, the MED of the circuit is given by:

$$d_m = \sum_i \mathrm{d_i} * \mathrm{q_i}, \tag{3.26}$$

where d_i is the ED of the outputs for input a_i, which can be computed by (3.25).

For simplicity, uniformly-distributed random inputs are considered hereafter, i.e., each input occurs with the same probability. Consider as an example a 3-bit adder. In Figure 3.3 and Figure 3.4, let k = 3, m = 1 and n = 2. A 3-bit CFA is used to calculate the correct output value. A gate error rate of 0.028 is used for PFA; this value is selected such that the MED is close to that of the LOA (as shown in subsequent sections). Table 3.3 shows the results of three experiments, each of which consists of four consecutive clock cycles. As expected, the CFA has a MED of 0 throughout the four clock cycles in all experiments. However, the MED for the PFA is significantly greater than that of its LOA counterpart in most cases. Also shown is that the MEDs of the LOA and AMAs can be reduced to 0 at an intermediate time step. For example, the MED of the LOA has a value of 0 at both Clk1 and Clk4 in Experiment 3; this is due to the effect of error masking, as discussed next.

Table 3.3. Mean Error Distance for four clock cycles with random inputs.

Experiment	Architecture	Clk1	Clk2	Clk3	Clk4
No. 1	CFA	0	0	0	0
	PFA	0.7400	1.1920	1.4620	2.1900
	LOA	0	2	1	1
	AMA1	1	1	2	3
	AMA2	1	0	1	2
	AMA3	0	0	1	2
No. 2	CFA	0	0	0	0
	PFA	0.7220	1.0260	1.5940	1.9640
	LOA	0	1	1	1
	AMA1	0	1	1	2
	AMA2	0	1	1	1
	AMA3	1	2	2	3
No. 3	CFA	0	0	0	0
	PFA	0.7820	1.2100	1.5180	2.2340
	LOA	0	2	1	0
	AMA1	0	2	3	3
	AMA2	1	0	2	3
	AMA3	2	1	2	3

$$\Phi_{CFA} = \begin{array}{c|cccccccc}
 & 000 & 001 & 010 & 011 & 100 & 101 & 110 & 111 \\
\hline
0000 & 1 & & & & & & & \\
0001 & & 1 & & & & & & \\
0010 & & & 1 & & & & & \\
0011 & & & & 1 & & & & \\
0100 & & 1 & & & & & & \\
0101 & & & 1 & & & & & \\
0110 & & & & 1 & & & & \\
0111 & & & & & 1 & & & \\
1000 & & & 1 & & & & & \\
1001 & & & & 1 & & & & \\
1010 & & & & & 1 & & & \\
1011 & & & & & & 1 & & \\
1100 & & & & 1 & & & & \\
1101 & & & & & 1 & & & \\
1110 & & & & & & 1 & & \\
1111 & & & & & & & 1 &
\end{array}$$

(a)

$$\Phi_{LOA} = \begin{array}{c|cccccccc}
 & 000 & 001 & 010 & 011 & 100 & 101 & 110 & 111 \\
\hline
0000 & 1 & & & & & & & \\
0001 & & 1 & & & & & & \\
0010 & & & 1 & & & & & \\
0011 & & & & 1 & & & & \\
0100 & & 1 & & & & & & \\
0101 & & 1 & & & & & & \\
0110 & & & & 1 & & & & \\
0111 & & & & 1 & & & & \\
1000 & & & 1 & & & & & \\
1001 & & & & 1 & & & & \\
1010 & & & & & & & 1 & \\
1011 & & & & & & & & 1 \\
1100 & & & & 1 & & & & \\
1101 & & & & 1 & & & & \\
1110 & & & & & & & & 1 \\
1111 & & & & & & & & 1
\end{array}$$

(b)

$$\Phi_{\mathrm{AMA1}} = \begin{array}{c|cccccccc}
 & 000 & 001 & 010 & 011 & 100 & 101 & 110 & 111 \\
\hline
0000 & & & & 1 & & & & \\
0001 & & & 1 & & & & & \\
0010 & & & & & & 1 & & \\
0011 & & & & & 1 & & & \\
0100 & & & & 1 & & & & \\
0101 & & & 1 & & & & & \\
0110 & & & & & & 1 & & \\
0111 & & & & & 1 & & & \\
1000 & & & & 1 & & & & \\
1001 & & & & & 1 & & & \\
1010 & & & & & & 1 & & \\
1011 & & & & & 1 & & & \\
1100 & & & & 1 & & & & \\
1101 & & & & & 1 & & & \\
1110 & & & & & & 1 & & \\
1111 & & & & & 1 & & &
\end{array}$$

(c)

$$\Phi_{\mathrm{AMA2}} = \begin{array}{c|cccccccc}
 & 000 & 001 & 010 & 011 & 100 & 101 & 110 & 111 \\
\hline
0000 & 1 & & & & & & & \\
0001 & 1 & & & & & & & \\
0010 & 1 & & & & & & & \\
0011 & 1 & & & & & & & \\
0100 & & & 1 & & & & & \\
0101 & & & 1 & & & & & \\
0110 & & & 1 & & & & & \\
0111 & & & 1 & & & & & \\
1000 & & & & & 1 & & & \\
1001 & & & & & 1 & & & \\
1010 & & & & & 1 & & & \\
1011 & & & & & 1 & & & \\
1100 & & & & & 1 & & & \\
1101 & & & & & 1 & & & \\
1110 & & & & & & & 1 & \\
1111 & & & & & & & 1 &
\end{array}$$

(d)

$$\Phi_{\mathrm{AMA3}} = \begin{array}{c|cccccccc}
 & 000 & 001 & 010 & 011 & 100 & 101 & 110 & 111 \\
\hline
0000 & 1 & & & & & & & \\
0001 & & 1 & & & & & & \\
0010 & & & 1 & & & & & \\
0011 & & & & 1 & & & & \\
0100 & 1 & & & & & & & \\
0101 & & 1 & & & & & & \\
0110 & & & 1 & & & & & \\
0111 & & & & 1 & & & & \\
1000 & & & & & 1 & & & \\
1001 & & & & & & 1 & & \\
1010 & & & & & & & 1 & \\
1011 & & & & & & & & 1 \\
1100 & & & & & 1 & & & \\
1101 & & & & & & 1 & & \\
1110 & & & & & & & 1 & \\
1111 & & & & & & & & 1
\end{array}$$

(e)

$$\Phi_{PFA} = \begin{array}{c|cccccccc}
 & 000 & 001 & 010 & 011 & 100 & 101 & 110 & 111 \\
\hline
0000 & 0.722 & 0.044 & 0.126 & 0.004 & 0.084 & 0.005 & 0.014 & 0.001 \\
0001 & 0.044 & 0.722 & 0.004 & 0.126 & 0.005 & 0.084 & 0.001 & 0.014 \\
0010 & 0.044 & 0.003 & 0.718 & 0.044 & 0.096 & 0.002 & 0.088 & 0.005 \\
0011 & 0.003 & 0.044 & 0.044 & 0.718 & 0.002 & 0.096 & 0.005 & 0.088 \\
0100 & 0.044 & 0.722 & 0.004 & 0.126 & 0.005 & 0.084 & 0.001 & 0.014 \\
0101 & 0.083 & 0.004 & 0.765 & 0.044 & 0.012 & 0.001 & 0.086 & 0.005 \\
0110 & 0.003 & 0.044 & 0.044 & 0.718 & 0.002 & 0.096 & 0.005 & 0.088 \\
0111 & 0.045 & 0.002 & 0.044 & 0.001 & 0.806 & 0.047 & 0.051 & 0.004 \\
1000 & 0.043 & 0.003 & 0.718 & 0.044 & 0.097 & 0.002 & 0.088 & 0.005 \\
1001 & 0.003 & 0.043 & 0.044 & 0.718 & 0.002 & 0.097 & 0.005 & 0.088 \\
1010 & 0.043 & 0.002 & 0.002 & 0 & 0.763 & 0.047 & 0.138 & 0.005 \\
1011 & 0.002 & 0.043 & 0 & 0.002 & 0.047 & 0.763 & 0.005 & 0.138 \\
1100 & 0.003 & 0.043 & 0.044 & 0.718 & 0.002 & 0.097 & 0.005 & 0.088 \\
1101 & 0.047 & 0.003 & 0.044 & 0.001 & 0.804 & 0.046 & 0.051 & 0.004 \\
1110 & 0.002 & 0.043 & 0 & 0.002 & 0.047 & 0.763 & 0.005 & 0.138 \\
1111 & 0.004 & 0.001 & 0.025 & 0.002 & 0.091 & 0.004 & 0.826 & 0.047
\end{array}$$

(f)

Figure 3.8. The S-PTM Φ_{ns} for the two lower bits of the 3-bit adder: (a) CFA (b) LOA (c) AMA1 (d) AMA2 (e) AMA3 and (f) PFA. Each row corresponds to an input consisting of two bits from the primary inputs and two bits from the feedbacks; each column corresponds to a 3-bit output of the 2-bit addition. An entry in a matrix indicates a transition (probability) from an input to an output.

For uniformly distributed inputs, the 16 input values occur with the same probability of 1/16 for this 3-bit adder. Based on the S-PTM model, the transition matrix $\boldsymbol{\Phi_{ns}}$ for the lower two bits is given in Figure 3.8 for each implementation (as the higher bits are accurate and are therefore the same). Given these transition matrices, the ED of an output can be computed for an input against the corresponding output of the CFA (taken as the golden value). Given $\boldsymbol{\Phi_{LOA}}$ in Figure 3.8(b), for example, the MED of the LOA is:

$$d_{m,\mathrm{LOA}} = \frac{1}{16} * (0+0+0+0+0+1+0+1+0+0+2+2+0+1+2+1) = 0.625. \tag{3.27}$$

The MED of the PFA is calculated using $\boldsymbol{\Phi_{PFA}}$ in Figure 3.8(f) as:

$$\mathrm{d}_{\mathrm{m,PFA}} = \frac{1}{16} * \sum_i \mathrm{d_i} = 0.6167. \tag{3.28}$$

Similarly, the MEDs of the AMAs are obtained as follows:

$$d_{m,\mathrm{AMA1}} = \frac{1}{16} * (3+1+3+1+2+0+2+0+1+1+1+1+0+0+0+2) = 1.125, \tag{3.29}$$

$$d_{m,\mathrm{AMA2}} = \frac{1}{16} * (0+1+2+3+1+0+1+2+2+1+0+1+1+0+1+0) = 1, \tag{3.30}$$

$$d_{m,\mathrm{AMA3}} = \frac{1}{16} * (0+0+0+0+1+1+1+1+2+2+2+2+1+1+1+1) = 1. \tag{3.31}$$

This shows that the AMA1 has the largest MED, while the MED of the PFA (0.6167) is only slightly different from that of the LOA for an error rate of 0.028. However, the cumulated error distances are quite different in the sequential adders, as shown in Table 3.2. This is mainly due to a result of partial error masking [17]. For the LOA and AMA2, several so-called restoring inputs are found, as indicated by the neighboring 1's in the columns of $\boldsymbol{\Phi}_{\boldsymbol{LOA}}$ and $\boldsymbol{\Phi}_{\boldsymbol{AMA2}}$. By these inputs, errors can be logically masked, which leads to the same next state from different present states. This can also be explained as follows. For certain primary inputs, the approximate logic masks the cumulative errors in the LOA and AMA2. However, this is not the case for the PFA (or CFA), because it is always affected by errors; therefore, errors accumulate rather than being masked.

3.3.3 Mean Error Distance (MED) Evaluation

In this section, the MEDs of the various designs are evaluated against a baseline design of a reduced precision adder, i.e., the lower-bit ignored adder (LIA). In this implementation, rather than using *n*-bit unreliable adders for computation, the lower significant bits are ignored.

Consider the general case of a 32-bit adder (*k*=32). Figure 3.9(a) shows the reliability of each adder design, while the MEDs are plotted in Figure 3.9(b) by varying the number of lower bits, *n*. A comparison of the adder implementations shows that the interpretation of these metrics (i.e., reliability and MED) leads to a different assessment result; for example under the MED metric, the performance of LOA is similar to a PFA with a gate error rate of 0.028. However, under the reliability metric the reliability of LOA is significantly lower than the reliability of PFA for the same gate error rate.

As shown in Figure 3.9(b), an *n*-bit ignored LIA performs slightly better in terms of the MED than an *n*-bit PFA at a gate error rate of 0.2 for the 32-bit adder. However, a PFA with an error rate significantly lower than 0.2 is expected to have a smaller MED. The AMAs have a similar performance as a PFA with an error rate of approximately 0.05. Moreover, as shown in Figure 3.9(b), the expected MEDs of both the LOA and the PFA increase exponentially with the number of lower bits. For the LOA, error masking occurs when the carry bit is ignored - there is no carry for the lower bits. Therefore, errors in a lower bit are operationally isolated. In this way, error masking occurs in the combinational circuit. Therefore, errors in the lower bits cannot be propagated to the higher bits, thus preventing their cumulative effect.

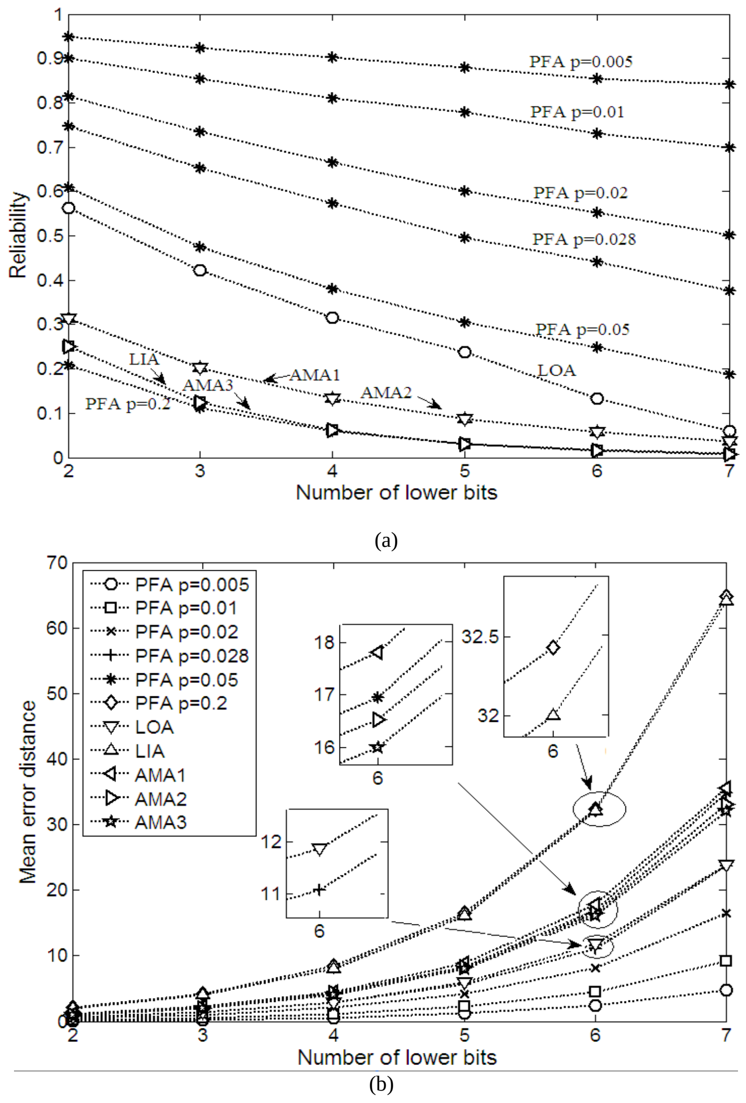


Figure 3.9. (a) Reliability vs. the number of lower bits in a 32-bit adder, and (b) MED vs. the number of lower bits in the 32-bit adder.

The simulation results of Figure 3.9 show the effectiveness of the proposed evaluation technique. It reveals that, although there is no cumulative error for the LOA, an error due to the approximate logic makes its MED not as optimistic as expected, especially for an increased number of lower bits. This is due to the fact that an error can occur in the most significant lower bit. The PFA shows a different scenario. Errors in the lower bits are likely to be propagated to the higher bits. So, its MED is rather large even for a small error rate, due to the cumulative errors from the lower bits (even though the error that results from the most significant bit is not that large).

3.3.4 Normalized Error Distance (NED) and its Evaluation

It is shown that in Figure 3.9(b), the MED increases exponentially with the number of lower bits in an adder. Therefore, MED is an unfair metric when a comparison is made between two adders with different lower bits, as the maximum value of error that can be effectively reached, has also changed. To overcome this limitation, a normalized MED, referred to as a *normalized error distance* (NED), is defined as follows:

$$d_n = \frac{d_m}{D}, \tag{3.32}$$

where d_m is the MED and *D* is the maximum value of error that an unreliable adder can have. This maximum value is usually 2^{n} for *n* lower bits, so we obtain

$$d_n = \frac{d_m}{2^{\mathrm{n}}}. \tag{3.33}$$

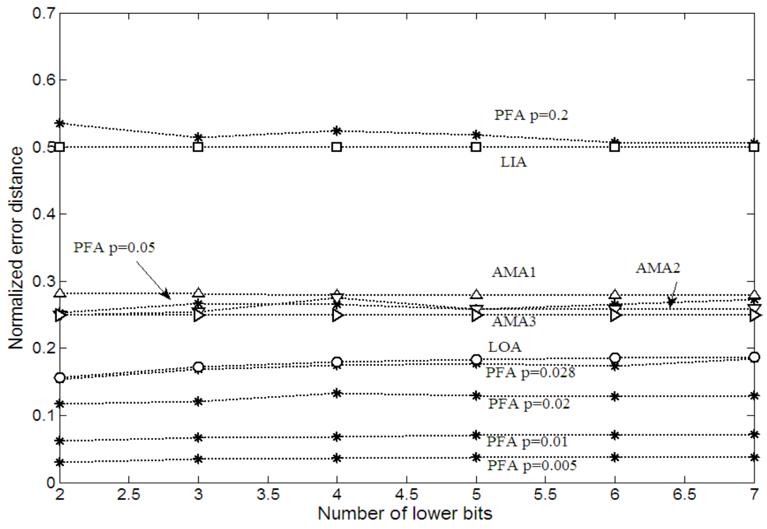


Figure 3.10. Normalized Error Distance (NED) vs. the number of lower bits.

Figure 3.10 shows the NEDs of the various adder implementations. The NEDs of the LIA are constant at 0.5 and thus serving as a baseline. For other implementations, as revealed in the figure, the NED shows little change and only fluctuates in a small interval when different lower bits are compared. This is consistent with the results in Figure 3.9(b), i.e., the MED increases exponentially with the number of lower bits.

Hence, NED is a stable metric that is almost independent of the size of an implementation and is useful in assessing the reliability of a specific design. This feature also brings a new perspective for the evaluation and comparison of different adder implementations for inexact computing.

3.4. Power and Precision Trade-off

As discussed previously, inexact computing is confronted with many similarities and often contradictory features that can also be found in bio-inspired systems, i.e., systems made of a large number of unreliable modules. These systems utilize extensive networks to circumvent the unreliable nature of the computational modules while still retaining low power/energy consumption. The adder configurations presented previously draw significant resemblance to this type of system. The LOA and AMAs resort to approximate logic to target reliable modules and PFA resorts to characterizing probabilistic behavior of nanoscale modules. However, one of the issues that must be addressed for inexact computing is that probabilistic implementations may likely tradeoff too much accuracy for little saving in area and power. In this section, this tradeoff is evaluated using the product of power and NED of an implementation.

Consider first the power consumption of the adders. As suggested in [42], the energy (E) consumed by a probabilistic inverter increases exponentially with the probability of its correct functioning, *p*, if the noise magnitude remains constant. For simplicity, here we assume that the energy consumption of any binary gate increases exponentially with p. It is further assumed that the power of a 1-bit CFA is normalized to 1, i.e.

$$E_{1-\mathrm{CFA}} = 1. \tag{3.34}$$

And it should be noted that all the power consumption in this chapter is normalized to a 1-bit CFA. Therefore, the power consumption of a 1-bit PFA is then:

$$\mathrm{E}_{1-\mathrm{PFA}} = \mathrm{E}_{1-\mathrm{CFA}} * \frac{\mathrm{e}^{\mathrm{p}}}{\mathrm{e}^{1}} = \mathrm{e}^{\mathrm{p}-1}. \tag{3.35}$$

For a *k*-bit CFA, its power is then:

$$\mathrm{E}_{\mathrm{CFA}} = \mathrm{k} * \mathrm{E}_{1-\mathrm{CFA}} = \mathrm{k}. \tag{3.36}$$

For a *k*-bit PFA with *m* higher bits and *n* lower bits, its power consumption is given by:

$$\mathrm{E}_{\mathrm{PFA}} = \mathrm{m} * \mathrm{E}_{1-\mathrm{CFA}} + \mathrm{n} * \mathrm{E}_{1-\mathrm{PFA}} = \mathrm{m} + \mathrm{n} * \mathrm{e}^{\mathrm{p}-1}. \tag{3.37}$$

Since adders perform a similar function (i.e., an addition), we assume that the switching activities of the gates in an adder are similar. Therefore, the power consumption is considered to be proportional to the number of logic gates for an approximate implementation. In a single-bit LOA, there is only one OR gate instead of five gates as in a conventional adder. So as a first-order estimation, the power consumption of the lower bits in a LOA is 1/5 of that in a CFA, i.e.,

$$\mathrm{E}_{\mathrm{LOA}} = \mathrm{m} * \mathrm{E}_{1-\mathrm{CFA}} + \mathrm{n} * \mathrm{E}_{\mathrm{OR}} = \mathrm{m} + 0.2 * \mathrm{n}. \tag{3.38}$$

For the three AMAs, a reduction in the number of transistors allows for a lower operating voltage, which subsequently reduces the power consumption. An application of the Inverse Discrete Cosine Transform (IDCT) is considered for evaluating the power consumption in [40]. Compared to an operating voltage of 1.13 V for the accurate IDCT operation (equivalent to an operation using the

CFA), an AMA-based IDCT operation requires a voltage of 1.04 V, 1.1 V and 1.01 V, respectively, for the three approximate implementations [40]. Therefore, the power can be estimated as

$$\mathrm{E}_{\mathrm{AMA-1}} = \mathrm{m} * \mathrm{E}_{1-\mathrm{CFA}} + \mathrm{n} * \mathrm{E}_{1-\mathrm{AMA-1}} = \mathrm{m} + \mathrm{n} * \frac{1.04^2}{1.13^2}. \tag{3.39}$$

$$\mathrm{E}_{\mathrm{AMA-2}} = \mathrm{m} * \mathrm{E}_{1-\mathrm{CFA}} + \mathrm{n} * \mathrm{E}_{1-\mathrm{AMA-2}} = \mathrm{m} + \mathrm{n} * \frac{1.1^2}{1.13^2}. \tag{3.40}$$

$$\mathrm{E}_{\mathrm{AMA-3}} = \mathrm{m} * \mathrm{E}_{1-\mathrm{CFA}} + \mathrm{n} * \mathrm{E}_{1-\mathrm{AMA-3}} = \mathrm{m} + \mathrm{n} * \frac{1.01^2}{1.13^2}. \tag{3.41}$$

Table 3.4. Power and saving per lower bit of the adder implementations (the power consumption value is normalized to 1-bit CFA).

Implementation	Power per bit	Power saving per bit
PFA	p=0.05 : 0.9512	0.0488
	p=0.028: 0.9724	0.0276
	p=0.02: 0.9802	0.0198
	p=0.01: 0.9900	0.0100
	p=0.005: 0.9950	0.0050
LOA	0.2	0.8
LIA	0	1
AMA1	0.8471	0.1529
AMA2	0.9476	0.0524
AMA3	0.7989	0.2011

For the LIA, its power consumption is simply:

$$\mathrm{E}_{\mathrm{LIA}} = \mathrm{m}. \tag{3.42}$$

The power consumption and saving of each lower bit compared to the CFA are reported in Table 3.4 for the various implementations. Given a fixed power budget, it has been shown that the MED of the PFA drastically increases with the number of lower unreliable bits, while the LIA has a significantly lower MED and the LOA performs the best with the smallest MED [18]. For a fixed MED, similarly, the power consumption of a 32-bit sequential adder can be considered.

Table 3.5 shows the power consumption of the various implementations at MED=16. There is no substantial difference in the power consumption; however, the LOA has the best performance with the LIA as a close second. The PFAs consume more power than the LIA. For more unreliable bits, a lower error rate is required, so a higher power consumption results. So far, the comparison is constrained by either a fixed power budget or MED. Next, a different metric is used to evaluate the tradeoff between power consumption and precision (as represented by NED).

Table 3.5. Power consumption of various implementations of a 32-bit adder with a largest MED of 16 (the power consumption value is normalized to 1-bit CFA).

Implementation	Features	Power consumption
PFA	m=27, n=5 : p=0.05	31.7561
	m=26, n= 6 : p=0.031	31.8169
	m=25, n= 7 : p=0.014	31.9027
LOA	m=26, n=6	27
LIA	m=28, n=4	28
AMA1	m=26, n=6	31.0823
AMA2	m=26, n=6	31.6856
AMA3	m=26, n=6	30.7933

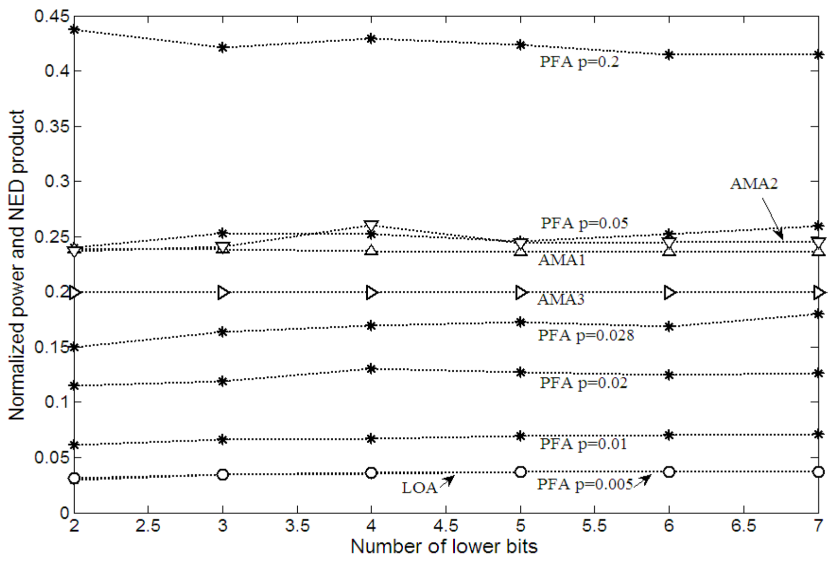


Figure 3.11. The power-NED product vs. the number of lower bits for different adder implementations and gate error rates.

This metric is given by the product of the normalized power and NED, i.e., the power-NED product. As shown in Figure 3.11, the power-NED product of an implementation has a rather constant value, which is nearly independent of the number of lower bits. For an implementation such as the PFA with a gate error rate of 0.005, its power consumption is considered high, while its NED is low. For a larger gate error rate, the power consumption decreases, while the NED increases. However, this synergetic effect of power and NED is captured by the power-NED product; the smaller the product, the better the design, in terms of a tradeoff between power consumption and precision. As shown in Figure 3.11, the AMA2 has a similar power precision tradeoff as the PFA with a gate error rate of 0.05, while the AMA3 has a better power efficiency. Also shown is that the PFA with a gate error rate of 0.005 has a comparable power-NED product to that of the LOA, so a PFA with a lower gate error rate is preferred.

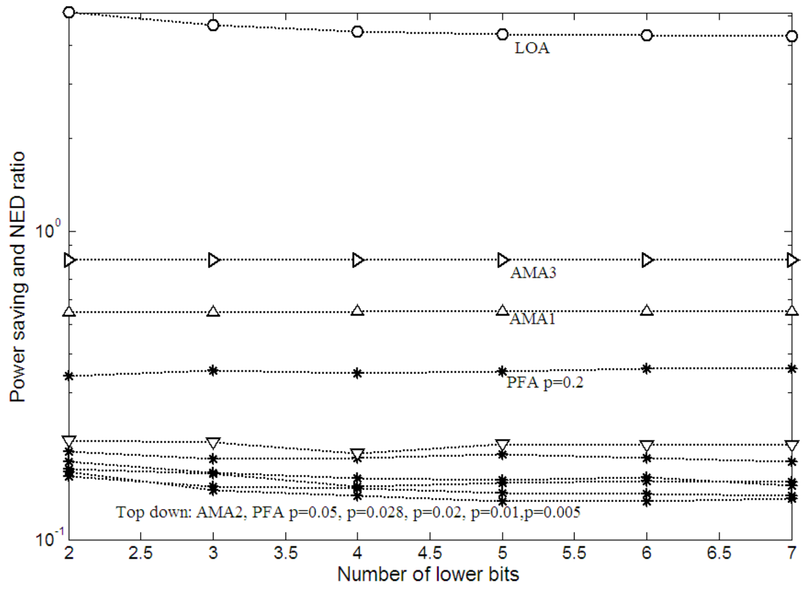


Figure 3.12. The power saving-NED ratio vs. the number of lower bits for different adder implementations and gate error rates.

This tradeoff can further be explained by the ratio of the normalized power saving and NED, i.e., the power saving per bit (compared to the CFA) divided by the NED. This power saving-NED ratio provides a quantitative measure to assess the implications of the efficiency as related to the power saving per unit of the resulting error distance. Since it is normalized, as shown in Figure 3.12, the power saving-NED ratio is also nearly independent of the number of lower bits.

Moreover from Figure 3.12, the LOA has the highest ratio; so when trading off precision for power saving, the LOA has a better efficiency than other designs considered in this chapter.

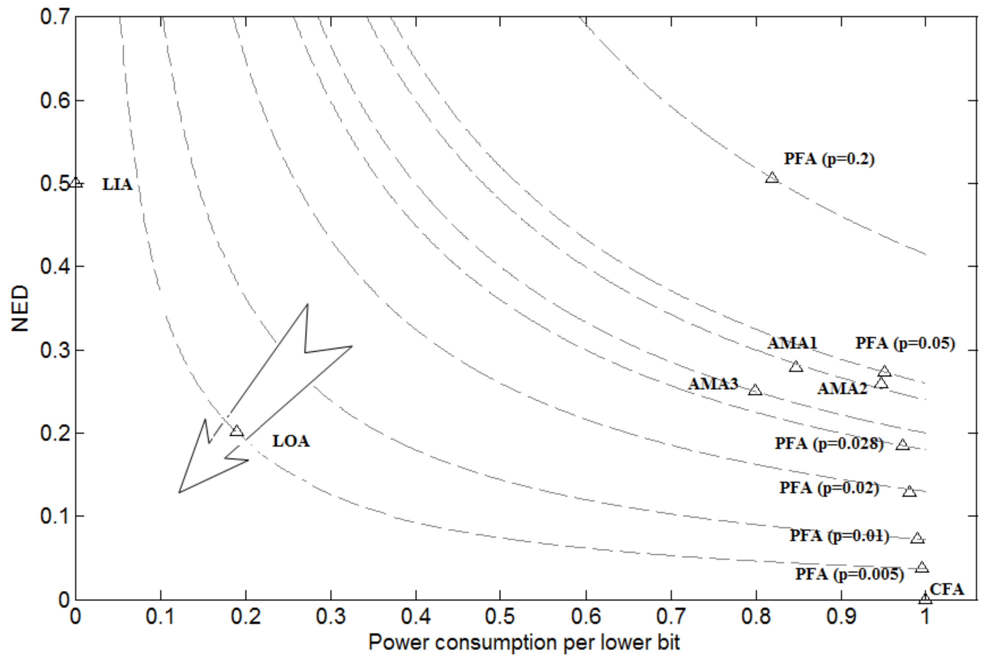


Figure 3.13. Relationship between power and precision, given by the power consumption per bit and the NED of a design. Each dashed curve indicates a value of the product of power per bit and the NED. The arrow points to the direction for a better design with a more efficient power and precision tradeoff.

The relationship between power and precision, however, is further revealed in Figure 3.13, in which the analysis of CFA and LIA is included. Power and precision are represented by the power consumption per bit and the NED of a design. While the LIA and CFA are the two extreme corner designs (with a power of 0 and an NED of 0, respectively), the other designs generally save power by allowing for some loss of precision. The LOA shows a better tradeoff between power saving and precision loss. Clearly, a design with a better power saving efficiency in terms of precision loss is desired (as indicated by the direction of the arrow in Figure 3.13). As the product of normalized power and NED is used to investigate the precision/power relationship, this imposes an equal weight on the impact of power consumption and precision. In practice, a different measure that emphasizes the importance of a particular metric (such as the power or precision) can be used for a better assessment of a design according to the specific requirement of an application.

3.5. Summary

This chapter has proposed several new metrics for evaluating approximate and probabilistic adders with respect to their reliability and power efficiency, as unified figures of merit for design assessment in inexact computing applications. The Error distance (ED) is defined as the arithmetic distance between an erroneous output and the correct one. The Mean error distance (MED) and the normalized error distance (NED) have been proposed by considering the averaging effect of multiple inputs and the normalization of multiple-bit adders. Since the NED is nearly invariant with the size of an implementation, it is useful for the reliability assessment of a specific design. To evaluate the tradeoff between power consumption and precision, the product of power and NED has been considered and the power efficiency against the precision loss is computed for gaining an insight into the effectiveness of a design. The so-called sequential probability transition matrices (SPTMs) have been used in the computation of the proposed metrics.

Using the proposed metrics, several adder implementations, namely the LOA, AMAs and PFA, are compared against a baseline implementation, the LIA. Simulation results have indicated that, compared to probabilistic adders such as the PFA, approximate adders such as the LOA and AMAs are advantageous in terms of power saving, but with a relatively low precision (comparable to that of the PFA with a high gate error rate). Probabilistic adders such as the PFA, on the other hand, are able to provide a high precision, especially for a low gate error rate, but at the cost of a relatively high power consumption. This tradeoff in precision and power are quantitatively evaluated using the proposed metric of power-NED product. The evaluation results are further supported by the analysis of the power saving and NED ratio that indicates the efficiency of a design in trading off precision for power. Although not discussed and beyond the scope of this chapter, the proposed metrics may also be useful in assessing other arithmetic circuits [43] for inexact computing and/or fault-tolerant designs [31] in nanocomputing applications.

CHAPTER 4

Design and Reliability Analysis of Multiple Valued Logic Gates using Carbon Nanotube FETs*

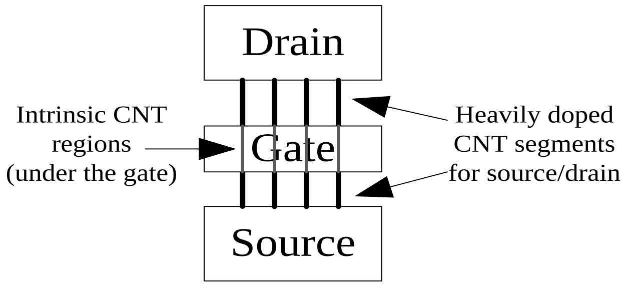
With emerging nanometric technologies, multiple valued logic (MVL) circuits have attracted significant attention due to advantages in information density and operating speed. In this chapter, a pseudo-complementary MVL design is initially proposed for implementations using carbon nanotube field effect transistors (CNTFETs). This design utilizes no resistors in its operation. To account for the properties and fabrication non-idealities of CNTFETs, a transistor-level reliability analysis is proposed to accurately estimate the error rates of MVL gates. This approach considers gate structures and their operation, so it yields a more realistic framework than a logic-level analysis of reliability. To achieve scalability, stochastic computational models (SCMs) are developed to accurately and efficiently analyze MVL gates; the extension of these models to circuits is briefly discussed.

**A version of this chapter has been accepted for publication in [21] and submitted for publication in [20].*

This chapter starts with a newly proposed CNTFET-based MVL family. Before modeling the reliability of MVL circuits, a mathematical analysis of our proposed SCM, especially compared to the traditional stochastic computing methods is presented in Section 4.2. Targeting at the reliability evaluation of the newly proposed MVL, a transistor-level analysis will be presented and SCM for MVL will be developed in Section 4.3. Finally Section 4.4 goes to the conclusion.

4.1. Design of Multiple Valued Logic Gates using CNTFETs

4.1.1 Review



Drain
Intrinsic CNT regions (under the gate)
Gate
Heavily doped CNT segments for source/drain
Source

Figure 4.1. CNTFET structure with four CNTs in the channel.

The features of high mobility of charge carriers and the reduction in subthreshold slope in gate geometry make the CNTFET a promising candidate as a post-CMOS device [10, 47]. Figure 4.1 illustrates the device structure of a CNTFET with four ideal single-wall semiconductor CNTs in the channel [48]. Current CNT fabrication processes are not ideal; in addition to the traditional CMOS fabrication defects (such as faulty open and bridge contacts), the CNTFET manufacturing process suffers from new variation challenges, such as in CNT diameters and bandgap. Therefore, many sources of uncertainty and defects affect the reliable operation of CNTFET devices.

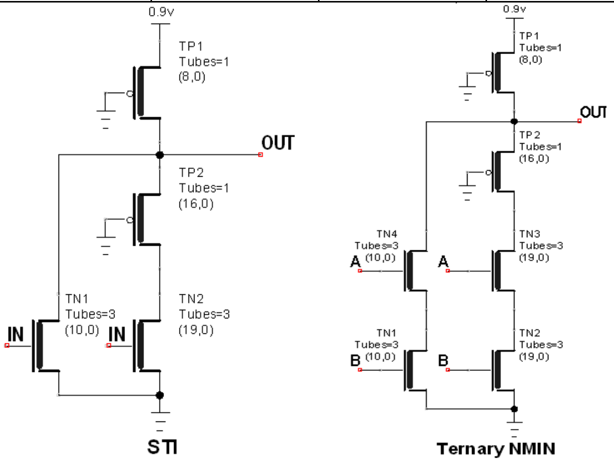
A metallic CNT is one of the most dominant defects; a CNT can be either metallic (m-CNT) or semiconducting (s-CNT) depending on its chirality. Currently, there is no known technique available to grow 100% s-CNTs. The conductivity of m-CNTs cannot be controlled by the gate due to the zero or nearzero bandgap and therefore the removal of m-CNTs or m-CNT tolerance is required.

4.1.2 Pseudo-complementary CNTFET-based MVLs

4.1.2.1 Pseudo-complementary ternary logic

Table 4.1. Truth table for three ternary inverters

Input	STI	NTI	PTI
0	2	2	2
1	1	0	2
2	0	0	0



(a) (b)

Figure 4.11(a) A pseudo-complementary STI; (b) A pseudo-complementary ternary NMIN operator.

Ternary logic gates can be designed using CNTFETs [48, 49]. In [49], a resistance-loaded design is realized for ternary logic as a basis to further use CNTFETs. This approach however suffers from the disadvantage of large area overhead (due to a large resistance) and power dissipation. However, its operational principles are valuable, which establishes some important features for CNTFET-based design. A pseudo-complementary implementation of MVL based on CNTFETs is proposed next. The proposed design replaces the resistors used in [49] with p-type CNTFETs (with the gate connected to ground), while threshold voltage operation is accomplished by adjusting the chirality and the number of CNTs in each CNTFET. This approach (referred to as *pseudo-complementary*) exploits the similarities in threshold voltage control in the p- and n-types while ensuring a correct MVL design for both ternary and quaternary logic gates.

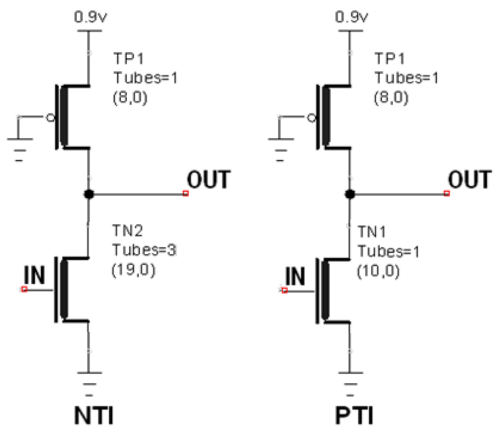


Figure 4.3 Proposed ternary pseudo-complementary NTI and PTI.

Consider first ternary operation. There are three types of ternary inverters (Table 4.1): standard ternary inverter (STI), negative ternary inverter (NTI), and positive ternary inverter (PTI). Figure 4.2(a) shows the proposed pseudo-complementary STI using CNTFETs. It consists of two n-type CNTFETs and two p-type CNTFETs. One of the CNTFETs (T_{P1}) has a chirality of (8, 0); it is used as the pull-up network, while the other three CNTFETs are used as the pull-down network. The chiralities of T_{N1} and T_{N2} are (10, 0) and (19, 0), and the corresponding threshold voltages are 0.559 V and 0.293 V respectively. Consider an input voltage V_{in}. For small values of V_{in}, both T_{N1} and T_{N2} are off. Hence, the output node (OUT) is held at V_{DD}. As V_{in} increases beyond V_{th2} (0.293 V), T_{N2} is turned on. The output voltage is determined by the resistance ratio of T_{P1}, T_{P2} and T_{N2}; therefore it is held approximately at $V_{DD}/2$ until V_{in} reaches V_{th1} (0.559 V). Once V_{in} exceeds V_{th1}, T_{N1} is turned on and the output is pulled down to nearly zero. The voltage at the output node is plotted in Figure 4.4 (obtained by HSPICE simulation). Similarly, Figure 4.3 shows the proposed pseudo-complementary NTI and PTI implementations. The HSPICE simulation results in Figure 4.4 and Figure 4.5 show the correct operations of the proposed designs.

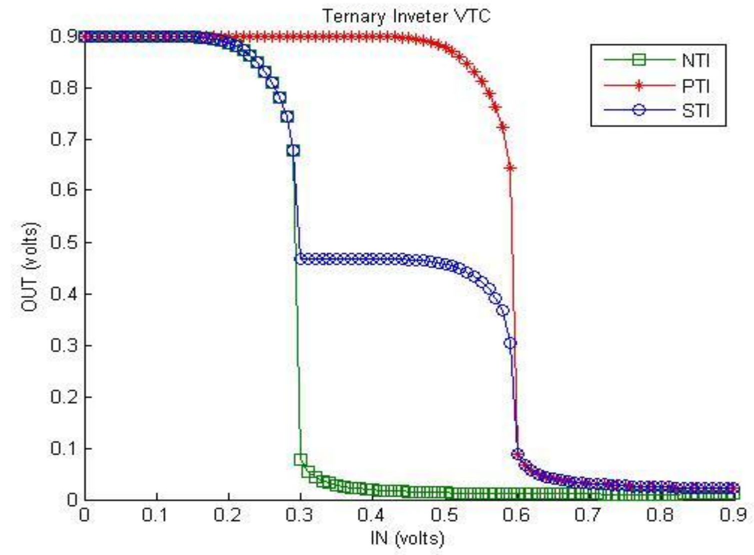


Figure 4.4 Voltage transfer diagram for the ternary inverters (STI, PTI and NTI).

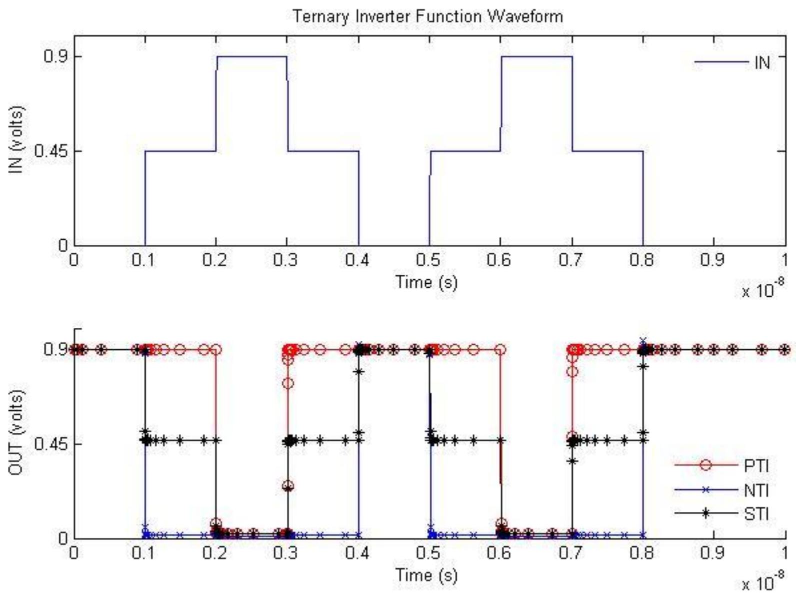


Figure 4.5 Transient simulation results of the ternary inverters.

A pseudo-complementary ternary NMIN is designed next (Figure 4.2(b)). This gate consists of six CNTFETs, with four different chiralities. In this gate, similar to the ternary STI in Figure 4.2(a), the CNTFETs with chiralities (10, 0) and (19, 0) have threshold voltages of 0.559 V and 0.293 V, respectively. HSPICE simulation (shown in Figure 4.6) confirms the correctness of the proposed design.

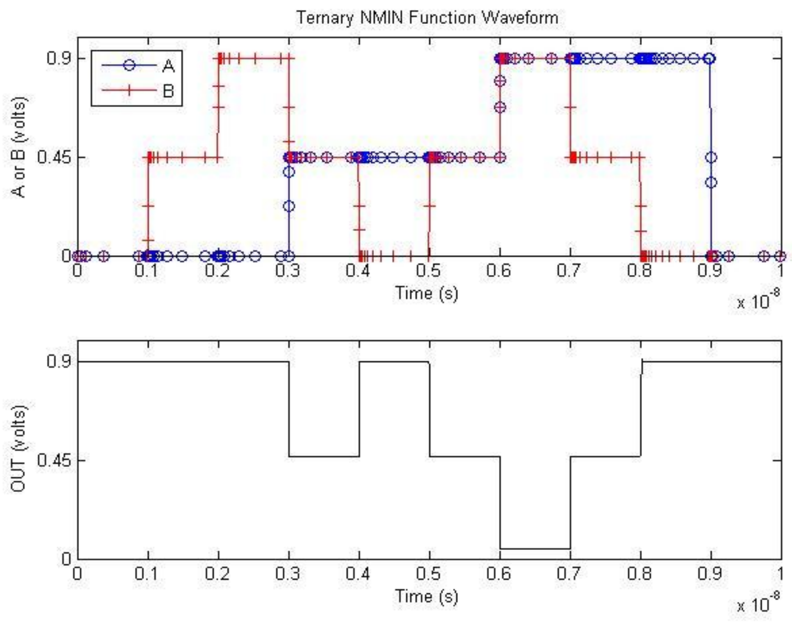
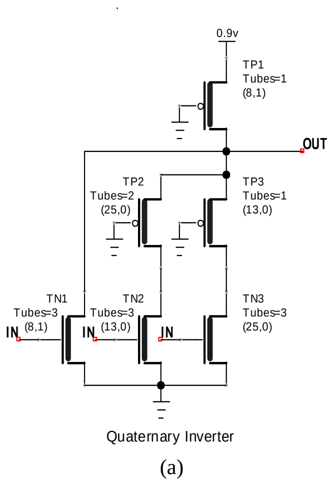


Figure 4.6 Transient simulation results of ternary NMIN operator.

4.1.2.2 Pseudo-complementary quaternary logic

Similarly, pseudo-complementary quaternary logic gates are designed in this section. Figure 4.9(a) shows a pseudo-complementary CNTFET quaternary inverter, while Figure 4.7(b) shows a pseudo-complementary CNTFET quaternary NMIN operator.

The inverter consists of three n-type CNTFETs and three p-type CNTFETs, each with a different chirality; the NMIN operator consists of six n-type CNTFETs and three p-type CNTFETs. Each of the p-type CNTFETs has a distinct chirality, while the six n-type CNTFETs have three chiralities. Figure 4.8 shows the voltage transfer diagram of the quaternary inverter. Compared to the simulation results of the ternary inverter (Figure 4.4), Figure 4.8 shows the reduced noise margin for the quaternary logic. Figure 4.9 shows the transient simulation results (simulated by HSPICE).



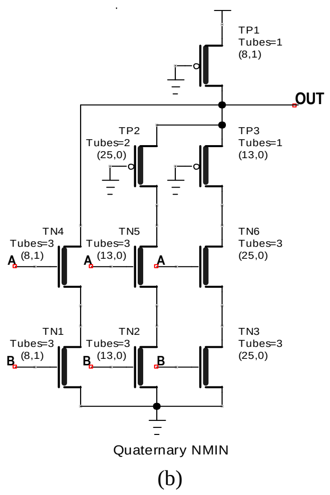


Figure 4.7 (a) A pseudo-complementary quaternary inverter; (b) A pseudo-complementary quaternary NMIN operator.

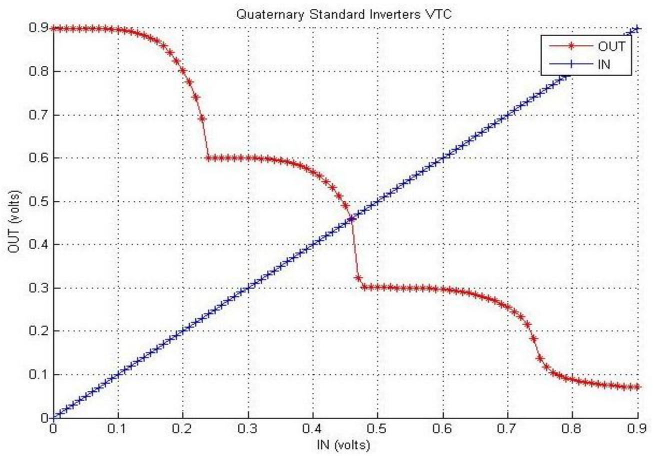


Figure 4.8 Voltage transfer diagram for the quaternary inverter of Figure 4.7(a).

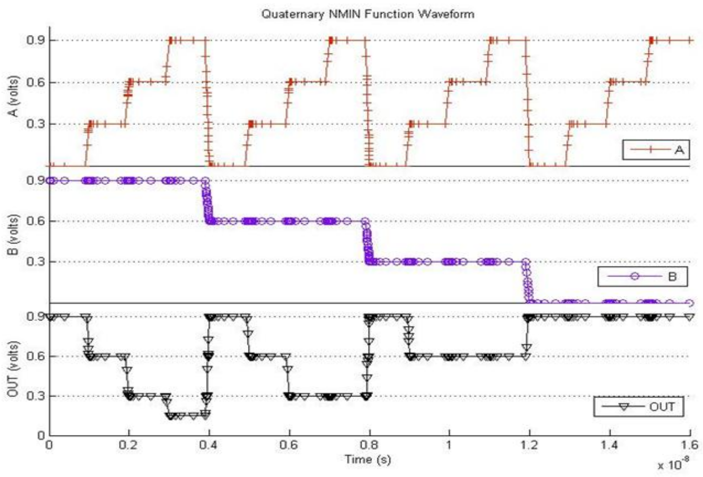


Figure 4.9 Transient simulation results of quaternary inverter and NMIN operator.

4.2. Stochastic Logic using Non-Bernoulli Sequences

4.2.1 Stochastic Logic

In stochastic computation, signal probabilities are encoded into binary bit streams, i.e., serially in the time domain. Randomly generated bit streams are used to encode signal probabilities; a specific probability is represented by a number of bits sequence to a value that is usually in proportion to the mean number of 1's in a bit stream. Figure 4.10 shows a stochastic encoding and an inverter. As Boolean operations can be mapped to arithmetic operations, the inverter probabilistically implements the complement operation of Rule I. Note that in Figure 4.10, a sequence length of 10 bits is used for illustration purposes; a larger sequence length is usually needed in practice.

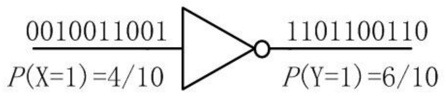


Figure 4.10 An inverter and a stochastic encoding.

Stochastic computation transforms Boolean logic operations into probabilistic computations in the real domain. Although each binary bit is processed by a Boolean gate, signal operations are no longer Boolean in nature, but they are arithmetic computations by stochastic logic. Bernoulli sequences are often used as binary bit streams in stochastic computation [16, 44]. In a Bernoulli sequence, every bit is independently generated with a probability p. The mean and variance of the number of 1's in an N-bit Bernoulli sequence are respectively given by

$$\mu = Np, \tag{4.1}$$

and

$$v = Np(1-p). \tag{4.2}$$

For the inverter of Figure 4.1, if the input probability is a, the mean number of 1's in its output sequence is

$$\mu_1 = N(1-a), \tag{4.3}$$

and the variance is

$$v_1 = Na(1-a). \tag{4.4}$$

This is the same as the variance of the input sequence.

Complex arithmetic operations can be implemented by simple stochastic logic. According to *Rule II*, for instance, multiplication can be implemented by an AND gate, as shown in Figure 4.2(b). In this multiplication, the input binary streams must not be correlated for a correct computation. However, the bit-wise dependencies between the input random binary streams can be used to yield new stochastic logic models that account for the statistical correlation in input signals. This is shown in Figure 4.2(a) as a general stochastic model of AND in which the two input signals may be correlated.

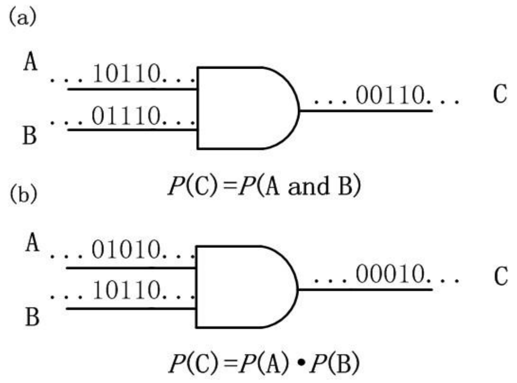


Figure 4.11 Stochastic AND logic: (a) the general model; (b) the special case of multiplication, when the two inputs are statistically independent.

If the inputs of the AND are two independent Bernoulli sequences with generating probabilities a and b respectively, the mean number of 1's in the output sequence is:

$$\mu_2 = Nab, \tag{4.5}$$

and the variance is given by:

$$v_2 = Nab(1 - ab). \tag{4.6}$$

For the AND gate in Figure 4.11(a) with possibly correlated inputs,

$$P(C = 1) = P(A = 1, B = 1) = P(A = 1)P(B = 1|A = 1). \tag{4.7}$$

Let $a = P(A = 1)$, $b = P(B = 1)$ and $p_c = P(B = 1|A = 1)$; then

$$P(C = 1) = ap_c. \tag{4.8}$$

The use of Bernoulli sequences as inputs results in a Bernoulli sequence at the output with a generating probability given by (4.8); therefore, the mean number of 1's in the output sequence and its variance are given by:

$$\mu_{2,g} = Nap_c, \tag{4.9}$$

and

$$v_{2,g} = Nap_c(1 - ap_c), \tag{4.10}$$

respectively.

The use of Bernoulli sequences however incurs a large computational overhead that severely limits its application for reliability analysis. This aspect is addressed through the use of non-Bernoulli sequences, as discussed next.

4.2.2 Non-Bernoulli Sequences

In this work, non-Bernoulli sequences are used for reducing the computational complexity and inaccuracy. Specifically, each initial input stochastic sequence contains a fixed number of 1's and the positions of the 1's are determined by a random permutation. For a given probability p and a sequence length of *N* bits, the number of 1's to be generated is given by *Np*. When *Np* is not an integer, it must be rounded to an integer, thus introducing a quantization error into the representation. The effect of quantization errors is discussed in a later Section; the output distributions of the inverter and AND gate, when non-Bernoulli sequences are used as inputs, are treated in more detail next.

For an inverter, assume that the input has a probability of a to be "1"; so *Na* is the number of 1's in the input sequence of *N* bits. Then the expected value of 1's in the output sequence is given by:

$$\mu_1' = N(1-a). \tag{4.11}$$

Since there is no variation in the input, the variance in the output is considered to be 0, i.e.,

$$v_1' = 0. \tag{4.12}$$

For an AND gate, the use of the non-Bernoulli sequences resembles von Neumann's NAND multiplexing technique, as discussed in [45, 46] for fault-tolerant logic design. The following Lemma shows that its output follows approximately a Gaussian distribution when the sequence length *N* is large.

Lemma 1: For an AND gate, assume that the two inputs are "1" with probabilities a and b and represented by non-Bernoulli sequences of *N* bits (as random permutations of fixed numbers of 1's and 0's). For a large *N*, the output sequence follows a Gaussian distribution with a mean number of 1's given by:

$$\mu_2' = Nab, \tag{4.13}$$

and a variance:

$$v_2' = Na(1-a)b(1-b). \tag{4.14}$$

Proof: The two input probabilities *a* and *b* give *r=a*N* and *s=b*N* as the numbers of 1's in the input sequences. In these two inputs, the numbers of possible permutations are:

$$C_a = \binom{N}{r} = \frac{N!}{r! * (N-r)!}, \tag{4.15}$$

and

$$C_b = \binom{N}{s} = \frac{N!}{s! * (N-s)!}, \tag{4.16}$$

respectively. Assume that the AND gate produces *t* 1's in the output sequence; then, the number of permutations that causes this occurrence, can be obtained by combinatorial analysis [45, 46]. This leads to:

$$C_o = \binom{N}{t} * \binom{N-t}{r-t} * \binom{N-r}{s-t} = \frac{N!}{t! * (r-t)!(s-t)!(N-r-s+t)!} \tag{4.17}$$

The probability that *t* 1's result in the output sequence, is given by the number of output permutations divided by the total possible number of input permutations, i.e.,

$$P(t) = \frac{C_o}{C_a * C_b} = \frac{r!(N-r)!s!(N-s)!}{t!(r-t)!(s-t)!(N-r-s+t)!N!}. \tag{4.18}$$

Assume that the expected output probability is *z*, and therefore

$$z = \frac{t}{N}. \tag{4.19}$$

As per [45], the application of Stirling's formula results in:

$$P(z) \sim \frac{1}{\sqrt{2\pi N}} \sqrt{\beta} e^{-\theta N}, \tag{4.20}$$

where

$$\beta \sim \frac{1}{a(1-a)b(1-b)}, \quad (4.21)$$

$$\theta \sim \frac{(z-ab)^2}{2a(1-a)b(1-b)}. \quad (4.22)$$

(4.20), (4.21) and (4.22) indicate that the output sequence follows approximately a Gaussian distribution with a mean number of 1's given by (4.13) and a variance given by (4.14). □

4.2.3 Non-Bernoulli vs. Bernoulli Sequences

Next, the comparison between the use of Bernoulli and non-Bernoulli input sequences in stochastic logic is pursued. For an inverter, it is easy to find that (4.11) = (4.3) and (4.12) = 0. This indicates that the use of non-Bernoulli input sequences results in a deterministic output value equal to the mean value of the one by using Bernoulli input sequences. For an AND gate, the following theorem applies for independent inputs.

Theorem 1: Compared to the case when Bernoulli sequences are used to represent the initial input probabilities, the use of large non-Bernoulli sequences as random permutations of fixed numbers of 1's and 0's results in an output sequence with the same mean number of 1's and a smaller variance for an AND gate when its inputs are independent.

Proof: From Lemma 1, it can be seen that (4.13) = (4.5) and

$$v_2 - v_2' = Nab(1-ab) - Na(1-a)b(1-b) = Nab(a(1-b) + b(1-a)) \geq 0, \quad (4.23)$$

so proving the theorem. □

The general case of correlated inputs is considered as follows. When non-Bernoulli sequences are used as inputs, the random permutation allows for some randomness in the inputs, albeit with a correlation between them. Without loss of generality, assume that input A is first generated; input B is then generated conditionally on *A*. For the 1's in the sequence of *A*, further assume that the corresponding bits in *B* are generated as a Bernoulli sequence with probability p_c. For the 0's in the sequence of *A*, subsequently, the number of 1' in the corresponding bits in B is actually determined due to the nature of the non-Bernoulli sequence used to represent the input *B*. Since the number of 1's in the sequence of *A* is *Na*, the mean number of 1's in the corresponding bits in *B* and its variance are given by:

$$\mu'_{2,g} = Nap_c, \tag{4.24}$$

and

$$v'_{2,g} = Nap_c(1 - p_c). \tag{4.25}$$

The combinations of 1's in inputs *A* and *B* produce the 1's in the output sequence, so the mean number of 1's at the output and the variance are given by (4.24) and (4.25) respectively for an AND gate with non-Bernoulli input sequences that may be correlated.

Hence, it can be seen that (4.9) = (4.15) and from (4.10) and (4.25),

$$v_{2,g} - v'_{2,g} = Nap_c(1 - ap_c) - Nap_c(1 - p_c) = \ Nap_c{}^2(1 - p_c) \geq 0. \tag{4.26}$$

This indicates that, when compared to Bernoulli input sequences, the use of non-Bernoulli input sequences as random permutations of fixed numbers of 1's and 0's results in an output sequence with the same mean number of 1's and a smaller variance for an AND gate when its inputs may be correlated.

Any logic function can be implemented with inverters and AND gates; so, a smaller variance in the output of AND gates (as achieved by using the non-Bernoulli inputs) will result in a smaller variance in the output of a function implemented with inverters and AND gates. Also, the same mean value results from the use of non-Bernoulli and Bernoulli inputs. Therefore in a logic network, the use of non-Bernoulli and Bernoulli sequences as initial inputs will produce evaluation results with the same mean, but different variance; the former method results in a smaller variance than the latter method.

We conjecture this result as follows: compared to the case when Bernoulli sequences are used to represent the initial input probabilities, the use of large non-Bernoulli sequences as random permutations of fixed numbers of 1's and 0's results in an output sequence with the same mean number of 1's and a smaller variance for a combinational logic network.

4.3. Reliability Analysis of Multiple Valued Logic Gates using CNTFETs

4.3.1 Fault Models for CNTFETs

4.3.1.1 The ACCNT technique

Since techniques such as the selective chemical etching [50] are not perfect and cannot guarantee a robust circuit fabrication, a VLSI-compatible methodology referred to as asymmetrically-correlated carbon nanotubes (ACCNTs) has been proposed for reliable circuit design [51].

As a metallic-CNT tolerant technique, ACCNT can tolerate short defects (as caused by metallic CNTs) by utilizing uncorrelated stacks of CNTFETs in series. Furthermore, the ACCNT technique uses correlated branches of parallel CNTFETs to increase the device drive strength without degrading the failure rate [51]. ACCNT requires a conventional CNTFET process, and does not conflict with other metallic removal or breakdown solutions. Although this technique incurs a large area overhead, it has been shown to be efficient in tolerating metallic-CNTs at wafer level in the manufacturing process flow. Therefore, the ACCNT technique can effectively enhance yield.

4.3.1.2 Open and short defect probabilities for ACCNT

An accurate mathematical model that considers the density variation in ACCNTs has been proposed in [52]. Let the CNT placement be totally random (i.e., at a probability of 0.5 at any given site), the average density of the CNTs be D CNTs/µm, the window size (or CNTFET width) be W µm; then the average number of CNTs in each CNTFET is given by

$$\bar{N}_{CNT} = WD. \tag{4.27}$$

For a placement probability of 0.5, $2\bar{N}_{CNT}$ CNTs need to be placed in the channel for an expected value given by (4.27). The probability of an open defect is then given by:

$$P_{O(CNFET)} = 0.5^{2\bar{N}_{CNT}}. \tag{4.28}$$

A CNTFET is defective (**s**hort) when at least one metallic CNT is present; so by considering the CNT density variation,

$$\bar{P}_{S(CNFET)} = \sum_{k=1}^{2\bar{N}_{CNT}} \frac{(2\bar{N}_{CNT})!}{k!(2\bar{N}_{CNT}-k)!} 0.5^{2\bar{N}_{CNT}} [1-(1-P_M)^k], \tag{4.29}$$

where P_M denotes the probability of a CNT to be metalic.

For uncorrelated CNTFETs in series of N_{STACK}stacks, the probability that an ACCNT-based transistor has short defects, is given by

$$P_{S(ACCNT)} = \bar{P}_S^{\,N_{STACK}}. \tag{4.30}$$

The probability that an ACCNT-based transistor has open defects, is

$$P_{O(ACCNT)} = 1-(1-P_O)^{N_{STACK}}. \tag{4.31}$$

In the general case, a metallic CNT has a probability of around $\frac{1}{3}$, i.e., $P_M = \frac{1}{3}$. Assume $\overline{N}_{CNT} = 4$ and $N_{STACK} = 14$, the following defect probabilities are found:

$$P_{S(ACCNT)}=0.046 \text{ and } P_{O(ACCNT)}=0.053. \tag{4.32}$$

For simplicity and without loss of generality, an independent defect rate of $P_{S(ACCNT)}=P_{O(ACCNT)}=0.05$ is considered in the following calculations.

4.3.2 Reliability analysis of MVL gates

Most previous methods for reliability evaluation are based on the simple assumption that every gate fails with a given probability. This assumption is common due to its simplicity in a mathematical model and efficiency in a gate-level evaluation. However, this assumption is not fully applicable when gate complexity is taken into account; for example, in a binary CMOS circuit, an inverter consists of one PMOS transistor and one NMOS transistor, while an AND gate usually consist of three PMOS transistors and three NMOS transistors. Due to the non-idealities in fabrication and operational conditions (such as induced noise), it is evident that an AND gate has a larger probability to fail compared to an inverter.

This section proposes a transistor level analysis to estimate the probabilistic behavior of MVL gates; with no loss of generality, the pseudo-complementary CNTFET logic gates proposed in the previous section are utilized to illustrate the proposed method. However, the method is sufficiently flexible that it can easily be extended and generalized to other MVLs.

4.3.2.1 Ternary logic

As discussed previously, open and short defects can be modeled on a probabilistic basis. Consider the function of the ternary inverter in Figure 4.2(a). When the input voltage is lower than 0.293 V (logic 0), both T_{N1} and T_{N2} are expected to be turned off. However, on a probabilistic basis by taking into account the error probability of each CNTFET, the correct response of the inverter (so by considering all CNTFETs in this gate) is given by a probability of only $(1-\mathrm{Po})(1-\mathrm{Ps})^2$. To consider other defect scenarios, for example, there is a probability of $\mathrm{Po}(1-\mathrm{Ps})^2$ for $\mathrm{T_{P1}}$ to be open, $\mathrm{T_{N1}}$ to be operating correctly or open, and $\mathrm{T_{N2}}$ to be operating correctly or open. The output can also be floating and therefore, its current value is determined by the previous value. The detailed analysis of all scenarios and the corresponding probabilities are given in Table 4.2.

Table 4.2. All possible scenarios for an STI when Input<0.3V (Logic 0) ("floating" indicates both pull-up and pull-down networks are off. N: normal; S: short defect; O: open defect; X: don't care; '/' means 'or'.)

Scenario	Probability	OUT logic
TP1:N/S;TP2:X;N1:N/O;TN2:N/O.	$(1-Po)(1-Ps)^2$	2
TP1:O;TP2:X; TN1:N/O;TN2:N/O.	$Po(1-Ps)^2$	Floating
TP1:X;TP2:X; TN1: S; TN2: X.	Ps	0
TP1:O;TP2:O; TN1: N/O; TN2: S.	$Po^2(1-Ps)Ps$	Floating
TP1:S/N;TP2:O;TN1: N/O; TN2:S.	$(1-Po)Po(1-Ps)Ps$	2
TP1:N/S;TP2:N/S;TN1:N/O;TN2: S.	$(1-Po)^2Po(1-Ps)$	1
TP1:O;TP2:N/S; TN1: N/O; TN2: S.	$Po(1-Po)(1-Ps)Ps$	0

Based on Table 4.2, the output probability distributions for an input at logic 0 can then be calculated (as shown in Table 4.3).

Similarly, the output probabilities of a ternary inverter when the input is at logic 1 and 2 can be found in Tables 4.4 and 4.5, respectively.

By combining for an input the values of the output probabilities (given in Table 4.3, Table 4.4 and Table 4.5), a single comprehensive table can be generated. This comprehensive table describes the probabilistic mapping from the primary inputs to the primary output of a gate.

Table 4.3. Output probabilities of an STI when Input<0.3V (Logic 0)

Output	**Probability**
0	$\mathrm{Prob}_{0\|0} = Ps + Po(1-Po)(1-Ps)Ps$
1	$\mathrm{Prob}_{1\|0} = (1-Po)^2Po(1-Ps)$
2	$\mathrm{Prob}_{2\|0} = (1-Po)(1-Ps)^2 + (1-Po)Po(1-Ps)Ps$
Floating	$\mathrm{Prob}_{\mathrm{floating}\|0} = Po(1-Ps)^2 + Po^2(1-Ps)Ps$

Table 4.4. Output probabilities of an STI when 0.3V<Input<0.6V (Logic 1)

Output	**Probability**
0	$\mathrm{Prob}_{0\|1} = \mathrm{Ps}+\mathrm{Po}(1-\mathrm{Ps})(1-\mathrm{Po})^2$
1	$\mathrm{Prob}_{1\|1} = (1-\mathrm{Po})^2(1-\mathrm{Ps})^2$
2	$\mathrm{Prob}_{2\|1} = (1-\mathrm{Po})(1-(1-\mathrm{Po})^2)(1-\mathrm{Ps})$
Floating	$\mathrm{Prob}_{\mathrm{floating}\|1} = \mathrm{Po}(1-\mathrm{Ps})(1-(1-\mathrm{Po})^2)$

Table 4.5. Output probabilities of an STI when Input>0.6V (Logic 2)

Output	**Probability**
0	$\mathrm{Prob}_{0\|2} = 1-\mathrm{Po}+\mathrm{Po}^2(1-\mathrm{Po})^2$
1	$\mathrm{Prob}_{1\|2} = \mathrm{Po}(1-\mathrm{Po})^3$
2	$\mathrm{Prob}_{2\|2} = (1-\mathrm{Po})\mathrm{Po}(1-(1-\mathrm{Po})^2)$
Floating	$\mathrm{Prob}_{\mathrm{floating}\|2} = \mathrm{Po}^2(1-(1-\mathrm{Po})^2)$

So for a ternary inverter, the input can have three different logic values, while the output can have four different values, including the additional floating scenario. When floating, the gate operates as a DRAM (Dynamic Random Access Memory) and the current output is determined by the previous value. The vector that estimates the probability of the previous value to be the logic values of '0', '1' and '2' is therefore given by

$$\boldsymbol{P_{pre}} = [\mathrm{p}_{\mathrm{pre=0}} \quad \mathrm{p}_{\mathrm{pre=1}} \quad \mathrm{p}_{\mathrm{pre=2}}]. \tag{4.33}$$

So for an input 'i', the probability for the output being 'j' is calculated as

$$P_{j|i} = Prob_{j|i} + Prob_{floating|i} * p_{pre=j}. \tag{4.34}$$

Assume $p_{pre=0} = p_{pre=1} = p_{pre=2} = \frac{1}{3}$; using the analysis in the previous section, Ps=Po=0.05. So, the values of the conditional probabilities are now given in Table 4.6:

Table 4.6. Output probabilities of the proposed STI

Input	Output=0	Output=1	Output=2
0	$P_{0\|0} = 0.0673$	$P_{1\|0} = 0.0579$	$P_{2\|0} = 0.8748$
1	$P_{0\|1} = 0.0945$	$P_{1\|1} = 0.8160$	$P_{2\|1} = 0.0895$
2	$P_{0\|2} = 0.9524$	$P_{1\|2} = 0.0429$	$P_{2\|2} = 0.0047$

Table 4.6 can then be expressed by a single equation given by:

$$Output_{inverter} = \left(2 * P_{2|0} + 1 * P_{1|0} + 0 * P_{0|0}\right) * P_{input=0}$$

$$+\left(2 * P_{2|1} + 1 * P_{1|1} + 0 * P_{0|1}\right) * P_{input=1}$$

$$+\left(2 * P_{2|2} + 1 * P_{1|2} + 0 * P_{0|2}\right) * P_{input=2}$$

$$= \sum_{i=0}^{2}(P_{input=i} * \sum_{j=0}^{2} j * P_{j|i}) \qquad (4.35)$$

By a similar process, the transistor-level equation for any ternary gate can be found. Table 4.7 shows the results for the NMIN operator of Figure 4.2(b), while Table 4.8 shows the results for the NTI and the PTI of Figure 4.3.

Table 4.7. Output probabilities of the NMIN operator

Inputs	Output=0	Output=1	Output=2
00	$P_{0\|00} = 0.0192$	$P_{1\|00} = 0.0189$	$P_{2\|00} = 0.9619$
01	$P_{0\|01} = 0.0206$	$P_{1\|01} = 0.0586$	$P_{2\|01} = 0.9208$
02	$P_{0\|02} = 0.0648$	$P_{1\|02} = 0.0603$	$P_{2\|02} = 0.8749$
10	$P_{0\|10} = 0.0206$	$P_{1\|10} = 0.0586$	$P_{2\|10} = 0.9208$
11	$P_{0\|11} = 0.0477$	$P_{1\|11} = 0.8148$	$P_{2\|11} = 0.1375$
12	$P_{0\|12} = 0.0906$	$P_{1\|12} = 0.7780$	$P_{2\|12} = 0.1314$
20	$P_{0\|20} = 0.0648$	$P_{1\|20} = 0.0603$	$P_{2\|20} = 0.8749$
21	$p_{0\|21} = 0.0906$	$P_{1\|21} = 0.7780$	$P_{2\|21} = 0.1314$
22	$p_{0\|22} = 0.9069$	$P_{1\|22} = 0.0797$	$P_{2\|22} = 0.0134$

Table 4.8. Output probabilities of the NTI and PTI operator

Type	Input	Output=0	Output=1	Output=2
NTI	0	$P_{0\|0} = 0.0738$	$P_{1\|0} = 0$	$P_{2\|0} = 0.9262$
	1	$P_{0\|1} = 0.9512$	$P_{1\|1} = 0$	$P_{2\|1} = 0.0488$
	2	$P_{0\|2} = 0.9512$	$P_{1\|2} = 0$	$P_{2\|2} = 0.0488$
PTI	0	$P_{0\|0} = 0.0738$	$P_{1\|0} = 0$	$P_{2\|0} = 0.9262$
	1	$P_{0\|1} = 0.0738$	$P_{1\|1} = 0$	$P_{2\|1} = 0.9262$
	2	$P_{0\|2} = 0.9512$	$P_{1\|2} = 0$	$P_{2\|2} = 0.0488$

4.3.2.2 Quaternary logic

The previously designed quaternary inverter is considered as a further example to show that the proposed method is applicable at a higher base of 4.

Table 4.9. All possible scenarios of quaternary inverter when Input<0.3V (Logic 0). ("Floating" indicates that both pull-up and pull-down networks are turned off. N: normal; S: short defect; O: open defect; X: don't care; '/' means 'or'.)

Scenario	Probability	OUT logic
TP1:X;TP2:X;TP3:X; TN1:S;TN2:X;TN3:X;	Ps	0
TP1:N/S;TP2:N/S;TP3:N/S; TN1: N/O; TN2:S;TN3:S.	$(1-\mathrm{Po})^3(1-\mathrm{Ps})\mathrm{Ps}^2$	1
TP1:N/S;TP2:N/S;TP3:N/S; TN1:N/O;TN2:S;TN3:N/O.	$(1-\mathrm{Po})^3(1-\mathrm{Ps})\mathrm{Ps}(1-\mathrm{Ps})$	1
TP1:N/S;TP2:N/S;TP3:O; TN1: N/O; TN2:S;TN3:X.	$(1-\mathrm{Po})^2\mathrm{Po}(1-\mathrm{Ps})\mathrm{Ps}$	1
TP1:N/S;TP2:N/S;TP3:N/S; TN1:N/O;TN2:N/O;TN3:S.	$(1-\mathrm{Po})^3(1-\mathrm{Ps})^2\mathrm{Ps}$	2
TP1:N/S;TP2:O;TP3:N/S; TN1: N/O; TN2:S;TN3:S.	$(1-\mathrm{Po})^2\mathrm{Po}(1-\mathrm{Ps})\mathrm{Ps}^2$	2
TP1: N/S; (TP2 OR TN2): O; (TP3 OR TN3): O;	$(1-\mathrm{Po})(1-\mathrm{Ps})(1-(1-\mathrm{Po})\mathrm{Ps})^2$	3
TP1: O; (TP2-TN2 PATH OR TP3-TN3 PATH): S;	$\mathrm{Po}(1-\mathrm{Ps})[1-(1-(1-\mathrm{Po})\mathrm{Ps})^2]$	0
TP1: O; (TP2 OR TN2): O; (TP3 OR TN3): O;	$Po(1-Ps)(1-(1-Po)Ps)^2$	Floating

Using the expressions in Table 4.9, the output probability distributions for an input at logic 0 can be calculated as shown in Table 4.10.

Table 4.10. Output probabilities of quaternary inverter when Input<0.3V (Logic 0)

Output	Probability
0	$\mathrm{Prob}_{0\|0} = \mathrm{Ps} + \mathrm{Po}(1-\mathrm{Ps})[1-(1-(1-\mathrm{Po})\mathrm{Ps})^2]$
1	$\mathrm{Prob}_{1\|0} = (1-\mathrm{Po})^2(1-\mathrm{Ps})\mathrm{Ps}$
2	$\mathrm{Prob}_{2\|0} = (1-\mathrm{Po})^2(1-\mathrm{Ps})[1-(1-\mathrm{Po})\mathrm{Ps}]\mathrm{Ps}$
3	$\mathrm{Prob}_{3\|0} = (1-\mathrm{Po})(1-\mathrm{Ps})(1-(1-\mathrm{Po})\mathrm{Ps})^2$
Floating	$\mathrm{Prob}_{\mathrm{floating}\|0} = \mathrm{Po}(1-\mathrm{Ps})(1-(1-\mathrm{Po})\mathrm{Ps})^2$

As previously discussed, the output probabilities for input = 1, 2 and 3 are calculated and shown in Table 4.11.

Table 4.11. Output probabilities of a quaternary inverter

Input	Output=0	Output=1	Output=2	Output=3
0	$P_{0\|0} = 0.0652$	$P_{1\|0} = 0.0536$	$P_{2\|0} = 0.0516$	$P_{3\|0} = 0.8296$
1	$P_{0\|1} = 0.0942$	$P_{1\|1} = 0.0440$	$P_{2\|1} = 0.7769$	$P_{3\|1} = 0.0849$
2	$P_{0\|2} = 0.0972$	$P_{1\|2} = 0.8146$	$P_{2\|2} = 0.0795$	$P_{3\|2} = 0.0087$
3	$P_{0\|3} = 0.9525$	$P_{1\|3} = 0.0429$	$P_{2\|3} = 0.0042$	$P_{3\|3} = 0.0004$

4.3.2.3 Generalized transistor level analysis

As indicated by (4.35), for any MVL gate with $d+1$ possible logic values (0, 1, 2, …, d), the final output of this gate can be described by the following equation:

$$\text{Output} = \sum_j P(input = j) * \left(\sum_{i=0}^{d} i * P(i|j)\right), \qquad (4.36)$$

where *j* is the index to an input from the set of all input vectors. (4.36) has significant implications on the reliability evaluation of a MVL gate. So rather than simply assuming that the gate is affected by a given probability, (4.36) describes the probabilistic behavior based on a physical structure, therefore providing a more detailed characterization and evaluation (as detailed in the next section).

4.4 Stochastic Computatioanl Models for MVLs

In this section, different stochastic computation models (SCMs) are analyzed for reliability evaluation of ternary/quaternary inverters, an arbitrary MVL gate and a MVL combinational circuit.

4.4.1 Ternary and quaternary inverters

As discussed in [9, 20], stochastic computation transforms Boolean logic operations into a probabilistic computation in the real domain. In this process the so-called stochastic multiplexer plays an important role. A *stochastic multiplexer* is equivalent to a weighted adder and its function can be described by:

$$Output = \sum_{j \in A} P_j * I_j, \tag{4.37}$$

where A represents the set of all combinations of the control bits, P_j is the probability of the control vector being j, and I_j represents the input value corresponding to the scenario for the control vector j.

(4.37) can then be used as basis for a stochastic computational model (SCM) as applicable to MVL gates. The SCM for the proposed STI is shown in Figure 4.12(a). The sequences of all three inputs can be determined using Table 6. In this case, a sequence length of 10000 is employed. The first sequence corresponding to input = '0', as discussed previously, consists of 673 0's, 579 1's and 8747 2's. The second input sequence consists of 945 0's, 8160 1's and 895 2's. The third input sequence consists of 9523 0's, 429 1's and 47 2's.

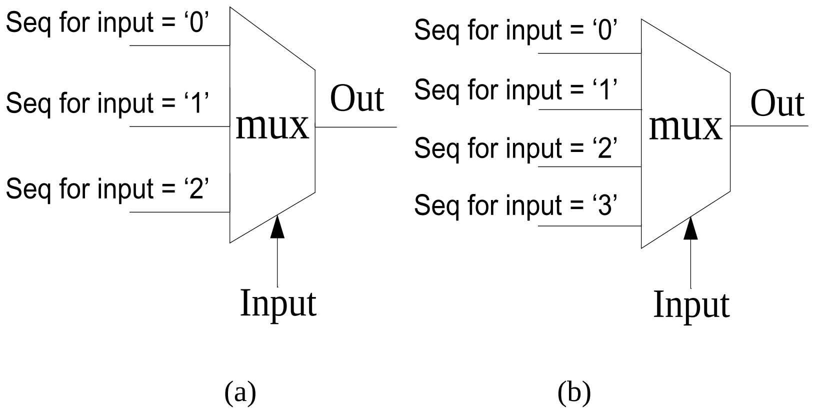


Figure 4.12 Stochastic computational model for (a) a ternary inverter (b) a quaternary inverter

Therefore, each sequence can be described by the following expression:

$$\text{Output}(j) = \sum_{i=0}^{2} i * P(i|j) \tag{4.38}$$

With the stochastic multiplexer function and (4.38), the equation of the final output of this gate is given by:

$$\text{Output} = \sum_{j=0}^{2} P(input = j) * \text{Output}(j) \tag{4.39}$$

By combining (4.38) and (4.39), it is clear that Figure 4.12(a) is an implementation of (4.35) (or the more general expression given by (4.36)). Similarly to the previous discussion for Table 4.11, a general stochastic computation model can also be obtained for a quaternary inverter. Figure 4.12(b) shows the SCM for a quaternary inverter.

The ternary and quaternary NMIN operators can also be evaluated by the proposed approach. In fact, the SCM represents a general framework that is applicable to the reliability evaluation of any MVL gate. Due to space limitations, however, this is not discussed in detail.

4.4.2 SCMs for combinational MVLs

Based on the proposed SCM, a stochastic computational network can be constructed using the SCMs of the gates for circuit reliability evaluation. As discussed in [9, 20], the computational network is a nonlinear structure constituted by SCMs. Feeding stochastic input sequences into the network and propagating them from the primary inputs to the outputs calculate the output probabilities. A distinguishing feature of the SCM approach is that it handles reconvergent fanouts at a very small effort; when signals are processed in the form of bit streams (such as consisting of 0's, '1' and '2's in ternary logic case), logic operations do not need to consider the correlation caused by reconvergent fanouts. Moreover, signal dependencies are inherently maintained in the distribution patterns of the random bit streams. A detailed analysis and discussion of these features can be found in [9, 20].

So, the evaluation procedure using the proposed SCM approach for a ternary circuit can be described as follows:

1. Compute the error rate for the CNTFET and execute the transistor level analysis for every type of ternary logic gates.

2. Construct the stochastic computational model by replacing every logic gate with a ternary multiplexer.
3. Generate the initial random bit streams by encoding the output distributions for every input vector. These random bit streams are used as inputs of the multiplexers.
4. Propagate the bit streams from the primary inputs to the outputs and obtain a random bit stream for each output.
5. Decode the signal probability and calculate the reliability of each output from the obtained random bit stream.

An example of a MVL circuit is analyzed in more detail next. As discussed in [48], a ternary decoder is required for designing arithmetic circuits such as ternary adders and multipliers. The ternary decoder is a one-input and three-output combinational circuit that generates unary functions for the input *X*; the function of the ternary decoder is described by:

$$\mathrm{X_k} = \begin{cases} 2, & \text{if } \mathrm{X} = \mathrm{k} \\ 0, & \text{if } \mathrm{X} \neq \mathrm{k} \end{cases} \tag{4.40}$$

where k has a logic value of 0, 1, or 2.

Using the proposed pseudo-complementary ternary gates (STI, NTI, PTI and NMIN) as discussed in Section III, a decoder is designed; this circuit is functionally equivalent as the design proposed in [48].

Based on Section 4.4, the SCM for the decoder in Figure 4.13 is constructed by replacing each gate with a multiplexer, as shown in Figure 4.14. Simulation was performed on a PC with an Intel(R) Core(TM) i5-2430M CPU @ 2.40GHz and a 4.00 GB RAM. Table 4.12 shows the simulation results.

As shown in Table 4.12, even if the number of gates in the decoder is small, the joint reliability is rather low; this is due to the low reliability of each gate. This suggests that to enhance reliability, the fabrication process of CNTFET gates should be improved and fault-tolerant techniques must be applied. Also, the time for SCM simulation is very small.

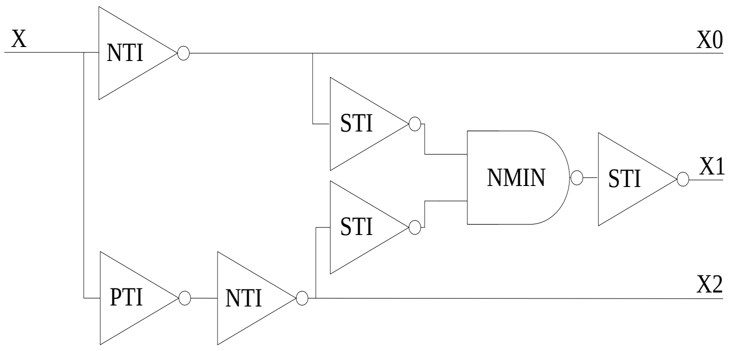


Figure 4.13 Schematic diagram of the ternary decoder.

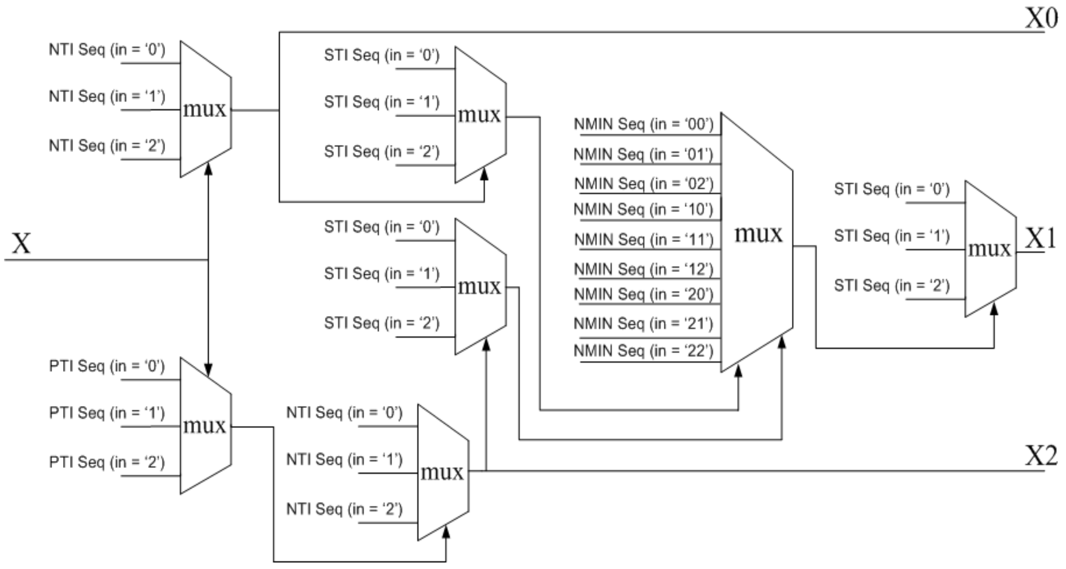


Figure 4.14 Reliability evaluation using stochastic computational models for the ternary decoder.

Table 4.12. Simulation results of the decoder using SCMs (sequence length = 10000 bits)

Input	Joint reliability	Simulation Time (s)
0	0.6737	0.068443
1	0.5346	0.071037
2	0.6905	0.070564
random	0.6407	0.063327

4.5. Summary

Stochastic Computational Models based on non-Bernoulli sequences are mathematically compared with traditional stochastic computing methods at the beginning of this chapter, which shows that the newly proposed SCM not only costs less time but also results in higher accuracy.

This chapter has also presented the design and reliability evaluation of multiple valued logic (MVL) gates. A pseudo-complementary implementation of MVL based on CNTFETs has been proposed; it replaces the resistors used in [49] with p-type CNTFETs (with their gates connected to ground) and utilizes threshold voltage operation by adjusting the chirality and the number of CNTs in each CNTFET. Therefore, this approach (referred to as *pseudo-complementary*) exploits the similarities in threshold voltage control in the p- and n-types while ensuring a correct MVL design for both ternary and quaternary logic gates. Simulation results using HSPICE have confirmed the validity of the proposed pseudo-complementary approach.

A transistor-level analysis has further been proposed to accurately estimate the error rate of the MVL gates. This analytical approach is based on the structure of the gate so it is significantly different from previous approaches that assume the same error rate for all logic gates. A general stochastic computational model for reliability evaluation of MVL gates has also been proposed. The initial application of this approach to MVL circuits has been briefly presented by simulating a ternary decoder.

CHAPTER 5

Stochastic Boolean Networks: An Efficient Approach to Modeling Gene Regulatory Networks*

Since a probabilistic Boolean network (PBN) considers molecular and genetic noise, it is regarded as one of the most important models of a gene regulatory network. The study of PBNs can not only provide more insights into the understanding of the dynamics of GRNs, but can also lead to advances in developing genetic therapeutic methods. However, the applications of PBNs are hindered by the huge computational complexities.

We have presented a novel implementation of PBNs based on the notions of stochastic logic and stochastic computation. This stochastic implementation of a PBN is referred to as a stochastic Boolean network (SBN). An SBN provides an accurate and efficient simulation of a PBN without and with random gene perturbation. Simulation results have shown that an SBN can not only recover biologically-proven regulatory behaviors but can also predict the network dynamics when the genes are under perturbation. The algorithms and methods presented in this chapter have been implemented in Matlab packages and can be applied in modeling of a general GRN.

**A version of this chapter has been accepted for publication in [22].*

This chapter is organized as follows. Section 5.1 reviews the background. Section 5.2 presents the SBN method. Applications and examples are also provided in this section. Section 5.3 presents simulation results and discussions. Section 5.4 summarizes this chapter.

5.1. Background

Biological systems are inherently noisy, yet robust in the presence of noise. The function and malfunction of a system are regulated through the interactions among genes, proteins and other molecules in the cellular network. For instance, the tumour suppressor gene p53 controls cell growth and plays an important role in preventing the development and progression of tumour cells [53-56]. Therefore, it has been of great interest to understand the regulatory mechanisms of genes, and various computational models have been developed for a better understanding of gene regulatory networks (GRNs) [57].

These models can be classified into three broad categories: logical models, continuous models and stochastic models at the single-molecule level [58]. Boolean networks (BNs) are logical models that utilize discrete state levels and usually assume synchronous and discrete time steps in the evolution of a network [59], whereas continuous models, such as those using linear or ordinary differential equations [60], employ real-valued state variables over a continuous timescale. Although continuous models are in principle more accurate and may describe the dynamics of a system in more detail, they require extensive high-quality experimental data that may not always be available to modelers. As a single-molecule level model, Gillespie's stochastic simulation algorithm (SSA) [61, 62] is based on the chemical master equation; it describes the interactions among single molecules and accounts for noise and stochasticity in the regulation of a genetic network. While the SSA provides the most accurate description of the regulatory behavior, it requires a large number of parameters and a detailed understanding of the regulatory mechanism. Despite the development of approximate SSAs that trade off accuracy for efficiency [63, 64], algorithms using single-molecule level models are generally slow to run, especially in the modeling of large genetic networks.

Albeit simplistic, BNs have been shown to be efficient in the modeling of GRNs by taking advantages of low complexity and a minimum requirement on the quality (and quantity) of experimental data [65]. To account for the intrinsic noise in genetic and molecular interactions, probabilistic Boolean networks (PBNs) have been developed as a generalization of BNs [66-68]. In a PBN, the inherent stochastic nature of molecular and genetic interactions dictates that the next state of target genes is predicted by several BNs with various probabilities. The evolution of such a system is thus a Markov chain and the state transitions can be described by a transition probability matrix. A steady-state analysis further tells whether a PBN will evolve into a stable target state in the presence of random gene perturbations, thereby providing valuable information for developing intervention-based therapeutic approaches [69 – 73].

The computation of the steady-state distribution of a PBN, however, presents a challenge. In a PBN with n genes and N Boolean networks, the complexity to compute the state transition matrix is $\mathrm{O}(\mathrm{nN}2^{2\mathrm{n}})$ [67] and it is more difficult to compute the steady-state distribution. This complexity is reduced to $\mathrm{O}(\mathrm{nN}2^{\mathrm{n}})$ for a sparse state transition matrix [74] and can further be reduced (to the same order, but with a smaller N) by ignoring the Boolean networks with probabilities below certain threshold [75]. Methodologies have also been developed by eliminating genes [76] and using optimal control policies [77] to reduce computational complexity. State reduction techniques have been used for network intervention [78] and to reduce the model complexity of context-sensitive PBNs [79]. Nevertheless, it remains a difficult problem to reduce the computational complexity of a PBN without compromising the accuracy of an evaluation.

Although synchronicity is usually assumed in the state transitions of PBNs, asynchronous PBNs have been considered by accounting for different updating periods of genes in the constituent BNs. Asynchronous PBNs are potentially more accurate in describing the regulatory behaviour of genetic networks and may provide a better vehicle for investigating intervention strategies that lead to optimal therapeutic methodologies [80, 81].

As an application of BNs, logic circuits have been used to simulate genetic networks [82]. Recently, circuit diagnosis techniques have been utilized to identify the most vulnerable molecules in cellular networks [83]. Synchronous simulation of Boolean networks has been proposed for the analysis of biological regulatory networks [84]. An unreliable logic circuit usually behaves probabilistically and thus becomes an instance of PBNs. Initially proposed for reliable circuit design [16, 45], stochastic computation has been demonstrated in several physical and biological applications [85, 86].

In this chapter, a stochastic computational model is presented for an efficient representation and simulation of PBNs; this implementation of a PBN is referred to as a stochastic Boolean network (SBN). It is shown that in an SBN, the complexity to compute the state transition matrix is $O(nL2^n)$, where L is a factor related to the minimum sequence length required for obtaining an evaluation accuracy and is significantly smaller than N in a network with a large number of genes. By using a time-frame expanded structure of the SBN, the steady-state distribution can be efficiently computed. Asynchronous PBNs can also be modeled by SBNs for studying the state dynamics of GRNs. With the recent development of BN models [65, 87, 88], the SBN technique is potentially useful in the modeling of large genetic networks. The accuracy and efficiency of the proposed techniques are demonstrated through extensive simulation results. Albeit proposed on the formalism of PBNs, the SBN framework is potentially applicable in improving the simulation efficiency of continuous models (using linear or ordinary differential equations) and single-molecule level models such as those based on SSAs. These aspects are further discussed in the Results and Discussion section.

5.2. Methods

5.2.1 Probabilistic Boolean Networks (PBNs)

In a PBN, genes are represented by a set of binary-valued nodes and the state transitions of genes are described by a list of Boolean functions. Following [67], a PBN is defined by *G (V, F)*, where *V* = $\{X_1, X_2, \dots X_n\}$ is a set of binary-valued nodes, $F = (F_1, F_2, \dots F_n)$ is a list of sets of Boolean functions: $F_i = \left\{f_1^{(i)}, f_2^{(i)}, \dots f_{l(i)}^{(i)}\right\}$ and *l(i)* is the number of possible functions for gene *i*. Each node X_i represents the state of gene i, denoted by x_i and $x_i = 1$ (or 0) indicates that gene *i* is (or not) expressed. The set F_i contains the rules that determine the next state of gene i. Each $f_{j(i)}^{(i)}: \{0,1\}^{\mathrm{n}} \to \{0,1\}$ for $1 \le j(i) \le l(i)$ is a mapping or a Boolean function determining the value of gene *i*.

Due to the noise in genetic networks, the functions in a PBN occur with certain probabilities. The next state of gene *i* is determined by all the *l(i)* functions in F_i, i.e., by $f_1^{(i)}$, $f_2^{(i)} \dots f_{l(i)}^{(i)}$ with probabilities $c_1^{(i)}$, $c_2^{(i)} \dots c_{l(i)}^{(i)}$. Thus, the next state of genes is totally determined by the possible functions and the present state of genes. This indicates that a PBN is modeled as a Markov chain. The fact that all genes are supposed to be updated synchronously also suggests a finite state machine (FSM) model for a PBN.

A PBN is independent if the functions from F_i are independent. This means that the selection of Boolean functions for gene *i* has no influence on the selection of Boolean functions for gene j ($i \ne j$) [89]. As a first study, the discussions in this chapter are limited to independent PBNs. For an independent PBN of *n* genes, there are a total number of $N = \prod_{\mathrm{i}=1}^{\mathrm{n}} l(i)$ possible BNs, each of which is a possible realization of the genetic network.

For the *j*th BN ($1 \le j \le N$), let $f_j = \left[f_{j(1)}^{(1)}, f_{j(2)}^{(2)} \dots f_{j(n)}^{(n)}\right]$, where $1 \le j(i) \le l(i)$ and $i = 1, 2 \dots n$. The probability that the *j*th BN is selected is:

$$P_j = \prod_{i=1}^{n} c_{j(i)}^{(i)}, \tag{5.1}$$

where $c_{j(i)}^{(i)}$ is the probability that the Boolean function *j(i)* is selected for gene *i*. By a different selection of the BNs during a state transition, the genes can reach a different state from their present state. This property of a PBN can be described by a state transition matrix as:

$$\boldsymbol{A} = \begin{bmatrix} p(0|0) & p(1|0) & \dots\dots\, p(2^{\mathrm{n}}-1|0) \\ p(0|1) & p(1|1) & \dots\dots\, p(2^{\mathrm{n}}-1|1) \\ \dots\dots & \dots\dots & \dots\dots \\ \dots\dots & \dots\dots & \dots\dots \\ p(0|\,2^{\mathrm{n}}-1) & p(1|\,2^{\mathrm{n}}-1) & \dots\dots\, p(2^{\mathrm{n}}-1|\,2^{\mathrm{n}}-1) \end{bmatrix}, \tag{5.2}$$

where each entry is a conditional (transition) probability that the genes transfer from a given present state into a next state. Since each BN results in a unique next state, the matrix $\boldsymbol{A}$ can be obtained by $\boldsymbol{A} = \sum_{j=1}^{N} P_j \boldsymbol{A}_{\mathbf{j}}$, where P_j is the probability that the *j*th BN occurs and $\boldsymbol{A}_{\mathbf{j}}$ is the state transition matrix due to the *j*th BN. This way of computing $\boldsymbol{A}$ results in a complexity of $O(nN2^{2n})$ [66]. Random gene perturbation, which can occur in an open genome system, is caused by random inputs from outside under external stimuli [69]. By a perturbation, a gene flips its state from 1 to 0 or vice versa. Since a PBN with perturbation is an aperiodic and irreducible homogeneous Markov chain [67], any PBN with perturbation will reach a steady state in a long run. A variant of the state transition matrix $\boldsymbol{A}$ can be used to model the effect of perturbation; however the analysis of its steady-state distribution is complex [69].

Usually, synchronicity is assumed in the state transitions of PBNs. However, a gene-level asynchronous model considers different updating periods of genes in the constituent BNs. In a deterministic-asynchronous Boolean network (DA-BN), a gene is assumed to have a fixed updating period [68]. A PBN that uses DA-BNs as constituent networks is defined as a deterministic-asynchronous probabilistic Boolean network (DA-PBN). More rigorously, a DA-PBN of *n* genes consists of a set $V = \{X_i\}_{i=1}^{n}$, where X_i represents the expression level of the ith gene, denoted by x_i and $x_i \in \{0, 1\}$ [68]. In a DA-PBN, a gene updates its state by its updating period using the DA-BN involved. At time t, a binary variable $\theta_i(\mathrm{t})$ can be used to indicate whether the state of gene i is updated or not, by a value of 1 or 0 respectively. The next state of gene i, $x_i(t+1)$, is then determined by:

$$x_i(t+1) = \begin{cases} f_{j(i)}^{(i)}\big(x_1(t), \ldots, x_n(t)\big) \text{ with probability } c_{j(i)}^{(i)}, & \text{if } \theta_i(\mathrm{t}+1) = 1 \\ x_i(t), & \text{otherwise} \end{cases} \quad (5.3)$$

where $f_{j(i)}^{(i)}$ is a function in the DA-BN for gene i, selected with probability $c_{j(i)}^{(i)}$ $(1 \leq j(i) \leq l(i))$.

5.2.2 Stochastic Boolean Networks (SBNs)

5.2.2.1. An SBN without perturbation

In stochastic computation, real numbers are represented by random binary bit streams and information is carried in the statistics of the binary streams [44]. A stochastic processing element is typically implemented by a logic gate. Stochastic logic processes information encoded in the random binary bit streams. Probability is represented by a proportional number of bits, usually the mean number of 1’s in a bit sequence. Given independent inputs, for example, an inverter computes the complement of a probability while the multiplication of probabilities is implemented by an AND gate. Hence, stochastic computation transforms Boolean logic operations into probabilistic computation in the real domain. Signal correlations can be efficiently handled in a stochastic network by the bit-wise dependencies encoded in the random binary streams, so making it an efficient approach to computing probabilities [20].

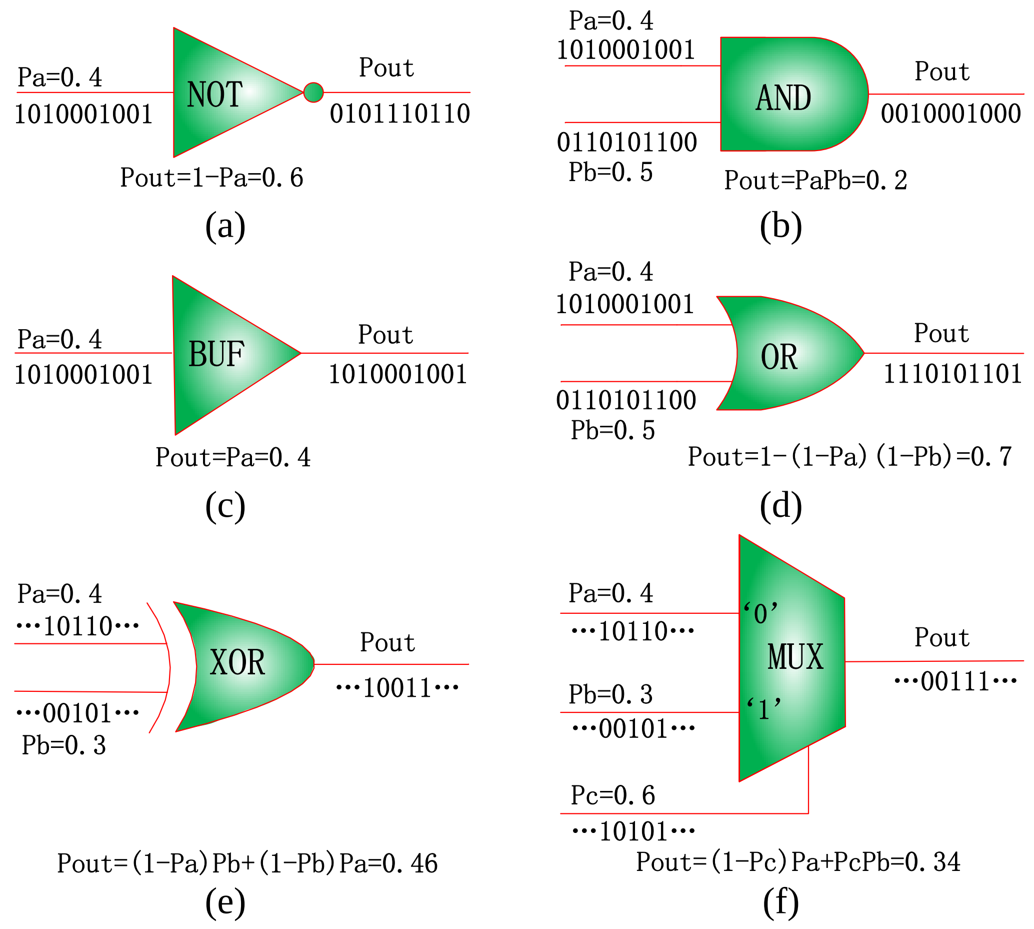


Figure 5.1. Stochastic logic. (a) a NOT gate, (b) an AND gate, (c) a Buffer, (d) an OR gate, (e) an XOR gate and (f) a Multiplexer.

Figure 5.1 shows an inverter (NOT), an AND, a buffer, an OR, an XOR gate and a multiplexer. While an XOR gate performs a controlled inversion, a multiplexer takes one of its inputs as output according to the values of the control bits. For the 2-to-1 multiplexer of Figure 5.1(f), for example, its output takes the value of its input 'a' or 'b' when the control bit 'c' is 0 or 1. Similarly, a stochastic multiplexer chooses one of its inputs as output according to the distributions of 0's and 1's and thus the probability of 0 and 1 encoded in the random sequences of the control bits. For a sequence length of 1000 bits, for example, an input probability of 0.4 indicates that approximately 400 1's are in the random sequence of the input 'a,' as shown in Figure 5.1(f). If the random input sequences are independent, the output of the multiplexer is expected to be $P_a(1 - P_c) + P_bP_c =$

0.34, which means that approximately 340 1's are expected in the output sequence. Note that this number is only approximate due to the stochastic fluctuations inherent in the representation of the random binary bit streams. This is an important feature in stochastic computation as probabilistic values are propagated rather than deterministic ones, which results in inevitable random fluctuations in the representation of probabilities. It has been shown, however, when non-Bernoulli sequences of random permutations of fixed numbers of 1's and 0's are used for representing initial probabilities, these fluctuations are significantly smaller than in a random sampling based simulation, which is equivalent to the case when Bernoulli sequences are used [20]. It is shown later in the Result section that these fluctuations are generally negligible when reasonably long random bit sequences are used.

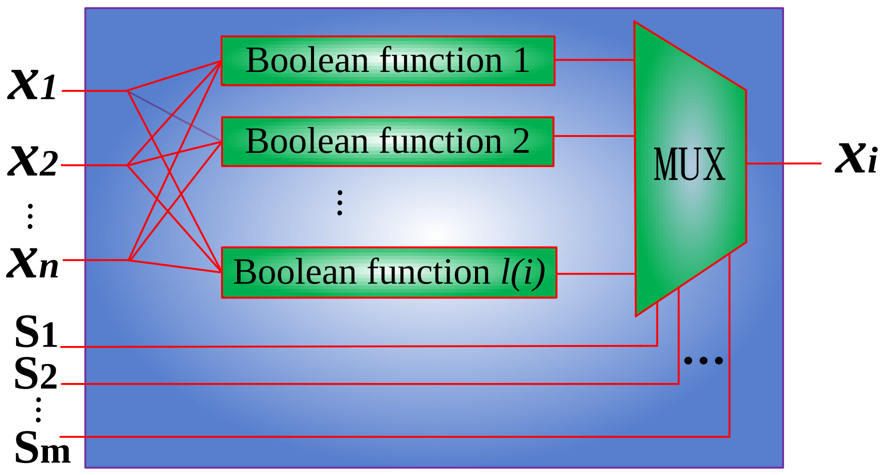


Figure 5.2. A stochastic Boolean network (SBN) without perturbation (for a single gene).

A general structure of the stochastic Boolean network (SBN) is defined as follows. As shown previously, the next state of genes in a PBN is updated by a set of Boolean functions according to certain probabilities. In an SBN, these probabilities are represented by random binary bit sequences and the selection of the Boolean functions is implemented by a stochastic multiplexer with properly generated control sequences. A general structure of an SBN for a single gene is shown in Figure 5.2.

Generally, if a total number of *l(i)* Boolean functions are needed to determine the next state of gene i, an *l(i)*-input multiplexer is used to simulate the selection of functions in the PBN for gene i. The number of control bits is given by m = $\lceil \log_2(l(i)) \rceil$. In fact, the number of possible Boolean functions for one gene is usually small - between 1 and 4 for 93% of genes [90, 91]. This indicates that one or two bits are usually sufficient to control a multiplexer in an SBN. By using the stochastic multiplexer with the control bit streams S_1~S_m, as shown in Figure 5.2, a function in the *j*th BN is selected with probability $c_{j(i)}^{(i)}$ for gene *i*. When all the genes are accounted for, therefore, an SBN accurately implements the probabilistic functions of a PBN, as dictated by (5.1).

5.2.2.2. An SBN with perturbation

While a switch of Boolean functions may indicate a structural change in the network, a random perturbation could cause a transient change of a gene's state under external stimuli. In a PBN with perturbation, a gene may change its value with a small probability *p* during each state transition.

Assume $\boldsymbol{x} = (x_1, x_2, \dots\ x_n)$ represents the current state of an *n*-gene network at time *t* and $\boldsymbol{\gamma}$ is the vector that indicates the effect of random perturbation, the next state $\boldsymbol{x}'$ is given by [69]:

$$\boldsymbol{x}' = \begin{cases} \boldsymbol{x} \oplus \boldsymbol{\gamma} & \text{with a probability of } 1 - (1-p)^{\boldsymbol{n}} \\ f_k(\boldsymbol{x}) & \text{with a probability of } (1-p)^{\boldsymbol{n}} \end{cases} \tag{5.4}$$

where $\oplus$ is the modulo 2 of additions and $f_k(\cdot)$ represents the function of the *k*th Boolean network at time *t*. The effect of perturbation to the state transition matrix can then be described by a matrix called the perturbation matrix [75]. The perturbation matrix is determined by the number of genes and the gene perturbation probability *p*. It is usually computed by a (complex) analytical approach.

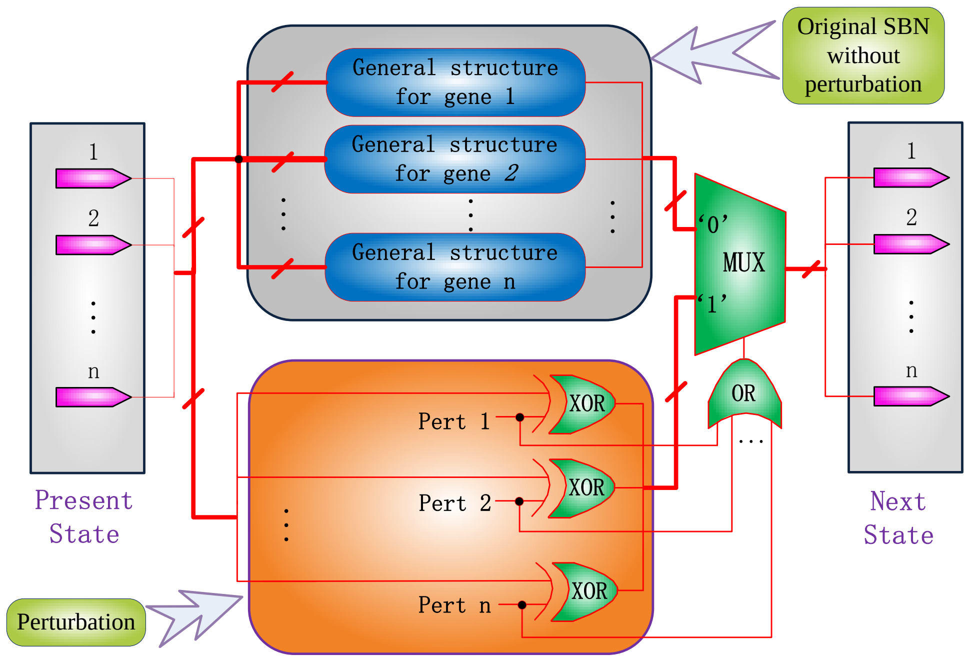

Original SBN without perturbation
General structure for gene 1
General structure for gene 2
General structure for gene n
1
2
n
Present State
Next State
'0'
MUX
'1'
OR
Pert 1
Pert 2
Pert n
XOR
XOR
XOR
Perturbation

Figure 5.3. An SBN with perturbation.

However, the effect of perturbation can be readily accounted for in an SBN. Figure 5.3 illustrates a general model of SBNs with perturbation. As perturbation introduces a probabilistic inversion to the state of a gene, XOR gates are used to implement the addition modulo 2 of the perturbation vector and the present state. The probability that either a Boolean function works or a perturbation works (given in (5.4)) is computed by a stochastic *n*-input OR gate. This probability is then encoded into the output sequence of the OR gate and used as the control sequence of a bus multiplexer. If the perturbation vectors ('Pert 1' … 'Pert n' in Figure 5.3) are all 0's, which means there is no perturbation, then the output sequence of the OR gate contains all 0's, which subsequently determines that the next state is given by the original SBN without perturbation; otherwise, the next state is determined by the perturbation probability encoded in the output sequence of the stochastic OR gate. Per the stochastic functions of XOR, OR and the multiplexer, the next state is given as the output of the SBN with perturbation, by:

$$\boldsymbol{x}' = (\boldsymbol{x} \oplus \boldsymbol{\gamma}) * (1-(1-p)^n) + f_k(\boldsymbol{x}) * (1-p)^n \,, \tag{5.5}$$

which is equivalent to (5.4). This indicates that a PBN with perturbation can be accurately implemented by an SBN with perturbation.

5.2.2.3. An SBN for asynchronous PBNs

In contrast to synchronous PBNs, each gene in an asynchronous PBN has a different period of updating time. Mathematically, this is described by (5.3) for the so-called deterministic-asynchronous probabilistic Boolean networks (DA-PBNs). In a DA-PBN, the state of each gene is independently updated according to its own updating period.

While the deterministic asynchronicity changes the temporal sequence of state transitions, it has no impact on the logic relationships among genes, so the Boolean functions are preserved for each gene in a DA-PBN. To model this asynchronicity, an SBN can be constructed by considering the timing information as follows:

(1) Construct the Boolean functions for each gene using the proposed SBN structure.

(2) Sort the genes by the updating period and record the sequence. For example, a sequence can be created as $G_t = \{g_{t\_1}, g_{t\_2}, \dots, g_{t\_n}\}$, where the updating periods of $g_{t\_1}, g_{t\_2}, \dots, g_{t\_n}$ are in an ascending order.

(3) Consider the first gene, i.e., the gene with the smallest updating period in G_t, denoted by g_{t_i}. Since the state of g_{t_i} will first be updated while the states of the other genes remain unchanged, the BNs at this stage consist of the Boolean functions of g_{t_i} and buffers for the other genes. A buffer is a logic element with a delayed input as its output. In this structure, a buffer is used to preserve the state of a gene that is not being updated.

(4) Delete g_{t_i} from G_t.

(5) Repeat steps (3) and (4) until G_t is empty.

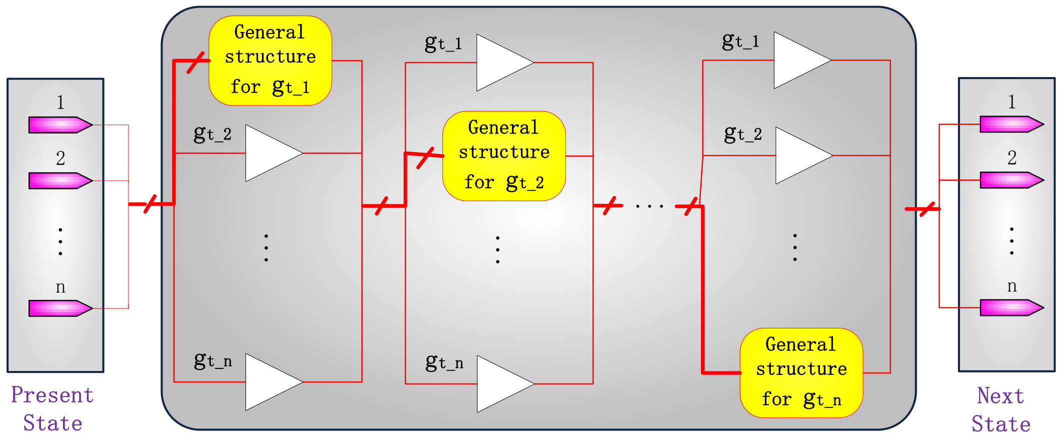

1
2
n
Present State
General structure for gt_1
gt_2
gt_n
gt_1
General structure for gt_2
gt_n
gt_1
gt_2
General structure for gt_n
1
2
n
Next State

Figure 5.4. An SBN for a deterministic asynchronous PBN.

An SBN for a DA-PBN is shown in Figure 5.4. Since the state transition of a fast-response gene may occur several times before a slow-response gene updates its state, the Boolean functions for a fast gene may appear in a number of times in the network of Figure 5.4.

5.2.3 Applications of SBNs

5.2.3.1. Computation of the State Transition Matrix

In an SBN, each input combination yields output sequences that contain information about the transition probabilities from this input state to an output state. Therefore, the statistics, i.e., the proportions of the number of each state encoded in the output sequences return the transition probabilities in a row in the state transition matrix. This row corresponds to the given input state and thus all the transition probabilities from this input can be generated in a single run. For a PBN with *n* genes, the SBN needs to be run for each of the 2^n input states and an $O(n)$ number of sequences need to be generated for the control signals of the multiplexers.

The accuracy in the computed state transition probabilities is determined by the sequence length of the random binary bit streams. In general, longer sequences are required in a larger network for achieving certain evaluation accuracy. To consider the overhead incurred in the use of a larger sequence length, a factor, L, is introduced and together with the other considerations, a complexity of $O(nL2^n)$ results for computing all the entries in the state transition matrix for a desired accuracy.

It has been shown that the required sequence length approximately increases linearly with the size of a combinational network [20]. In an SBN, the network size is typically on a polynomial order of the number of genes. This is in contrast with the number of BNs, N, which generally increases exponentially with the number of genes. As a result, the complexity of using an SBN to compute the transition matrix, i.e., $O(nL2^n)$, is significantly smaller than the analytical result of $O(nN2^n)$, especially for a network with a large number of genes. This is demonstrated by simulations using several measures to determine the minimum sequence length required for certain accuracy.

The procedure of computing the state transition matrix using an SBN is summarized as follows:

(1) Construct an SBN by inserting a multiplexer for each gene in a PBN.

(2) For each input state, generate initial random binary streams encoding the control signal probabilities for each multiplexer.

(3) Propagate the binary streams from the present state (inputs) to the next state (outputs) and obtain a random bit sequence for each output.

(4) Obtain the statistics, i.e., the proportions of the number of each state encoded in the output sequences as the transition probabilities for this input state.

(5) Repeat steps (2), (3) and (4) for all 2^n input states to compute all the entries in the state transition matrix.

For an SBN with perturbation, the state transition matrix can be similarly computed using the procedure outlined above with an exception in the construction of the SBN (Step 1).

5.2.3.2. Estimation of the Steady-State Distribution

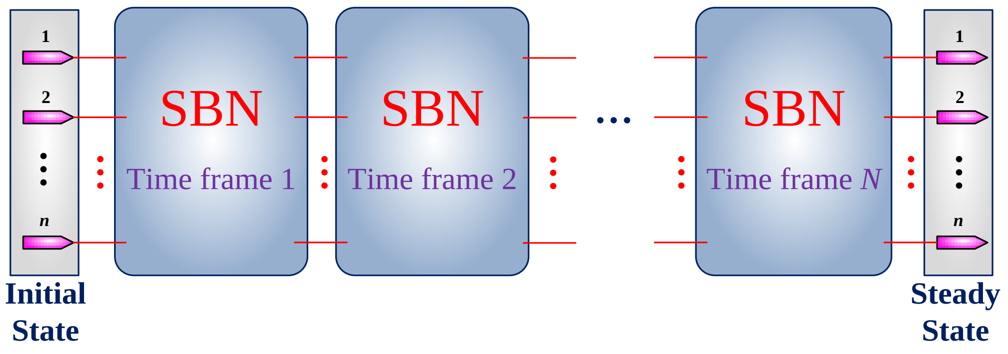


Figure 5.5. A time-frame extended SBN.

Given the size of the state transition matrix of a PBN, the analysis of the steady-state distribution is challenging for using both analytical and simulative approaches. The Markovian nature of a PBN makes its analysis similar to that of a finite state machine (FSM). An FSM is equivalent to a sequential circuit implementation. By a time-frame expansion, a sequential circuit can be unrolled into a series of identical combinational modules connected in the spatial domain. Using a similar technique, the temporal operation of an SBN can be transformed into a spatial operation of identical SBNs connected in series. This is shown in Figure 5.5. This spatial extension of an SBN can be used for the steady-state analysis and the required iterations of the SBN are determined by the number of state transitions before reaching a steady state.

A steady-state analysis using a time-frame expanded SBN starts with an initial input state, generate the random bit sequences for the inputs and control bits to multiplexers, and then propagate the stochastic signals through the expanded SBN structure. This process is equivalent to an analytical procedure of multiplying the input probabilities with the powers of the state transition matrix. Finally, a small variance threshold is used to determine whether the system has reached a steady state. The steady-state distribution is then obtained from the output sequences at the end of the operation.

In the above process, the speed of convergence to a steady state depends on a number of factors, including the length of random bit sequence, the variance threshold value and the perturbation rate. In practice, a sequence length that is long enough to have a resolution of at least two magnitudes smaller than the threshold value is used to guarantee that the convergence is not dominated by stochastic fluctuations. It is shown later that the analysis using an extended SBN structure provides an alternative and efficient way of estimating the steady-state distribution of a PBN without resorting to the state transition matrix.

5.2.4 Example: The p53-Mdm2 Network

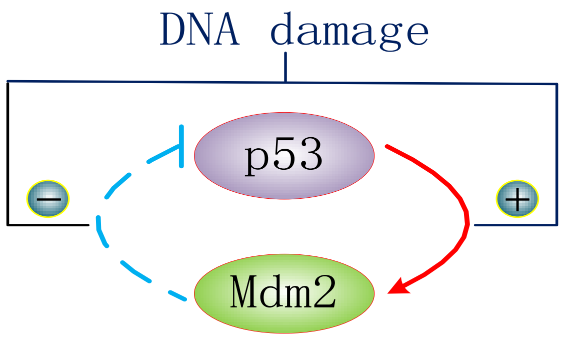


Figure 5.6. The p53-Mdm2 network (adapted from [55]).

In a p53 network, signaling pathways are triggered by DNA damage and external factors such as chemotherapeutic drugs and ultraviolet light. For instance, DNA double strand breaks (DSBs) activate pathways that involve the p53 and Mdm2 genes (Figure 5.6) [55, 56]. In response to the DSBs, the ATM kinase is first stimulated and the Chk2 is then stimulated by ATM. These activated kinases subsequently induce an increase in the concentration level of p53 and a decrease in the interactions between p53 and Mdm2. The increase in the p53 protein level and its transcription activity promote the expression of the Mdm2 gene, which in turn proceeds to trigger the degradation and destruction of p53. This prior knowledge enables us to come up with the transition rules for the p53-Mdm2 interactions, as shown in Table 5.1. Based on these rules, an independent PBN of the two genes p53 and Mdm2 can be established: V = (X_1, X_2) with the function classes $F_1 = \{f_1^{(1)}, f_2^{(1)}, f_3^{(1)}, f_4^{(1)}\}$ and $F_2 = \{f_1^{(2)}, f_2^{(2)}, f_3^{(2)}, f_4^{(2)}\}$. The state transitions of this PBN are given in the truth table of Table 5.2.

Table 5.1. State transition probabilities of the p53-Mdm2 network.

Present State	Next State Probability			
p53, Mdm2 (or, x_1x_2)	p53		Mdm2	
	0	1	0	1
00	0.01	0.99	0.99	0.01
01	0.1	0.9	0.9	0.1
10	0.9	0.1	0.1	0.9
11	0.5	0.5	0.5	0.5

Table 5.2. Truth table of the PBN for the p53-Mdm2 network.

x_1x_2	$f_1^{(1)}$	$f_2^{(1)}$	$f_3^{(1)}$	$f_4^{(1)}$	$f_1^{(2)}$	$f_2^{(2)}$	$f_3^{(2)}$	$f_4^{(2)}$
00	1	1	1	0	0	0	0	1
01	1	1	0	0	0	0	1	1
10	0	0	1	1	1	1	0	0
11	0	1	1	1	1	0	0	0
$c_j^{(i)}$	0.5	0.4	0.09	0.01	0.5	0.4	0.09	0.01

In Table 5.2, the leftmost column indicates the present state of the genes p53 and Mdm2. The internal entries in the table indicate whether a function will result in a logical 1 or 0 at the next state of each gene. The row on the bottom shows the probability of each transition by a function. Given an initial state of '01,' for example, the next state of the genes can be '00' with a probability of $(0.09 + 0.01) * (0.5 + 0.4) = 0.09$, '01' with a probability of $(0.09 + 0.01) *$

$(0.09 + 0.01) = 0.01$, '10' with a probability of $(0.5 + 0.4) * (0.5 + 0.4) = 0.81$ or '11' with a probability of $(0.5 + 0.4) * (0.09 + 0.01) = 0.09$. A PBN is determined by the truth table of Table 5.2 and its state transition matrix can be computed as:

$$\boldsymbol{A_{PBN}} = \begin{bmatrix} 0.0099 & 0.0001 & 0.9801 & 0.0099 \\ 0.0900 & 0.0100 & 0.8100 & 0.0900 \\ 0.0900 & 0.8100 & 0.0100 & 0.0900 \\ 0.2500 & 0.2500 & 0.2500 & 0.2500 \end{bmatrix}. \tag{5.6}$$

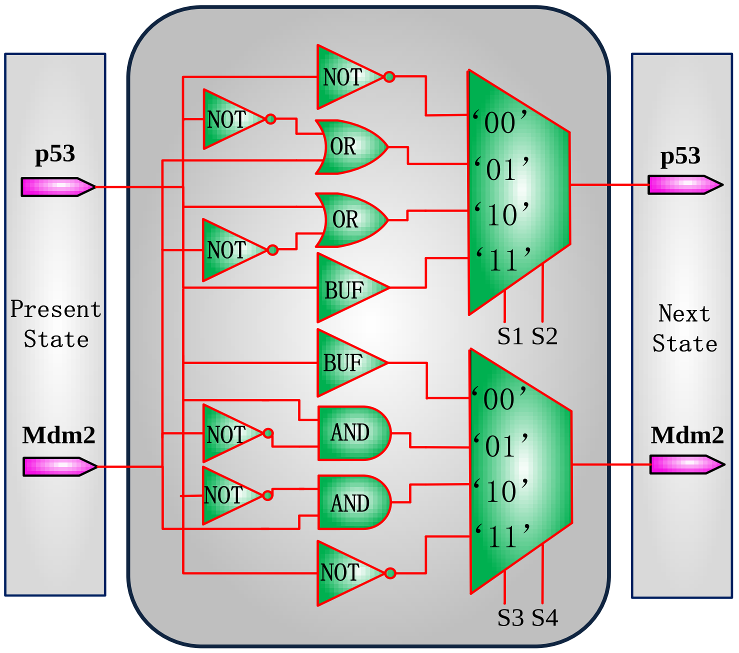


Figure 5.7. An SBN for the p53-Mdm2 network (without perturbation).

For this PBN, an SBN can be constructed using stochastic multiplexers and random binary bit streams as information carriers, as shown in Figure 5.7. As discussed previously, the control binary sequences determine the probability that each Boolean network is selected. For example, as the Boolean functions for the p53 gene occur with probabilities 0.5, 0.4, 0.09 and 0.01, the binary bit sequences for the control vectors 'S1S2' to the multiplexer are generated with a probability of 0.5 to be '00,' a probability of 0.4 to be '01', a probability of 0.09 to be '10' and a probability of 0.01 to be '11.' Then the output bit sequences are read out and decoded into (transition) probabilities. With a sequence length of 10000 bits, the state transition matrix is obtained as follows:

$$\boldsymbol{A}_{\boldsymbol{SBN}} = \begin{bmatrix} 0.0097 & 0.0003 & 0.9803 & 0.0097 \\ 0.0899 & 0.0101 & 0.8101 & 0.0899 \\ 0.0904 & 0.8096 & 0.0096 & 0.0904 \\ 0.2511 & 0.2489 & 0.2489 & 0.2511 \end{bmatrix}. \qquad (5.7)$$

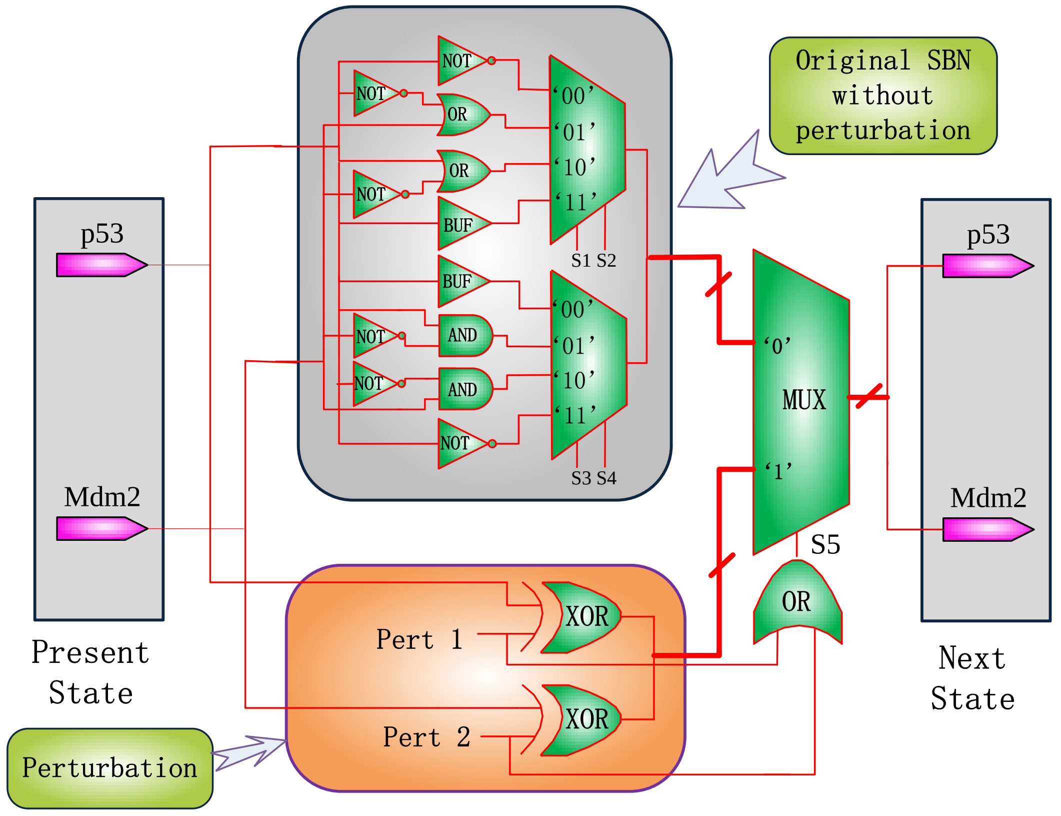


Figure 5.8. An SBN for the p53-Mdm2 network (with perturbation).

The difference between (5.6) and (5.7) can be evaluated using the following norms: $||\boldsymbol{A}||_1$ and $||\boldsymbol{A}||_\infty$, which specify the maximum absolute value of the summed differences of columns and rows of the two matrices respectively, and $||\boldsymbol{A}||_2$, which is a measure on the average difference of all the entries in these matrices. For (5.6) and (5.7), we obtain $||\boldsymbol{A}||_1 = 0.0018$, $||\boldsymbol{A}||_2 = 0.0024$ and $||\boldsymbol{A}||_\infty = 0.0044$, which indicate that the SBN structure accurately computes the state transition matrix of the PBN.

With random gene perturbation, an SBN with perturbation can be constructed, as shown in Figure 5.8. If the stochastic OR outputs a ‘1’ (indicated by S5 in Figure 5.8), which means that at least one of the p53 and Mdm2 are perturbed, the multiplexer is then switched to the perturbation network. If the output of the OR is 0, the multiplexer is switched to the original SBN and the network works as the one in Figure 5.7 without perturbation.

A similar procedure can be used to compute the state transition matrix of the SBN with perturbation – the result is shown in (5.8) for a perturbation probability of 0.01:

$$\widetilde{\boldsymbol{A}}_{\boldsymbol{SBN}} = \begin{bmatrix} 0.0097 & 0.0100 & 0.9705 & 0.0098 \\ 0.0975 & 0.0106 & 0.7946 & 0.0973 \\ 0.0998 & 0.7921 & 0.0082 & 0.0999 \\ 0.2444 & 0.2565 & 0.2551 & 0.2440 \end{bmatrix}. \tag{5.8}$$

Compared to the analytical result by a method based on (5.4):

$$\widetilde{\boldsymbol{A}}_{\boldsymbol{PBN}} = \begin{bmatrix} 0.0097 & 0.0100 & 0.9705 & 0.0098 \\ 0.0981 & 0.0098 & 0.7940 & 0.0981 \\ 0.0981 & 0.7940 & 0.0098 & 0.0981 \\ 0.2451 & 0.2549 & 0.2549 & 0.2450 \end{bmatrix}, \tag{5.9}$$

the differences between (5.8) and (5.9) are revealed in the measures of $||\boldsymbol{A}||_1 = 0.0032$, $||\boldsymbol{A}||_2 = 0.0030$ and $||\boldsymbol{A}||_\infty = 0.0042$. These show that the proposed approach using an SBN can accurately and efficiently compute the state transition matrix. The differences in these results come from the stochastic fluctuation, which is an intrinsic property of stochastic computation. More simulation results are presented in the Results and Discussion section, which show that the fluctuations are generally small. A steady state analysis using (5.8) further confirms the p53-Mdm2 oscillatory dynamics observed in experiments.

An SBN for an asynchronous p53-Mdm2 network can also be constructed, as in Figure 5.4 and following the aforementioned procedure. Due to space limitations, however, this is not further discussed and will be pursued in future work.

5.3. Results and Discussion

5.3.1 Simulations with Randomly Generated Networks

Table 5.3. Errors in the state transition matrices obtained using SBNs without perturbation, compared to the results by using the analytical approach in [74].

Number of genes (n)		2	3	4	5	6
Error	Length (bits)					
$\text{Error}_{\|\|\cdot\|\|_1}$	1000	0.0070	0.0330	0.0420	0.0477	0.0649
	10000	0.0027	0.0052	0.0105	0.0179	0.0186
$\text{Error}_{\|\|\cdot\|\|_2}$	1000	0.0100	0.0314	0.0408	0.0287	0.0405
	10000	0.0038	0.0047	0.0102	0.0109	0.0099
$\text{Error}_{\|\|\cdot\|\|_\infty}$	1000	0.0160	0.0640	0.0908	0.0735	0.1293
	10000	0.0056	0.0096	0.0248	0.0303	0.0248

The state transition matrices of several randomly generated PBNs have been computed using the proposed SBN structure. The Boolean functions of each network are generated for a given number of genes (n) and a total number of BNs (N). The simulation is run on a PC with an Intel Core i3-2100 CPU (@3.10 GHz) and 6G memory. The results for using sequence lengths of 10000 and 1000 bits are first compared to those obtained using an analytical approach, as shown in Table 5.3. While a larger sequence length of 10000 bits produces results with a higher precision, a sequence length of 1000 bits also provides highly accurate results and is therefore used next in the evaluation of the computed state transition matrices for larger networks.

The run time and accuracy for the SBNs without and with perturbation are shown in Tables 5.4 and 5.5, respectively. It can be seen that the SBN approach requires a significantly shorter runtime than the analytical approach, especially in the evaluation of large networks.

Table 5.4. Run time and errors in the computation of state transition matrices (the original SBN sequence length = 1000 bits, no perturbation, n: the number of genes, and N: the number of BNs).

n	N	SBN (s)	Method [74] (s)	$\text{Error}_{\|\|\cdot\|\|_1}$	$\text{Error}_{\|\|\cdot\|\|_2}$	$\text{Error}_{\|\|\cdot\|\|_\infty}$
2	6	0.015169	0.009068	0.0080	0.0126	0.0240
3	8	0.020842	0.009696	0.0190	0.0186	0.0280
4	16	0.044102	0.028376	0.0310	0.0269	0.0610
5	32	0.073996	0.129098	0.0520	0.0278	0.0672
6	64	0.170900	0.569229	0.0676	0.0274	0.0780
7	128	0.356299	2.591382	0.1375	0.0402	0.0998
8	256	0.778123	8.748113	0.1897	0.0559	0.1411
9	512	1.736358	40.189629	0.2929	0.0609	0.2322
10	1024	4.174700	178.876554	0.3946	0.0699	0.2401
11	2048	9.685876	783.659986	0.5950	0.0765	0.3584
12	4096	22.673064	3473.738188	1.1312	0.1041	0.4799

Table 5.5. Run time and errors in the computation of state transition matrices (sequence length = 1000 bits, perturbation probability = 0.01, n: the number of genes, and N: the number of BNs).

n	N	SBN (s)	Method [74] (s)	$\text{Error}_{\|\|\cdot\|\|_1}$	$\text{Error}_{\|\|\cdot\|\|_2}$	$\text{Error}_{\|\|\cdot\|\|_\infty}$
2	6	0.062898	0.043643	0.0086	0.0096	0.0165
3	8	0.040697	0.017449	0.0200	0.0150	0.0228
4	16	0.065944	0.046525	0.0452	0.0310	0.0607
5	32	0.101961	0.135997	0.0507	0.0327	0.0688
6	64	0.206113	0.623719	0.0941	0.0397	0.0890
7	128	0.410313	2.739103	0.1262	0.0467	0.1131
8	256	0.905610	12.567407	0.1924	0.0587	0.1378
9	512	2.061321	58.454630	0.2746	0.0639	0.2376
10	1024	4.770391	254.769805	0.4458	0.0686	0.2642
11	2048	11.481596	973.364318	0.6339	0.0771	0.3311
12	4096	25.686332	3892.347395	1.0837	0.0970	0.4433

Table 5.6. Minimum sequence length and run time required in the computation of state transition matrices for given accuracies, measured by Norm 2 (no perturbation, n: the number of genes, and N: the number of BNs).

n	N	SBN (Norm 2 = 0.04)		SBN (Norm 2 = 0.02)		Method [74] (s)
		Sequence length	Time (s)	Sequence length	Time (s)	
2	6	100	0.005797	500	0.011028	0.009068
3	8	400	0.017755	800	0.018843	0.009696
4	16	500	0.022734	1000	0.044102	0.028376
5	32	800	0.057780	1500	0.108644	0.129098
6	64	1000	0.160139	2200	0.348080	0.569229
7	128	1200	0.412106	3500	1.140480	2.591382
8	256	1900	1.445489	5200	3.890074	8.748113
9	512	2400	4.054003	6500	10.800441	40.189629
10	1024	3200	12.861136	8000	32.625211	178.876554
11	2048	4000	37.448054	10000	92.367577	783.659986
12	4096	5000	109.651805	12000	266.546885	3473.738188

Table 5.7. Minimum sequence length and run time required in the computation of state transition matrices for given accuracies, measured by Norm 2 (perturbation probability = 0.01, n: the number of genes, and N: the number of BNs).

n	N	SBN (Norm 2 = 0.04)		SBN (Norm 2 = 0.02)		Method [74] (s)
		Sequence length	Time (s)	Sequence length	Time (s)	
2	6	100	0.007354	300	0.015849	0.043643
3	8	300	0.009444	800	0.022467	0.017449
4	16	600	0.034080	1200	0.051354	0.046525
5	32	700	0.066767	1500	0.128628	0.135997
6	64	1000	0.206113	2300	0.413640	0.623719
7	128	1300	0.514317	4000	1.542643	2.739103
8	256	2000	1.754765	5200	4.411751	12.567407
9	512	2800	5.505697	6500	12.855831	58.454630
10	1024	3500	15.903428	9000	40.870411	254.769805
11	2048	4200	44.044262	11000	115.636105	973.364318
12	4096	5300	130.875398	14000	355.611421	3892.347395

Table 5.8. Run time and errors in the computation of state transition matrices for SBN and the approximation method in [75] (no perturbation, n: the number of genes, and N: the number of BNs).

n	N	SBN (s) (Length = 10000 bits)	Method [75] (s) (lower bound= 10^{-4})	$\text{Error}_{\|\|\cdot\|\|_2}$ (SBN)			$\text{Error}_{\|\|\cdot\|\|_2}$ [75]		
				$\text{Error}_{\|\|\cdot\|\|_1}$	$\text{Error}_{\|\|\cdot\|\|_2}$	$\text{Error}_{\|\|\cdot\|\|_\infty}$	$\text{Error}_{\|\|\cdot\|\|_1}$	$\text{Error}_{\|\|\cdot\|\|_2}$	$\text{Error}_{\|\|\cdot\|\|_\infty}$
11	2048	92.367577	183.617225	0.2031	0.0268	0.1209	0.2416	0.0463	0.0221
12	4096	221.849183	1125.969347	0.3448	0.0301	0.1540	0.6387	0.0929	0.0386
13	8192	489.265478	4395.954714	0.4581	0.0552	0.2249	1.6583	0.1414	0.0874
14	16384	1063.892415	9415.812415	1.0152	0.0825	0.4287	2.1642	0.2283	0.1895

On the other hand, however, the error incurred due to stochastic fluctuations also increases with the size of the network under evaluation. Subsequently, therefore, a minimum accuracy requirement is given and the length of the stochastic sequences is increased for a larger network in order to meet this requirement. Tables 5.6 and 5.7 show the minimum sequence lengths and run time required for two different accuracy values, given by the aforementioned "norm 2" that measures the average difference of all the entries in two matrices. Finally, the efficiency of the SBN technique is compared to that of an approximate analytical approach [75] for several networks with more than 10 genes. The results are shown in Table 5.8.

As revealed in the tables, while an analytical approach is fast in computing the state transition matrices of small networks, it becomes cumbersome to use for larger networks. This is because an analytical approach is limited by the number of BNs (*N*), which generally increases exponentially with the number of genes in a PBN. In an SBN, however, all the state transition probabilities for each input state are encoded in the output sequences, so the computation of the state transition matrix is very efficient. Although a longer stochastic sequence length is required to meet an evaluation accuracy, the proposed SBN approach still outperforms an analytical approach for networks with a large number of genes and BNs. This is further shown in Figure 5.9, in which the runtime of the two techniques is compared for the same accuracy requirements. While both techniques require an exponential complexity (in the number of genes), the proposed SBN technique is more efficient as it is not directly limited by the number of BNs.

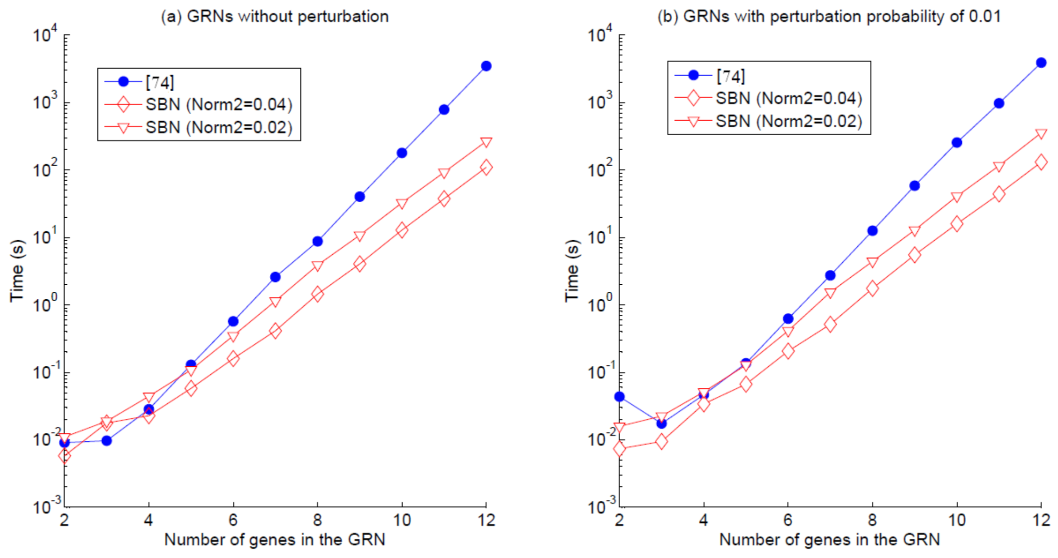


Figure 5.9. Comparisons of runtime of the SBN technique and the technique in [74].

The state transition matrix computed using an SBN can be used to obtain the steady state of a network. However, the size of the network that can be evaluated is restricted due to the exponential increase of the size of the matrix. As an alternative and efficient approach, the time-frame expansion technique can be used to evaluate much larger networks under perturbation. Recently, several BN models have been developed for GRNs with tens of genes [65, 87, 88]. Although the parameters for use in a PBN have not been obtained, the time frame expansion technique is well suited for simulating a network of such size, once the necessary parameters become available. Our experiments have shown that a 20-gene network with a perturbation rate of 0.01 can be evaluated in approximately 3.6 seconds using the time-frame expanded SBN technique. In Table 5.9, the runtime for simulating networks of 20 and 30 genes is shown for various accuracy requirements and perturbation rates. These results indicate that the time-frame expanded SBN technique is potentially useful in the analysis of large GRNs.

Table 5.9. Time consumption of the time frame expansion technique for randomly-generated networks.

Number of genes	Sequence length (bits)	Threshold value (Norm infinity)	Perturbation rate	SBN	
				No. of Periods before convergence	Time consumption (s)
20	100,000	0.001	0.0001	1723	1524.043060
	100,000	0.001	0.001	202	195.614374
	10,000	0.01	0.01	29	3.570570
30	1,000,000	0.0035	0.0001	71	721.123598
	1,000,000	0.01	0.0001	23	258.757348
	1,000,000	0.1	0.0001	19	199.884261

5.3.2 Experiments on a T-cell Time Series Dataset

A network inferred from a time series gene expression dataset [91] is further modelled using SBNs. The dataset was taken from an IL-2-stimulated immune response experiment using a murine T cell line called CTLL-2. Cells were collected at 12 different time points before IL-2 stimulation (0 h) and after IL-2 stimulation (15, 30 mins, 1, 2, 4, 6, 8, 10, 12, 16 and 24 h). The dataset was then normalized to the same expression level and clustered based on the similarities in the regulatory behaviour of the genes. This produced simplified networks of gene groups, referred to as meta-genes, instead of actual genes. This result has significantly reduced the complexity of the analysis and interpretation of the inferred networks. Finally, the dataset was discretized for the implementation of a Boolean network inference algorithm. This algorithm is discussed in detail next.

5.3.2.1. Inference of Boolean Dynamics of the GRN

PBNs have been inferred from steady-state data using the coefficient of determination [67] and from time series data to estimate the perturbation probabilities and switching probabilities between the constituent BNs [92]. Large amounts of data are usually required by these methods due to their computational complexity. In [91], the Boolean inference is based on the activation and inhibition functions of a target gene and its control genes. This is similar to the qualitative inference method used in [93], but it considers all possible networks rather than a single most likely one. While the number of possible inputs to a Boolean function is limited in this method, the restriction on the amount of data required to perform an inference is released. The number of possible networks is then counted and all networks are enumerated.

For the T-cell time series dataset, a total of 161,558 networks were discovered by the inference algorithm. The inference algorithm further explores the dynamics of the inferred networks. This is based on the fact that finite BNs are expected to exhibit a cyclic pattern of expression [59]. During this step, the steady states or attractors are computed to validate the inferred networks. It was found that 160,657 (99.4%) of these networks did not exhibit the fluctuations expected in the steady-state dynamics of the IL-2 stimulated T cell network. Therefore, these networks were discarded and 901 (0.6%) of the networks that produced biologically meaningful attractors were left for further analysis. The 901 networks were based on twelve meta-genes and yielded a consensus network as shown in Figure 5.10. The steady-state dynamics in the 901 networks consist of three time points (shown in Table 3 of [91]). It has also been shown that the computational complexity of this inference algorithm increases exponentially with the maximum number of inputs to a node [91]. Since the maximum input number is limited by the size of a network with a power low [94], this number is expected to be smaller than 5 for a network with less than 100 nodes.

The resulting network is not unique in that the occurrence of different Boolean functions results in different BNs. In Figure 5.10, the activation and inhibition relationships that occur in all 901 networks are indicated by solid arrows, while the relationships that occur in a fraction of the networks are indicated by dashed arrows. The value associated with a dashed arrow indicates the fraction of networks having that relationship. To infer a PBN, this fractional occurrence of a function is considered probabilistic and its associated value is taken as the occurrence probability of a Boolean function in the network. These probabilities are then utilized to obtain the switching probabilities between the constituent BNs in the PBN. Since a solid arrow indicates a relationship that exists in all 901 networks in Figure 5.10, this function is considered to occur with a probability of 1. The inferred PBN is shown in the truth tables, for which the Boolean functions are assumed to occur independently in a BN.

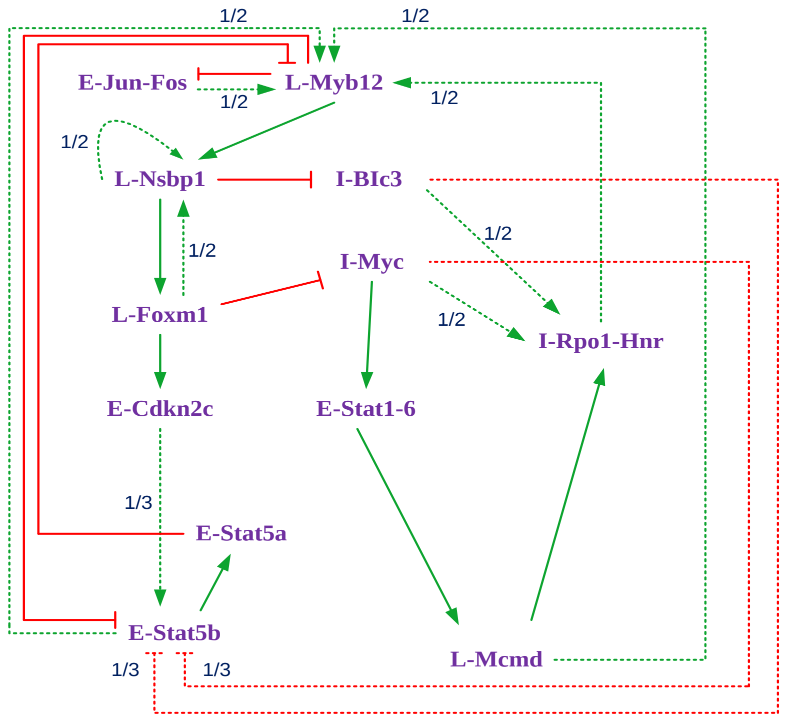


Figure 5.10. A T cell immune response network inferred from a time series gene expression dataset (adapted from [91]).

5.3.2.2. Modeling the network with SBN

To build an SBN for the inferred network of Figure 5.10, each of the 12 genes is assigned a number, as shown in Table 5.10. For these 12 genes, there are $2^{12} = 4096$ states, each of which is indexed by the state of each gene as follows:

$$k = \sum_{i=1}^{12} \mathrm{g}(i) * 2^{i-1} + 1, \tag{5.10}$$

where i is the gene index and $\mathrm{g}(i)$ is the state of gene i (i.e., 1 or 0).

Table 5.10. Code of the 12 genes in the T cell immune response network.

Gene	E-Jun-Fos	L-Nsbp1	L-Foxm1	I-BIc3	I-Myc	L-Myb12	E-Cdkn2c	E-Stat1-6	I-Rpol-hnr	E-stat5a	E-stat5b	L-Mcmd
Symbol	g(1)	g(2)	g(3)	g(4)	g(5)	g(6)	g(7)	g(8)	g(9)	g(10)	g(11)	g(12)

Since solid arrows in Figure 5.10 indicate regulatory interactions found in all 901 networks, they are considered to have a priority over other interactions, i.e., any other relationships are overruled by a solid-line interaction if they occur simultaneously. For the dashed arrows, the priority is determined according to the observations in the experiments. Take 'E-stat5b' for example; the solid arrow indicates that L-Myb12 inhibits E-stat5b in all the networks, so the activation of L-Myb12 overrules any other function applied on E-stat5b. When the state of E-stat5b is only affected by the dashed arrows, the activation by E-Cdkn2c is considered to take precedence over the inhibitions by I-BIc3 and I-Myc, as the upregulation of E-stat5b has been observed in the experiments.

An SBN is constructed for the genetic network of Figure 5.10, as shown in Figure 5.11. The construction is based on the following principles:

(1) An inhibiting signal is considered logical "low" while an activating signal is considered logical "high." Therefore, an inverter or a buffer is applied to represent an inhibition or an activation relationship between genes. For example, L-Myb12 inhibits E-Jun-Fos, so an inverter is used to simulate this relationship between $g_t(6)$ and $g_{t+1}(1)$. For the activation of L-Foxm1 by L-Nsbp1, a buffer is applied between $g_t(2)$ and $g_{t+1}(3)$.

(2) An OR gate is applied to model multiple activations while a NOR (inverted OR) gate is applied to model multiple inhibitions on the same gene. For example, L-Myb12 can be activated by any one of E-Jun-Fos, I-Rpol-Hnr, E-stat5b and L-

Mcmd, so in Figure 5.11, $g_t(1)$, $g_t(9)$, $g_t(11)$ and $g_t(12)$ are used as the four inputs to an OR gate. However, due to the inhibition of L-Myb12 by E-stat5a, an inverter is applied and its output is ANDed with the output of the 4-input OR gate to produce the output of $g_{t+1}(6)$. The use of the AND is dictated by the priority rule of the inhibition over the activation of L-Myb12, as explained as follows.

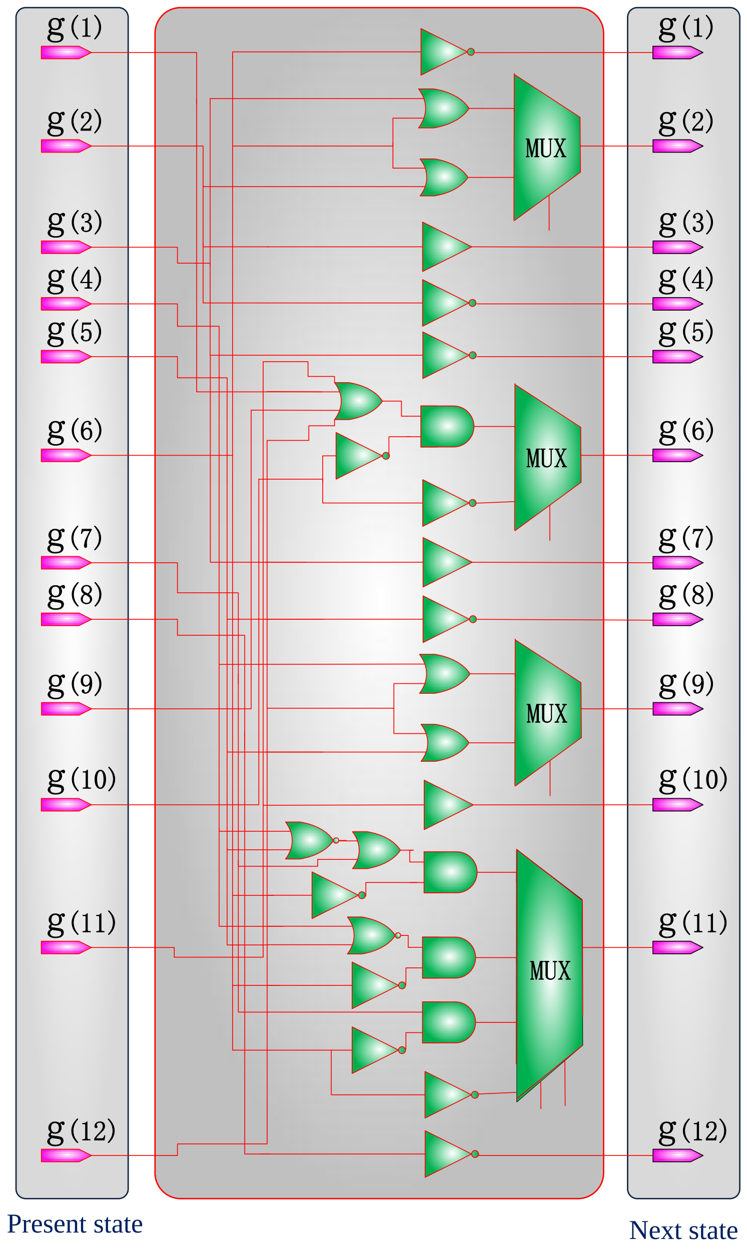


Figure 5.11. An SBN for the GRN in Figure 5.10.

(3) When an inhibition and activation occur on the same gene, the logic gate is determined by the priority of the two functions: an AND gate is applied if the inhibition has a higher priority, while an OR gate is used if the activation has a higher priority. For instance, an AND gate is used to model the relationship between the activation and inhibition of L-Myb12 in the example of (2), as shown in Figure 5.11.

(4) A solid arrow indicates a relationship that exists in all 901 networks and therefore is considered to occur with a probability of 1. The corresponding function then exists in every Boolean function that produces an input to a MUX. For example, E-stat5a inhibits L-Myb12 in all the networks, so inverters are present in both of the two Boolean functions that lead to $\mathrm{g}_{\mathrm{t+1}}(6)$.

5.3.2.3. Steady-state Evaluation

For this SBN, the state transition matrix $\boldsymbol{A_T}$ is of the size 4096 x 4096 and computed in about 70s. See additional file 5: The Matlab program that describes the structure of the SBN in Figure 5.11 and computes its state transition matrix (for both without and with perturbation).

Given an initial input, $\boldsymbol{I_0}$ = [0, 0… 0, 1, 0… 0], as indicated by the vector at T = *1h* in Table 5.3 of [91] that corresponds to the state 1730 (by (5.10)), the output response after *t* clock cycles can be computed by:

$$\mathrm{Output}(t) = I_0 * A_T^t. \tag{5.11}$$

A clock cycle here corresponds to the time interval between two discrete time points as a period of biological response. It has been shown that the network reaches a steady state consisting of three time points [91]. In our simulation, a periodic behaviour of state transitions has been observed after 20 clock cycles.

Table 5.11. Attractors found by the SBN approach, compared to the experimental results in [91].

Number of cycles	States with highest probabilities	Attractors found in [91]
28	1224	Attractor 1
	711	Attractor 3
	1768	Attractor 2
29	1768	Attractor 2
	1224	Attractor 1
	711	Attractor 3
30	711	Attractor 3
	1768	Attractor 2
	1224	Attractor 1

As shown in Table 5.11, the obtained stationary states perfectly match the three attractors found at the time points *t1*, *t2* and *t3* in [91], referred to as Attractors 1, 2 and 3 at states 1224, 1768 and 711.

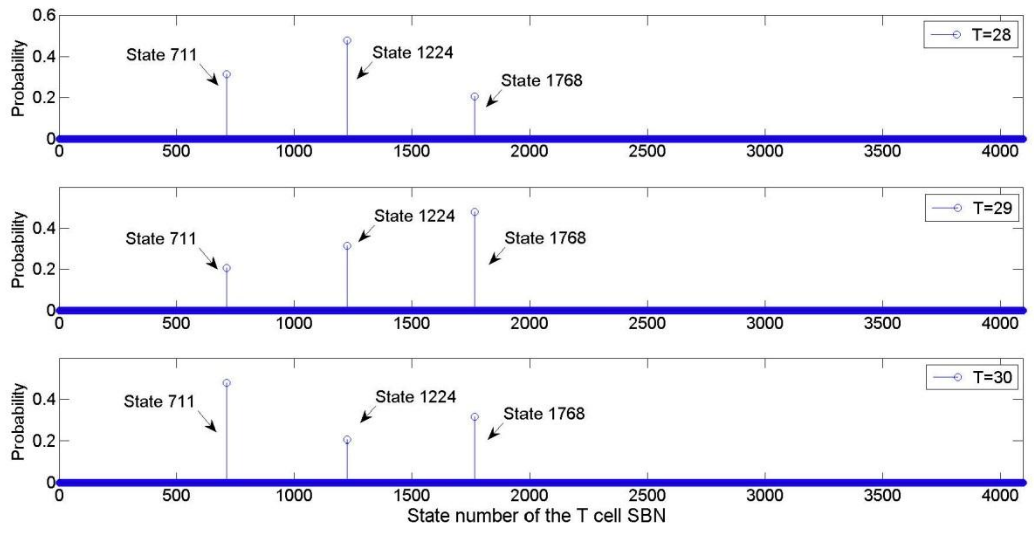


Figure 5.12. State distributions of the SBN in Figure 5.11 after 28, 29 and 30 clock cycles obtained using the time-frame expansion technique.

Alternatively, and more efficiently, the aforementioned time-frame expansion technique can be used to find the attractors with a greatly reduced complexity. The results are shown in Figure 5.12 for the same SBN simulation of 28, 29 and 30 cycles and the largest runtime is only 0.22s, compared to more than 70s by using the matrix-based analysis. It can be seen that the attractors match those in Table 5.11. These show the effectiveness and efficiency of the time-frame expansion technique.

5.3.2.4. Perturbation and Prediction

When the genes in a network are perturbed with a small probability, an SBN with perturbation can be constructed (as in Figure 5.3) for analyzing the stability of the network under perturbation. Since biological networks are usually robust and stable, the same attractors are expected to exist for the same network by a small perturbation. Assume that each gene is independently perturbed by a probability 0.01, Figure 5.13(a) shows the steady state distributions of the SBN with perturbation for the network of Figure 5.10.

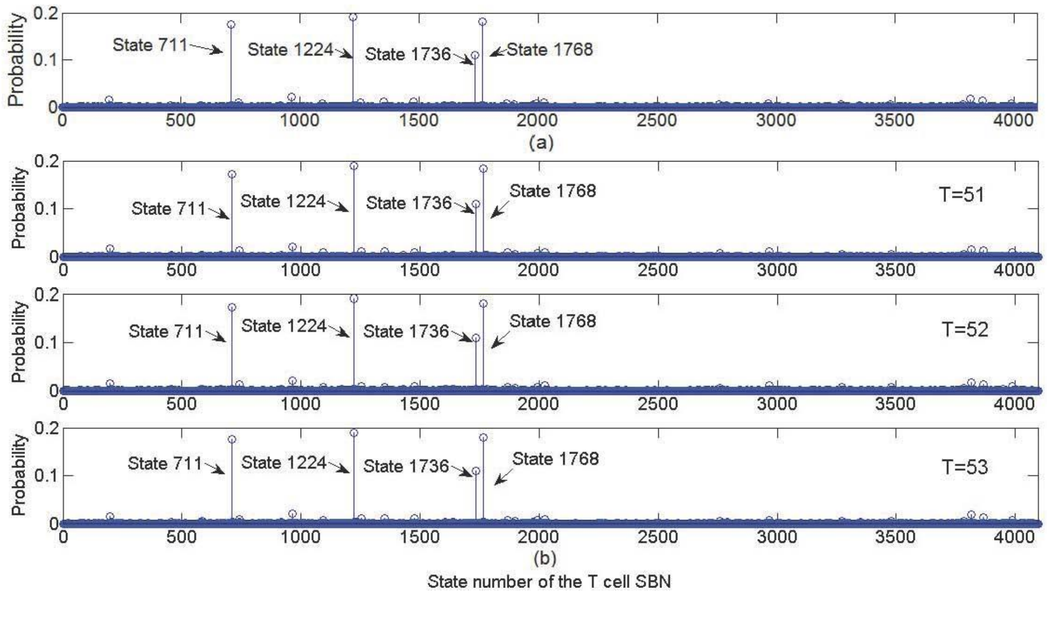


Figure 5.13. Steady state distribution of the T cell network with perturbation rate of 0.01: (a) computed using state transition matrices and (b) obtained using the time frame expansion technique.

It can be seen that the known Attractors 1, 2 and 3 (or, states 1224, 1768 and 711) are among those shown in Figure 5.13(a) with probability 0.1901, 0.1804 and 0.1750, respectively. What is interesting, however, is that pseudo-attractors, i.e., the attractors due to random gene perturbation, exist. These pseudo-attractors are listed in Table 5.12 with their steady state probabilities. It can be seen that most of the pseudo-attractors differ from the closest known attractor by only one gene. In particular, the most prominent pseudo-attractor, located at state 1736 with a probability larger than 0.1, differs from Attractor 2 or state 1768 by the expression of L-Myb12. L-Myb12 is a late response gene and plays an important role in the regulation of the T-cell network, so this result confirms the sensitivity of L-Myb12 in the regulatory behavior. Since biological experiments are not straightforward or easy to be implemented for investigating the T-cell network under perturbation, such study may provide insights into the understanding of potential physiological implications in a perturbed network. In a long run, this may be helpful in the development of genetic therapeutic methodologies.

Table 5.12. Pseudo-attractors found by the SBN with perturbation (perturbation probability = 0.01; state 1224 with probability 0.1901, state 1768 with probability 0.1804 and state 711 with probability 0.1750).

State Number	Probability	Closest attractor	Difference
1736	0.1099	Attractor 2	g(6) (L-Myb12)
967	0.0203	Attractor 3	g(9) (I-Rpol-hnr)
199	0.0164	Attractor 3	g(10) (E-stat5a)
3816	0.0147	Attractor 2	g(12) (L-Mcmd)
3866	0.0135	Different from all the attractors by more than 3 genes	
743	0.0120	Attractor 3	g(6) (L-Myb12)

1352	0.0101	Attractor 1	g(8) (E-Stat1-6)
			g(9) (I-Rpol-hnr)

1256	0.0100	Attractor 1	g(6) (L-Myb12)

Application of the time-frame expansion technique yields similar predictions for the network under perturbation. For a perturbation rate of 0.0001 and a threshold value of 0.001 for norm 2, it only takes 93.72s to obtain the steady state distribution using a sequence length of 100,000 bits. The simulation results are shown in Figure 5.13(b), which agree with those in Figure 5.13(a), so the time-frame expansion technique provides a highly efficient tool for analyzing the dynamics of a network with (and without) perturbation.

The proposed SBN technique is more efficient than a random sampling approach, due to the use of non-Bernoulli sequences of random permutations of fixed numbers of 1's and 0's in the representation of initial probabilities [20]. The time-frame expansion technique is also more efficient compared to the Markov Chain Monte Carlo (MCMC) method. It can be proved that the time-frame expanded SBN technique converges faster to a steady state than the MCMC method, because it requires a fewer number of clock cycles or time frames to converge and generates less pseudo-random numbers at each time frame. These indicate that the proposed SBN approach is more accurate and more efficient than a simple random sampling approach (such as the MC simulation) in the computation of state transition matrices and the evaluation of steady state distributions.

5.3.3 Relationship to Other GRN Models

5.3.3.1. Continuous Models

Continuous models based on linear or ordinary differential equations can potentially be implemented using SBNs, provided that the underlying principles of the differential equations can be formulated in state transition matrices. In this case, a network of *n* genes is modeled by:

$$\begin{bmatrix} \frac{dg_1}{dt} \\ \frac{dg_2}{dt} \\ \vdots \\ \frac{dg_n}{dt} \end{bmatrix} = \boldsymbol{T} \begin{bmatrix} g_1 \\ g_2 \\ \vdots \\ g_n \end{bmatrix}, \tag{5.12}$$

where g_i, (i = 1, 2, … n), indicates the level of a gene and $\boldsymbol{T}$ is a matrix of n rows and n columns. The entries in $\boldsymbol{T}$ are determined by factors such as the reaction rate constants. If the gene level can be expressed as the occurrence rate of a gene, denoted by p_i, (i = 1, 2, … n), which, for example, can be obtained by the ratio between the number of a particular type of genes and the total number of genes, then (5.12) can be expressed as:

$$\begin{bmatrix} \frac{dp_1}{dt} \\ \frac{dp_2}{dt} \\ \vdots \\ \frac{dp_n}{dt} \end{bmatrix} = \boldsymbol{T} \begin{bmatrix} p_1 \\ p_2 \\ \vdots \\ p_n \end{bmatrix}. \tag{5.13}$$

In an SBN, the next state of genes, $\boldsymbol{X_{t+1}}$, is determined by the current state, $\boldsymbol{X_t}$, and the state transition matrix, $\boldsymbol{A}$, i.e.,

$$\boldsymbol{X_{t+1}} = \boldsymbol{A}\boldsymbol{X_t}, \tag{5.14}$$

where $\boldsymbol{A}$ is a 2^nx2^n matrix, as given by (5.2). Then a new transition matrix of n rows and n columns, denoted by G, can be obtained by summarizing the entries in the rows and columns of $\boldsymbol{A}$, such that

$$\boldsymbol{P_{t+1}} = \boldsymbol{G}\boldsymbol{P_t}, \tag{5.15}$$

where $\boldsymbol{P_{t+1}}$ and $\boldsymbol{P_t}$ indicate the gene levels at two consecutive time steps. Further assume that

$$\Delta\boldsymbol{P} = \boldsymbol{P_{t+1}} - \boldsymbol{P_t}. \tag{5.16}$$

In the limit, we obtain:

$$\frac{d\boldsymbol{P}}{dt} = \frac{\boldsymbol{P_{t+1}} - \boldsymbol{P_t}}{dt} = \frac{\boldsymbol{G} - \boldsymbol{I}}{dt}\boldsymbol{P_t}, \tag{5.17}$$

where $\boldsymbol{I}$ is the identity matrix. Finally, (5.13) and (5.17) lead to

$$\boldsymbol{G} - \boldsymbol{I} = \boldsymbol{T} * dt, \tag{5.18}$$

which describes the relationship between the transition matrices in a continuous model and an SBN.

5.3.3.2. Single-Molecule Level Models

In a single-molecule level model, significant stochastic effects of biochemical reactions are accounted for each molecular species. The stochastic simulation algorithm (SSA) tracks the number of molecular species in a biochemical system, so it accurately simulates the discrete, random biochemical reactions specified by the chemical master equation (CME) [61, 62]. Essentially, the SSA follows a discrete Markov process, in which two values are generated from two independent random variables at each time step. The first value predicts when the next reaction will occur and the second decides which reaction will occur. In order to characterize the evolution of the system, repeated trials are required to perform, which leads to a significant run time for simulating a large network.

Due to the same underlying Markov models in the SSA and PBNs, the SSA can, in principle, be implemented using SBNs. However, this implementation is not straightforward as the SSA simulates the function of the CME while the SBN implements the state transitions of Boolean functions. A challenge is therefore to formulate the underlying principles of the CME in the form of state transition matrices. Nevertheless, it is possible for the SSA and SBN to be used in a hybrid method. In this method, a logical model is first used to simulate a large network and to identify the sensitive nodes in the network. Then, a single-molecule level model such as the SSA can be used to find out more details of the identified sensitive genes. In this way, this hybrid method leverages the efficiency of a logical model and the accuracy of a single-molecule level model, so it may provide an effective means to model large gene regulatory networks.

5.3.4 Application on GRN Analysis

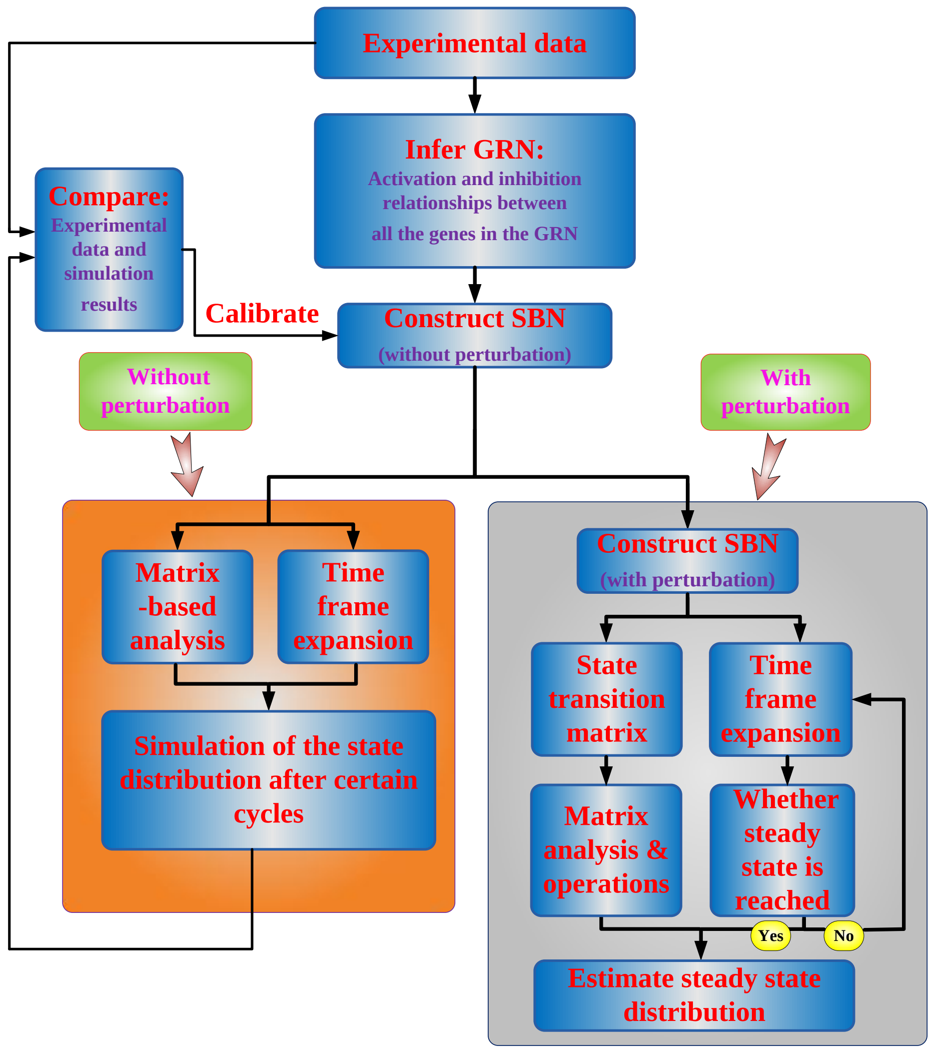

Experimental data
Infer GRN:
Activation and inhibition relationships between all the genes in the GRN
Compare:
Experimental data and simulation results
Calibrate
Construct SBN
(without perturbation)
Without perturbation
With perturbation
Matrix -based analysis
Time frame expansion
Simulation of the state distribution after certain cycles
Construct SBN
(with perturbation)
State transition matrix
Time frame expansion
Matrix analysis & operations
Whether steady state is reached
Yes
No
Estimate steady state distribution

Figure 5.14. A flowchart for the application of the SBN approach in GRN analysis.

In summary, for a GRN inferred from microarray time series data, an SBN can be constructed to analyze the dynamics of the network with or without gene perturbation. This provides the biologists an efficient means to evaluate the steady state distribution of a genetic network. A general procedure for applying the proposed SBN approach in a GRN analysis is given in the flowchart of Figure 5.14.

5.4. Summary

This chapter proposes a novel structure of stochastic Boolean networks (SBNs) for an accurate and efficient implementation of probabilistic Boolean networks (PBNs). The application of an SBN is demonstrated through the computation of the state transition matrix and the steady-state analysis of a PBN. The state transition matrix can be accurately and efficiently computed in an SBN with a complexity of $O(nL2^n)$, where n is the number of genes in a PBN and L is a factor determined by the stochastic sequence length. Since the required minimum sequence length for a given evaluation accuracy usually increases slower with n than the number of Boolean networks, i.e., N, L is typically much smaller than N, especially in a network with a large number of genes. This result is an improvement compared to the previous results of $O(nN2^{2n})$ and $O(nN2^n)$. The steady state can be estimated using the obtained state transition matrix or the time-frame expansion technique. The latter approach has shown a significant speedup in the computation of the steady state distribution.

SBNs have been constructed for the p53-Mdm2 network and an inferred T cell immune response network. Simulations of the SBNs have recovered the state dynamics that have been experimentally demonstrated for these two networks. The proposed approach is able to discover network dynamics when the genes are under perturbation, which is a task difficult to implement in experiments or by other modeling approaches due to its complexity. So in this case, the SBN technique can be used to provide biologically meaningful insights for a first understanding of the dynamics of a GRN. The relationship between an SBN and continuous/stochastic models has also been discussed and a hybrid approach may be useful in a more efficient modelling of a large GRN. Finally, the SBN approach is able to account for signalling pathway information [95], so it may provide an effective solution to the modeling of complex genetic networks.

CHAPTER 6

Conclusions

Based on applications in nanoelectronic circuits and biological networks, the efficiency and effectiveness of probabilistic and stochastic computation have been shown in this thesis.

The aggressive scaling of complementary metal-oxide-semiconductor (CMOS) technology has resulted in small device dimensions and low tolerance to design and process variations, thus having a negative impact on the reliability of digital circuits [1]. New failure modes have been observed due to high integration and device fabrication effects, such as the time-dependent dielectric breakdown of materials, hot carrier injection and negative bias temperature instability in transistors. In addition to permanent defects, soft errors have also become a concern as the temporary interference by noisy environments affects the reliable operation of nanometric digital circuits. High integration densities and low voltage/current thresholds have increased the soft error rates (SERs) of circuits and systems.

The approach proposed in [17] and Chapter 2 in this thesis is able to characterize and assess the reliable operation of a sequential circuit through a detailed analysis of its state transition matrices (STMs). The dependency of circuit reliability on its input distribution and sequences is revealed due to the masking of errors. Error masking in a sequential circuit refers to the logic masking effect imposed on the feedback signals by specific combinations of primary inputs (referred to as restoring inputs). As a result, the presumably monotonically decreasing reliability of a sequential circuit can actually be interrupted and restored by the primary inputs. The restoring inputs are equivalent to the synchronizing sequences in switching theory [96, 97], that have been extensively used to facilitate testing of sequential circuits [98]. Both experimental and theoretical approaches have been used to compute synchronizing sequences for testing an FSM [99, 100].

Error masking is theoretically analyzed using the STMs in a finite state machine (FSM) model as a mathematical framework. To alleviate the complexity issues in the STM computation, an efficient approach using binary decision diagrams (BDDs) is further employed for analyzing error masking in large circuits in [17] and Chapter 2 in this thesis. Simulation results are presented to show that single and multiple step restoring inputs can be found by the proposed approach. Future work is to complement this methodology and related observations with timing and electrical information for formulating error mitigation schemes for nanoscale systems.

Methodologies for inexact (or soft) computing rely on the feature that many applications can tolerate some loss of precision and therefore, the solution can tolerate some degree of uncertainty [101, 102]. Deterministic, explicit, and precise models and algorithms are not always suitable to solve these types of problems. However, inexact computing applications are mostly implemented using digital binary logic circuits, thus operating with a high degree of predictability and precision. A framework based on a precise and specific implementation can still be used with a methodology that intrinsically has a lower degree of precision and an increasing uncertainty in operation. While this may be viewed as a potential conflict, such an approach tailors the significant advantage of inexact computing (and its inherent tolerance to some imprecision and uncertainty) to a technology platform implemented by conventional digital logic and systems [102]. The paradigm of inexact computation relies on relaxing fully precise and completely deterministic building blocks (such as a full adder) when for example, implementing bio-inspired systems. This allows nature-inspired computation to redirect the existing design process of digital circuits and systems by taking advantage of a decrease in complexity and cost with possibly a potential increase in performance and power efficiency [101, 102].

One of the fundamental arithmetic operations in many applications of inexact computing is addition [103, 104]. Soft additions are generally based on the operation of deterministic approximate logic or probabilistic imprecise arithmetic (categorized in [105] as design-time and run-time techniques). Several recently proposed adder architectures are representatives of these types. To evaluate the effectiveness of these architectures, new design metrics are urgently needed. However, the traditional metric of reliability (defined as the probability of system survival) is not appropriate for use in evaluating deterministic approximate designs. Therefore new metrics for assessing adder designs with respect to reliability and power efficiency for inexact computing are proposed in [18, 19] and Chapter 3 in this thesis. A new figure of merit, referred to as error distance (ED), is proposed to characterize the reliability of an output of an adder. ED is then used to obtain two new metrics: the mean error distance (MED) and the normalized error distance (NED). The MED and NED can be obtained using sequential probability transition matrices (SPTMs) and are able to evaluate the reliability of both probabilistic and deterministic adders. NED is a stable metric that is almost independent of the size of an implementation; this feature brings a new perspective for the evaluation and comparison of different adder implementations. The power and NED product is further used to evaluate the power and precision tradeoff. An adder implementation with reduced precision, referred to as the lower-bit ignored adder (LIA), is investigated as a baseline for assessing the lower-bit OR adder (LOA), approximate mirror adders (AMAs) and probabilistic full adders (PFAs). A detailed analysis and simulation results are presented to assess the reliable performance of these adders using the proposed new metrics.

As CMOS approaches physical and technological limits, new devices have been proposed to implement nanoscale architectures, such as the multiple-valued logic (MVL) operation with a base higher than two. MVL allows for more than two levels of logic; depending on the number of levels, ternary (base 3) and quaternary (base 4) logic have been advocated for different applications. MVL enjoys many advantages over its binary counterpart; for example, each wire can transmit more information than binary, so the number of connections in a chip can be reduced, thus decreasing circuit complexity. However, MVL circuits are subject to issues such as low noise margins.

Recently, carbon nanotube field-effect transistors (CNTFETs) have been extensively studied as a potential alternative to the silicon-based metal–oxide–semiconductor field effect transistors (MOSFETs) for implementing MVL circuits [48, 49]. The resistor-loaded designs utilize fewer transistors to implement MVL gates, but the off-chip resistors and also large static power consumptions limit the integration and applications [49]. The complementary designs, which can be fully integrated, consume little static power but use more transistors [48]. In order to make a trade-off between static power consumption and area cost (i.e., the number of transistors), a pseudo-complementary CNTFETs-based MVL design is proposed in [21] and also Chapter 4 in this thesis.

Similar to nanoscale CMOS circuits, CNTFETs-based MVL circuits are affected by manufacturing variations and noise, so their operation is probabilistic and subject to errors. Therefore, the analysis of reliability of MVL circuits is of significant concern. A number of approaches have been proposed for the reliability evaluation of binary circuits. However, no approach has been proposed for the reliability analysis of MVL gates before. In particular, the structure and topologies of MVL gates need to be taken into consideration in an accurate evaluation approach. Hence, a transistor-level analysis is highly desirable because it can provide a better assessment of the gate structure as well as the error susceptibility of a particular implementation. For this process to be viable, it is important to efficiently evaluate the reliability though a simple, yet efficient method to provide insight on reliability as well as its enhancements. Stochastic computational models (SCMs) for MVL are developed for evaluating the reliability of gates in [21] and also Chapter 4 in this thesis; the applicability of these models to circuits is briefly treated through an illustrative example.

As discussed previously, biological systems also present some probabilistic behaviors because of noise or perturbation. In recent years, various computational models have been of interest due to their use in the modeling of gene regulatory networks (GRNs). As a logical model, probabilistic Boolean networks (PBNs) consider molecular and genetic noise, so the study of PBNs provides significant insights into the understanding of the dynamics of GRNs, which will ultimately lead to advances in developing therapeutic methods that intervene in the process of disease development and progression. The applications of PBNs, however, are hindered by the complexities involved in the computation of the state transition matrix and the steady-state distribution of a PBN. For a PBN with n genes and N Boolean networks, the complexity to compute the state transition matrix is $\mathrm{O}(nN2^{2n})$ or $\mathrm{O}(nN2^{n})$ for a sparse matrix.

Stochastic Boolean networks (SBNs) are proposed as an efficient approach to modeling gene regulatory networks (GRNs) in [22] and also Chapter 5 in this thesis. The state transition matrix is computed in an SBN with a complexity of $\mathrm{O}(nL2^{n})$, where L is a factor related to the stochastic sequence length. Since the minimum sequence length required for obtaining an evaluation accuracy approximately increases linearly with the size of a network, L is typically smaller than *N*, which is usually on an exponential order of *n*, especially in a network with a large number of genes. Hence, the computational complexity of an SBN is primarily limited by the number of genes, but not directly by the total possible number of Boolean networks. Furthermore, a time-frame expanded SBN enables an efficient analysis of the steady-state distribution of a PBN. These findings are supported by the simulation results of a simplified p53 network, several randomly generated networks and a network inferred from a T cell immune response dataset. An SBN can also implement the function of an asynchronous PBN and is potentially useful in a hybrid approach in combination with a continuous or single-molecule level stochastic model. The SBN approach is able to recover biologically-proven regulatory behaviors and also further predict the network dynamics when the genes are under perturbation. The proposed algorithms and methods in [22, Chapter 5] have been implemented in Matlab packages and can be applied in the modeling of a general GRN.

BIBLIOGRAPHY

[1] S. Borkar, “Designing Reliable Systems from Unreliable Components: The Challenges of Transistor Variability and Degradation,” in IEEE Micro, vol. 25, no. 6, pp. 10-16, Nov. 2005.

[2] S. Krishnaswamy, G. F. Viamontes, I. L. Markov, and J. P. Hayes, “Probabilistic transfer matrices in symbolic reliability analysis of logic circuits,” ACM Trans. Des. Autom. Electron. Syst., vol. 13, no. 1, pp. 1–35, 2008.

[3] J. Han, H. Chen, E. Boykin, J. Fortes, "Reliability evaluation of logic circuits using probabilistic gate models," Microelectronics Reliability, vol. 51, no. 2, 2011, pp. 468-476.

[4] A. Abdollahi, “Probabilistic Decision Diagrams for Exact Probabilistic Analysis,” Proc. Int’l Conference on Computer Aided Design, 2007.

[5] T. Rejimon and S. Bhanja, “Scalable probabilistic computing models using Bayesian networks,” in Proc. Int. Midwest Symp. Circuits Syst., 2005, pp. 712–715.

[6] N. Mohyuddin, E. Pakbaznia, M. Pedram, "Probabilistic Error Propagation in Logic Circuits Using the Boolean Difference Calculus," in IEEE Intl. Conf. on Comp. Des., Lake Tahoe, CA, USA, pp. 7-13 (2008)

[7] S. Sivaswamy, K. Bazargan, M. Riedel, "Estimation and Optimization of Reliability of Noisy Digital Circuits," in International Symposium on Quality Electronic Design, San Jose, CA, USA, pp. 213-219 (2009)

[8] M. R. Choudhury and K. Mohanram, “Reliability analysis of logic circuits,” IEEE TCAD, vol. 28, no. 3, pp. 392–405, March 2009.

[9] H. Chen, J. Han, "Stochastic Computational Models for Accurate Reliability Evaluation of Logic Circuits," Proc. Great Lakes Symp. VLSI (GLVLSI), Providence, RI, USA, 2010.

[10] International Technology Roadmap for Semiconductors, Process Integration, Devices, and Structure. ITRS 2009 edition. [Online] Available: http://www.itrs.net/Links/2011ITRS/2011Chapters/2011PIDS.pdf. Last accessed on Aug. 1st, 2012.

[11] Rejimon and S. Bhanja, “Scalable probabilistic computing models using Bayesian networks,” in Proc. Int. Midwest Symp. Circuits Syst., 2005, pp. 712–715.

[12] N. Mohyuddin, E. Pakbaznia, M. Pedram, "Probabilistic Error Propagation in Logic Circuits Using the Boolean Difference Calculus," in IEEE Intl. Conf. on Comp. Des., Lake Tahoe, CA, USA, pp. 7-13 (2008)

[13] S. Sivaswamy, K. Bazargan, M. Riedel, "Estimation and Optimization of Reliability of Noisy Digital Circuits," in International Symposium on Quality Electronic Design, San Jose, CA, USA, pp. 213-219 (2009)

[14] M. R. Choudhury and K. Mohanram, “Reliability analysis of logic circuits,” IEEE TCAD, vol. 28, no. 3, pp. 392–405, March 2009.

[15] P. Shivakumar, M. Kistler, S.W. Keckler, D. Burger, L. Alvisi, “Modeling the Effect of Technology Trends on the Soft Error Rate of Combinational Logic”, In Proc. Intl. Conf on Dependable Systems &Networks. 2002, pp. 389-398.

[16] B. R. Gaines, “Stochastic Computing”, Spring Joint Computer Conf., 1967, Vol. 30, pp. 149-156.

[17] J. Liang, J. Han, F. Lombardi: "Analysis of Error Masking and Restoring Properties of Sequential Circuits" IEEE Transactions on Computers. (Accepted).

[18] J. Liang, J. Han, F. Lombardi: “On the Reliable Performance of Sequential Adders for Soft Computing” IEEE int’l Symposium on Defect and Fault Tolerant in VLSI systems (DFT), Vancouver, BC, Canada, pp. 3-10, 2011.

[19] J. Liang, J. Han, F. Lombardi: “New Metrics for the Reliability of Approximate and Probabilistic Adders” IEEE Transactions on Computers. (Accepted).

[20] H. Chen, J. Liang, J. Han and F. Lombardi, “A Stochastic Computational Approach for Accurate and Efficient Reliability Evaluation,” IEEE Transactions on Computers. (Revised).

[21] J. Liang, J. Han, F. Lombardi: "Design and Reliability Analysis of Multiple Valued Logic Gates using Carbon Nanotube FETs" IEEE/ACM International Symposium on Nanoscale Architectures, the Netherland, pp. 131-138, 2012.

[22] J. Liang, J. Han. "Stochastic Boolean Networks: An Efficient Approach to Modeling Gene Regulatory Networks" BMC Systems Biology (Accepted).

[23] G. D. Hachtel and F. Somenzi, Logic Synthesis and Verification Algorithms, Springer; 1 edition. Jun 30, 1996.

[24] R. Das and J. P. Hayes, "Monitoring Transient Errors in Sequential Circuits," pp.319-322, 16th IEEE Asian Test Symposium (ATS 2007), 2007.

[25] R. Das and J. P. Hayes, “Recovery from Transition Errors in Sequential Circuits” Design and Test Workshop, 2007. IDT 2007. 2nd International. Cario, Egypt, pp. 105-110.

[26] S. Krishnaswamy, G. F. Viamontes, I.L. Markov and J. P. Hayes: “Probabilistic transfer matrices in symbolic reliability analysis of logic circuits.” ACM Trans. on Design Automation of Electronic Systems, vol. 13, article 8, Jan. 2008.

[27] C-C Yu and J. P. Hayes, “Scalable and accurate estimation of probabilistic behavior in sequential circuits” IEEE VLSI Test Symposium (VTS), 2010 28th. Santa Cruz, CA. pp. 165 – 170.

[28] G. D. Hachtel, E. Macii, A. Pardo, and F. Somenzi, “Markovian Analysis of Large Finite State Machines,” In IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD), Vol. 15, No. 12, pp. 1479-1493, December 1996.

[29] J. Han and P. Jonker, “A System Architecture Solution for Unreliable Nanoelectronic Devices,” IEEE Trans. Nanotechnology, vol. 1, no. 4, pp. 201-208, December 2002.

[30] J. Han and P. Jonker, “A defect- and fault-tolerant architecture for nanocomputers,” Nanotechnology, vol. 14, no. 2, pp. 224–230, 2003.

[31] J. Han, J. Gao, Y. Qi, P. Jonker, J.A.B. Fortes. “Toward Hardware-Redundant, Fault-Tolerant Logic for Nanoelectronics," IEEE Design and Test of Computers, July/August 2005, vol. 22, no. 4, 328-339.

[32] R. E. Bryant,. Graph-Based Algorithms for Boolean Function Manipulation. Computers, IEEE Transactions on. Aug. 1986 pp.677 – 691

[33] F. Somenzi. CUDD: CU decision diagram package release 2.4.2. http://vlsi.colorado.edu/~fabio/CUDD/. (2009).

[34] R. E. Bryant, “Symbolic Boolean Manipulation with Ordered Binary Decision Diagrams,” ACM Computing Surveys, Vol. 24, No. 3 (September, 1992), pp. 293-318

[35] N. Miskov-Zivanov, D. Marculescu, “Modeling and Optimization for Soft-Error Reliability of Sequential Circuits,” In IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD), Vol. 27, Issue. 5, pp. 803-816, 2008

[36] J. M. Rabaey, A. Chandrakasan and B. Nikolic, “Digital Integrated Circuits; a design perspective,” Second Edition, Pearson Education, 2003.

[37] H. R. Mahdiani, A. Ahmadi, S. M. Fakhraie, C. Lucas, “Bio-Inspired Imprecise Computational Blocks for Efficient VLSI Implementation of Soft-Computing Applications,” IEEE Transactions on Circuits and Systems I: Regular Papers, vol. 57, no. 4, pp. 850-862, April 2010.

[38] A. K. Verma, P. Brisk, P. Ienne, “Variable latency speculative addition: A new paradigm for arithmetic circuit design,” DATE, pp. 1250-1255, 2008.

[39] N. Zhu, W. L. Goh, K. S. Yeo, “An enhanced low-power high-speed adder for error tolerant application,” ISIC, pp. 69-72, 2009.

[40] V. Gupta, D. Mohapatra, S. P. Park, A. Raghunathan, K. Roy, “IMPACT: IMPrecise adders for low-power approximate computing” Low Power Electronics and Design (ISLPED) 2011 International Symposium on. 1-3 Aug. 2011.

[41] J. Kim, S. Tiwari “Inexact computing for ultra low-power nanometer digital circuit design”, in IEEE/ACM International Symposium on Nanoscale Architectures (NANOARCH), 2011, pp. 24-31.

[42] K.V. Palem, “Energy Aware Computing through Probabilistic Switching: A Study of Limits,” IEEE Trans. Computers, 2005, pp. 1123-1137.

[43] P. Kulkarni, P. Gupta and M. Ercegovac. "Trading Accuracy for Power with an Underdesigned Multiplier Architecture," Proc., IEEE/ACM International Conference on VLSI Design, Mar 31, 2011.

[44] B. Brown and H. Card, "Stochastic neural computation I: Computational elements," IEEE Tran. Computers, vol. 50, pp. 891–905, Sept. 2001.

[45] J. von Neumann, "Probabilistic logics and the synthesis of reliable organisms from unreliable components," Automata Studies, Shannon C.E. & McCarthy J., eds., Princeton University Press, pp. 43-98, 1956.

[46] Jie Han, "Fault-Tolerant Architectures for Nanoelectronic and Quantum Devices", Universal Press, Veenendaal, The Netherlands, 2004. A Ph.D. dissertation of the Delft University of Technology, 1-135. ISBN: 90-9018888-6

[47] S. J. Tans, A. R. M. Verschueren, and C. Dekker, "Room-temperature transistor based on a single carbon nanotube", Nature 393, 49-52, May, 1998.

[48] S. Lin, Y. Kim and F. Lombardi, "CNTFET-Based Design of Ternary Logic Gates and Arithmetic Circuits" IEEE Transactions on Nanotechnology, Vol. 10, No. 2, pp. 217-225, March 2011.

[49] A. Raychowdhury and K. Roy, "Carbon-nanotube-based voltage-mode multiple-valued logic design," IEEE Trans. Nanotechnol., vol. 4, no. 2, pp. 168–179, Mar. 2005.

[50] G. Zhang, et al., "Selective Etching of Metallic Carbon Nanotubes by Gas-Phase Reaction", Science, Vol. 314, pp. 974 –977, 2006.

[51] A. Lin, N. Patil, J. Zhang, H. Wei, S. Mitra and H.-S.P. Wong, "ACCNT - A Metallic-CNT-Tolerant Design Methodology for Carbon Nanotube VLSI: Analysis and Design Guidelines," IEEE Trans. Electron Devices, 2010.

[52] P. Zarkesh-Ha, A. A. M. Shahi, "Stochastic Analysis and Design Guidelines for CNFETs in Gigascale Integrated Systems," Electron Devices, IEEE Transactions on , vol.58, no.2, pp.530-539, Feb. 2011

[53] R. A. Weinberg, "The Biology of Cancer. Garland Science", 1 edition. 2006.

[54] Vogelstein, D. Lane, A. J. Levine, "Surfing the p53 network", Nature 2000, 408:307–310.

[55] G. Lahav, N. Rosenfeld, A. Sigal, N. Geva-Zatorsky, A. J. Levine, M. B. Elowitz, and U. Alon, "Dynamics of the p53-Mdm2 feedback loop in individual cells", Nature Genetics 2004, 36, 147 – 150.

[56] E. Batchelor, A. Loewer, and G. Lahav, "The ups and downs of p53: understanding protein dynamics in single cells", Nature Reviews Cancer 2009, 371-377.

[57] H. de Jong, "Modeling and simulation of genetic regulatory systems: a literature review", Journal of Computational Biology. January 2002, 9(1): 67-103. doi:10.1089/10665270252833208.

[58] G. Karlebach, R. Shamir, "Modelling and analysis of gene regulatory networks", Nature Reviews Molecular Cell Biology. Volume 9, Oct. 2008.

[59] S. A. Kauffman, "Metabolic stability and epigenesis in randomly constructed genetic nets", J. Theor. Biol. 1969, 22, 437-467.

[60] E. Klipp, "Systems Biology In Practice: Concepts, Implementation And Application", Wiley-VCH, Weinheim, 2005.

[61] D. T. Gillespie, "A general method for numerically simulating the stochastic time evolution of coupled chemical reactions", J. Comput. Phys. 22, 403. 1976.

[62] D. T. Gillespie, "Exact stochastic simulation of coupled chemical reactions", J. Phys. Chem. 81, 2340-61. 1977.

[63] M. Gibson, and J. Bruck, "Efficient exact stochastic simulation of chemical systems with many species and many channels", J. Phys. Chem. 104, 1876–1889. 1999.

[64] D. T. Gillespie, "Approximate accelerated stochastic simulation of chemically reacting systems", J. Chem. Phys. 115, 1716-1733. 2001.

[65] S. Pandey, R. Wang, L. Wilson, S. Li, Z. Zhao, T. Gookin, S. Assmann, and R. Albert, "Boolean modeling of transcriptome data reveals novel modes of heterotrimeric G-protein action", Molecular Systems Biology, Article number 372; doi:10.1038/msb.2010.28

[66] I. Shmulevich, E. R. Dougherty, W. Zhang, "From Boolean to probabilistic Boolean networks as models of genetic regulatory networks", Proceedings of IEEE, 2002, 90, pp.1778-1792.

[67] I. Shmulevich, E. R. Dougherty, S. Kim, W. Zhang, "Probabilistic Boolean networks: a rule-based uncertainty model for gene regulatory networks", Bioinformatics, 2002, 18: 261-274.

[68] I. Shmulevich, E. R. Dougherty, "Probabilistic Boolean Networks: The Modeling and Control of Gene Regulatory Networks", Society for Industrial & Applied Mathematics, U.S., 2010.

[69] I. Shmulevich, E. R. Dougherty, W. Zhang, "Gene perturbation and intervention in probabilistic Boolean networks", Bioinformatics, 2002, 18 (10):1319-1331.

[70] I. Shmulevich, I. Gluhovsky, R. F. Hashimoto, E. R. Dougherty, W. Zhang, "Steady-state analysis of genetic regulatory networks modelled by probabilistic Boolean networks", Comparative and Functional Genomics, 2003, 4: 601–608. doi: 10.1002/cfg.342

[71] E. R. Dougherty, R. Pal, X. Qian, M. L. Bittner, A. Datta, "Stationary and Structural Control in Gene Regulatory Networks: Basic Concepts", International Journal of Systems Science, 2010, Vol. 41, No. 1, 5-16.

[72] B. Faryabi, G. Vahedi, A. Datta, J F. Chamberland, E. R. Dougherty, "Recent Advances in Intervention in Markovian Regulatory Networks", Curr Genomics. 2009, 10(7): 463–477.

[73] G. Karlebach, R. Shamir, "Minimally perturbing a gene regulatory network to avoid a disease phenotype: the glioma network as a test case", BMC Systems Biology 2010, 4:15.

[74] S. Zhang et al. “Simulation study in probabilistic Boolean network models for genetic regulatory networks”, Int. J. Data Min. 2007, 1:217-240.

[75] W. Ching et al. “An approximation method for solving the steady-state probability distribution of probabilistic Boolean networks”, Bioinformatics, 2007, 23 pp. 1511–1518.

[76] I. Ivanov, R. Pal, E.R. Dougherty, “Dynamics Preserving Size Reduction Mappings for Probabilistic Boolean Networks”, IEEE Transactions on Signal Processing, 2007, Vol. 55, No. 5, 2310-2322.

[77] X. Qian, I. Ivanov, N. Ghaffari, E. R. Dougherty, “Intervention in gene regulatory networks via greedy control policies based on long-run behavior”, BMC System Biology, 2009, 3:61.

[78] X. Qian, N. Ghaffari, I. Ivanov, E. R. Dougherty, “State reduction for network intervention in probabilistic Boolean networks”, Bioinformatics, 2010, 26 (24): 3098-3104.

[79] R. Pal, “Context-Sensitive Probabilistic Boolean Networks: Steady-State Properties, Reduction, and Steady-State Approximation”, Signal Processing, IEEE Transactions on, vol.58, no.2, pp.879-890, Feb. 2010.

[80] B. Faryabi, J-F. Chamberland, G. Vahedi, A. Datta, E. R. Dougherty, “Optimal intervention in asynchronous genetic regulatory networks”, IEEE Journal of Selected Topics in Signal Processing. 2(3):412-23. 2008.

[81] B. Faryabi, G. Vahedi, J. F. Chamberland, A. Datta, E. R. Dougherty, “Intervention in context-sensitive probabilistic Boolean networks revisited”, EURASIP Journal on Bioinformatics Systems Biology. 2009.

[82] H.H. McAdams, L. Shapiro, “Circuit simulation of genetic networks”, Science 1995, 269 (5224), 650.

[83] A. Abdi, M. B. Tahoori, E. S. Emamian, “Fault diagnosis engineering of digital circuits can identify vulnerable molecules in complex cellular pathways”, Sci Signal. 2008 Oct 21;1(42):ra10.

[84] G. Kervizic, L. Corcos, “Dynamical modeling of the cholesterol regulatory pathway with Boolean networks”, BMC Systems Biology 2008, 2:99.

[85] R. Adar, Y. Benenson, G. Linshiz, A. Rosner, N. Tishby, E. Shapiro, “Stochastic computing with biomolecular automata”, PNAS July 6, 2004 vol. 101 no. 27 9960-9965.

[86] Y. Benenson, B. Gil, U. Ben-Dor, R. Adar, E. Shapiro, “An autonomous molecular computer for logical control of gene expression”, Nature, 2004, 429, 423-429.

[87] D. Wittmann, J. Krumsiek, J. Saez-Rodriguez, D. Lauffenburger, S. Klamt, F. Theis, “Transforming Boolean models to continuous models: methodology and application to T-cell receptor signaling”, BMC Systems Biology, September 2009.

[88] N. Duarte, S. Becker, N. Jamshidi, I. Thiele, Mo, M., Vo, T., Srivas, R. and Palsson, B.: Global reconstruction of the human metabolic network based on genomic and bibliomic data. PNAS, vol. 104, no. 6, pp. 1777–1782, February 6, 2007.

[89] H. Lähdesmäki et al. "Relationships between probabilistic Boolean networks and dynamic Bayesian networks as models of gene regulatory networks", Signal Processing 2006, 86:814-834.

[90] N. Guelzim, S. Bottani, P. Bourgine, F. Kepes, "Topological and causal structure of the yeast transcriptional regulatory network", Nature, 2002, Genetics 31, 60–63.

[91] S. Martin, Z. Zhang, A. Martino, J-L. Faulon, "Boolean dynamics of genetic regulatory networks inferred from microarray time series data", Bioinformatics, 2007, 23(7): 866-874.

[92] S. Marshall, L. Yu, Y. Xiao, ER. Dougherty, "Inference of a probabilistic boolean network from a single observed temporal sequence", EURASIP Journal on Bioinformatics and Systems Biology. 2007:32454.

[93] T. Akutsu et al. "Inferring qualitative relations in genetic networks and metabolic pathways", Bioinformatics, 16, 727–734. 2000.

[94] K. Basso et al. "Reverse engineering of regulatory networks in human B cells", Nat. Genet., 37, 382–390. 2005.

[95] R. K. Layek, A. Datta, E. R. Dougherty, "From biological pathways to regulatory networks", Mol. BioSyst., 2011, 7, 843-851.

[96] E.F. Moore, "Gedanken-experiments on sequential machines," in Automata Studies. C.E. Shannon and J. McCarthy, Eds., Princeton University Press, 1956, pp129-153.

[97] Z. Kohavi and N.K. Jha, "Switching and Finite Automata Theory", Cambridge University Press, 2010.

[98] I. Pomeranz, S.M. Reddy, "On removing redundancies from synchronous sequential circuits with synchronizing sequences," Computers, IEEE Transactions on. Jan. 1996, pp. 20 – 32.

[99] I. Pomeranz, S.M. Reddy, "On Synchronizable Circuits and Their Synchronizing Sequences," in IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD), Vol. 19, No. 9, pp. 1086-1092, September 2000.

[100] C. Pixley, S.-W. Jeong, and G. D. Hachtel, "Exact calculation of synchronizing sequences based on binary decision diagrams," in IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD), Vol. 13, No. 8, pp. 1024-1034, August 1994

[101] Y. Dote, and S.J. Ovaska, "Industrial Applications of Soft Computing: a Review," Proc. IEEE, Vol. 89, no. 9, pp. 1243-1265, 2001.

[102] R. Hegde and N.R. Shanbhag, "Soft digital signal processing," IEEE Trans. VLSI Syst., vol. 9, no. 6, pp. 813–823, 2001.

[103] V. Beiu, S. Aunet, J. Nyathi, R.R. Rydberg III and W. Ibrahim, “Serial Addition: Locally Connected Architectures,” IEEE Trans. Circ. and Sys. I, vol. 54, no. 11, Nov. 2007, pp. 2564–2579.

[104] S. Cotofana, C. Lageweg, S. Vassiliadis, “Addition related arithmetic operations via controlled transport of charge,” IEEE Trans. Computers, , vol. 54, no. 3, pp. 243-256, March 2005.

[105] J. Huang and J. Lach, “Exploring the Fidelity-Efficiency Design Space using Imprecise Arithmetic,” ASPDAC, 2011.